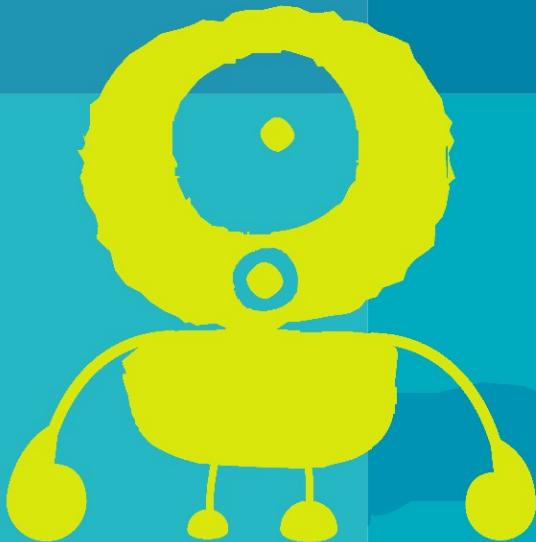


PETER HUBWIESER

# Didaktik der Informatik

3. AUFLAGE



eXamen.press

 Springer

eXamen.press

**eXamen.press** ist eine Reihe, die Theorie und Praxis aus allen Bereichen der Informatik für die Hochschulausbildung vermittelt.

Peter Hubwieser

# Didaktik der Informatik

Grundlagen, Konzepte, Beispiele

3., überarbeitete und erweiterte Auflage

Mit 91 Abbildungen und 68 Tabellen

Peter Hubwieser

Technische Universität München  
Fakultät für Informatik  
Boltzmannstr. 3  
85748 Garching

Peter.Hubwieser@in.tum.de  
<http://ddi.in.tum.de>

Bibliografische Information der Deutschen Bibliothek  
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Ursprünglich erschienen in der Springer-Lehrbuch Reihe

ISBN-13 978-3-540-72477-3 Springer Berlin Heidelberg New York

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wege und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zu widerhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

Springer ist ein Unternehmen von Springer Science+Business Media  
[springer.de](http://springer.de)

© Springer-Verlag Berlin Heidelberg 2000, 2001, 2004 und 2007

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften. Text und Abbildungen wurden mit größter Sorgfalt erarbeitet. Verlag und Autor können jedoch für eventuell verbliebene fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Satz und Herstellung: LE-TeX, Jelonek, Schmidt & Vöckler GbR, Leipzig  
Umschlaggestaltung: KünkelLopka Werbeagentur, Heidelberg

Gedruckt auf säurefreiem Papier 176/3180 YL - 5 4 3 2 1 0

*Für Anna, Katharina, Elena und Teresa*

# Vorwort zur ersten Auflage

Dieses Buch entstand aus den Vorlesungen über „Didaktik der Informatik“, die ich in den vergangenen vier Wintersemestern an der Technischen Universität München abgehalten habe. Im Lauf der Jahre hat sich die Zielsetzung dieser Veranstaltung stark verändert. Anfangs war sie in Ermangelung von Lehramtsstudenten mit dem Fach Informatik vor allem als Fortbildungsveranstaltung für aktive Lehrkräfte gedacht. Mittlerweile fanden unsere neuen Lehramtsstudiengänge jedoch so regen Zuspruch, dass mehr und mehr die didaktische Grundausbildung von zukünftigen Lehrerinnen und Lehrern in den Vordergrund rückte. So entsprang aus der steigenden Zahl von Studenten schließlich auch das Bedürfnis nach einem schriftlichen Begleitwerk. Dennoch besteht auch unter den berufstätigen Lehrkräften immer noch ein starkes Interesse an der Didaktik der Informatik, insbesondere vor dem Hintergrund der sehr erfreulichen Entwicklung der bayerischen Schulinformatik in den letzten Jahren. Leider hat dieser Personenkreis selten den zeitlichen Freiraum, um die Vorlesung besuchen zu können, was wiederum ebenfalls zu häufigen Anfragen nach schriftlichen Unterlagen führt.

Unglücklicherweise verfügen die Lehramtsstudenten in der Regel nur über sehr geringe Vorkenntnisse in den Bereichen der Lernpsychologie und der allgemeinen Didaktik, so dass sich in der Vorlesung immer wieder die Notwendigkeit einer kurzen Einführung in diese Grundlagengebiete zeigte. Teil A des Buches ist deshalb diesen *Grundlagen* gewidmet.

Im Laufe der Jahre trug die Vorlesung, nicht zuletzt auf Grund zahlreicher Anregungen durch die Hörer, viel zur Verbesserung unseres didaktischen Ansatzes für den Informatikunterricht an Schulen bei. Aus der Anerkennung, die diese Konzeption inzwischen gefunden hat, entstand auch das Bedürfnis nach einer systematischen Darstellung, was in Teil B (*Konzepte*) des Buches geschehen soll.

Als immer noch aktiver Lehrer ist mir aber auch klar, dass die schönste Theorie ohne erklärende *Beispiele* nichts wert ist. Deshalb enthält Teil C eine ausführliche Sammlung von Unterrichtsbeispielen und -sequenzen für alle möglichen Stufen des Informatikunterrichtes. Meiner einschlägigen Berufserfahrung gemäß sind die Beispiele vor allem auf das Gymnasium ausgerichtet, einer Übertragung auf andere Schularten steht jedoch, entsprechende Anpassungen vorausgesetzt, nichts im Wege.

Mein Dank gilt vor allem Prof. Dr. Manfred Broy, der mich (damals als Dekan) an die Fakultät für Informatik der Technischen Universität München holte, dort mit allem ausstattete, was gut und hilfreich war, mir vertrauensvoll den für meine Arbeit nötigen zeitlichen Freiraum einräumte, mich immer wieder durch Kostproben aus seinem immensen Fachwissen von abwegigen Ideen abbrachte und mir mit seinem unerschütterlichen Optimismus über viele schwierige Phasen hinweghalf. So manche der in diesem Buch vorgestellten Ideen gehen auf seine Anregun-

gen zurück. Weiter geht mein herzlicher Dank an Dr. Peter Müller vom Bayerischen Staatsministerium für Unterricht und Kultus, der die Arbeit an unserer Fakultät gegen so manche Widerstände nach Kräften unterstützte und meine Abordnung an die TU München immer wieder verlängerte. Ohne ihn wäre keiner der jüngsten Fortschritte der bayerischen Schulinformatik zustande gekommen. Ebenfalls herzlich danken möchte ich Prof. Dr. Dr. h.c. Wilfried Brauer, der mich durch seine wertvollen Ratschläge und durch viele Hinweise auf passende Veröffentlichungen oft genau zur rechten Zeit inspiriert und weitergeführt hat. Ihm verdanke ich auch die Herstellung meiner Kontakte zu den einschlägigen Arbeitsgruppen der *International Federation for Information Processing* (IFIP), die mir anlässlich vieler internationaler Konferenzen Gelegenheit zum Austausch mit Fachkollegen aus anderen Ländern eröffneten. Außerdem muss ich mich bei Ulli Freiberger für die vielen Fachgespräche, Anregungen und Ermunterungen bedanken. Seine beeindruckende Unterrichtserfahrung hat mir oft zu mehr Bodenhaftung verholfen. Ich danke auch Prof. Dr. Steffen Friedrich von der TU Dresden für meine freundliche Aufnahme in den Arbeitskreis „Informatik und Schule“ der *Gesellschaft für Informatik* (GI) und sein unermüdliches Engagement für die sehr fruchtbaren *Fachdidaktischen Gespräche* in Königstein sowie Frau Prof. Dr. Sigrid Schubert von der Uni Dortmund für alle Schützenhilfe und Anregung, die sie mir zuteil werden hat lassen. Schliesslich danke ich auch allen Professoren und Mitarbeitern der Fakultät für Informatik der Technischen Universität München, die mich durch Rat und Tat auf meinem Weg zu diesem Buch unterstützt haben, darunter besonders Herrn Prof. Dr. Schlichter, u.a. für sein selbstloses Engagement in der Lehrerausbildung Informatik.

Natürlich bin ich auch dem Springer Verlag, vertreten durch Dr. Hans Wössner, außerordentlich verbunden, weil er mich mit so viel Geduld und Freundlichkeit bis zur Druckreife dieses Buchs begleitet hat.

München, Februar 2000

Peter Hubwieser

## **Vorwort zur zweiten Auflage**

Drei Jahre nach dem Erscheinen der ersten Auflage und zwei Jahre nach einem korrigierten Nachdruck im Jahr 2001 hat die große Nachfrage nach diesem Buch die Lagerbestände soweit geleert, dass eine zweite Auflage notwendig wird. In dieser Zeit hat das hier beschriebene Unterrichtskonzept nicht nur zu einem breit angelegten Schulversuch an Bayerns Gymnasien, sondern sogar zur Einführung eines Pflichtfaches Informatik an allen bayerischen Gymnasien geführt. Ab dem Schuljahr 2004 werden die Schülerinnen und Schüler der 6. Klassen aller bayerischen Gymnasien erstmals einem zweistündigen Pflichtfach Informatik begegnen (mit Ausnahme der musischen Zweige, in denen Informatik erst in der 7. Jahrgangsstufe beginnt). Darüber hinaus wird das Fach am neuen naturwissenschaftlich-technologischen Zweig (dem Nachfolger der mathematisch-naturwissenschaftlichen Ausbildungsrichtung) in den Jahrgangsstufen 9 bis 11 ebenfalls zweistündig fortgesetzt. Somit wurden auch die Voraussetzungen geschaffen, in der Kollegstufe endlich reguläre Grund- und Leistungskurse in Informatik einzurichten.

Im Rahmen der Erprobung der geplanten Unterrichtsinhalte für das neue Pflichtfach ergaben sich Erkenntnisse und Ideen, die eine Überarbeitung einiger Stellen dieses Buches nahe legten. Der bisher intensivste Test betraf den Anfangsunterricht und führte zu zahlreichen Änderungen in Teil C, Kapitel 1.

Abschließend möchte ich dem neuen Fach Informatik in Bayern viel Erfolg und allen Schülerinnen und Schülern viel Spaß dabei wünschen.

Garching bei München, den 30.7.2003

Peter Hubwieser

# **Vorwort zur dritten Auflage**

Nachdem die anhaltende Nachfrage nach diesem Buch auch die zweite Auflage aus dem Jahr 2003 vollständig aus den Lagerbeständen des Verlags entfernt hat, haben sich Autor und Lektorat entschlossen, eine dritte Auflage anzugehen. Mittlerweile hat das junge Pflichtfach Informatik an den bayerischen Gymnasien, dessen Konzeption im wesentlichen in diesem Buch beschrieben wird, seine ersten Jahre in der Praxis erfolgreich überstanden. In dieser Zeit entstanden durch breiten Einsatz in der Praxis viele neue Ideen und Verbesserungen, die in diese Auflage eingebaut wurden, wobei Teil A beinahe unverändert blieb.

Besonders hart fiel der Schulinformatik in Bayern das Überleben der Verkürzung der gymnasialen Ausbildungsdauer auf acht Jahre, was leider nicht ganz ohne Verluste abging. Das ursprünglich selbstständige zweistündige Fach der 6. Jahrgangsstufe wurde zusammen mit Physik und Biologie in den Fächerverbund „Natur und Technik“ eingegliedert und auf zwei einstündige Teile in der 6. und 7. Jahrgangsstufe aufgeteilt. Außerdem wurde Informatik im naturwissenschaftlich-technologischen Zweig aus der Fächertafel der 8. Jahrgangsstufe wieder gestrichen, womit in der Mittelstufe nur noch zwei Jahre verbleiben.

Andererseits hat sich die Informatik in der neuen gymnasialen Oberstufe gut behauptet. Aller Voraussicht nach wird sie als zweites verpflichtendes Fach aus dem naturwissenschaftlich-technischen Feld (zusammen mit Physik, Biologie und Chemie) gewählt werden können und die Möglichkeit einer schriftlichen Abiturprüfung anbieten können. Da auch hier der Lehrplan ziemlich eng unseren Vorschlägen gefolgt ist, wurde dieses Buch um einige Unterrichtsvorschläge für die Oberstufe erweitert, die ursprünglich aus meiner Habilitationsschrift stammen.

Den Schülerinnen und Schülern wünsche ich viel Spaß und Erfolg im neuen Fach Informatik, den Lehrerinnen und Lehrern daneben auch viel Geduld und Hartnäckigkeit bei der Überwindung der durch die neue Schulform verursachten Probleme und Schwierigkeiten.

Garching bei München, den 30.4.2007

Peter Hubwieser

# Inhaltsverzeichnis

## Teil A: Grundlagen

<b>1 Lernpsychologische Fundierung.....</b>	<b>3</b>
1.1 Grundlegende Strömungen .....	3
1.1.1 Behaviourismus .....	3
1.1.2 Kognitivismus.....	5
1.2 Integrative Theorien.....	6
1.2.1 Bandura.....	6
1.2.2 Gagné.....	7
1.3 Entwicklungspsychologie nach Piaget .....	7
1.4 Konstruktivismus .....	10
1.5 Das Gedächtnis .....	11
1.6 Aufmerksamkeit.....	13
1.7 Lernstörungen .....	13
<b>2 Prinzipien didaktischen Handelns .....</b>	<b>15</b>
2.1 Motivierung .....	15
2.1.1 Erzeugung von Motivation .....	16
2.1.2 Verlaufsmotivierung .....	16
2.1.3 Das ARCS Modell .....	17
2.2 Kreativitätsförderung .....	17
2.3 Strukturierung .....	18
2.4 Übung .....	19
2.5 Veranschaulichung.....	20
2.6 Bewertung und Erfolgssicherung.....	21
2.7 Variabilität und Flexibilität.....	22
2.8 Differenzierung .....	22
<b>3 Theoretische Ansätze der allgemeinen Didaktik .....</b>	<b>25</b>
3.1 Bildungstheoretischer Ansatz .....	25
3.2 Lerntheoretischer Ansatz .....	26
3.3 Informationstheoretisch-kybernetischer Ansatz.....	27
3.4 Kommunikative Didaktik.....	28

<b>4 Unterrichtsplanung und -gestaltung .....</b>	<b>29</b>
4.1 Was ist Unterricht? .....	29
4.2 Lerninhalte.....	30
4.2.1 Berliner Didaktik .....	30
4.2.2 Göttinger Schule .....	30
4.2.3 Wagenschein.....	31
4.3 Zeitliche Planung.....	32
4.4 Lernziele .....	33
4.4.1 Lernzieltaxonomien .....	33
4.4.2 Operationalisierung von Lernzielen .....	34
4.5 Lehr- und Lernmethoden .....	35
4.5.1 Artikulation .....	35
4.5.2 Lehrformen .....	36
4.5.3 Sozialformen.....	37
4.5.4 Lehrerverhalten.....	37
4.6 Medien .....	39

## Teil B: Konzepte

<b>1 Informatische Bildung und Informatikunterricht .....</b>	<b>43</b>
1.1 Informatiksysteme und Schulen.....	43
1.1.1 Unterstützung von Lernprozessen .....	44
1.1.2 Bedienerschulung .....	46
1.1.3 Informatikunterricht .....	48
1.1.4 Die Synthese: informatische Bildung .....	48
1.2 Historische Ansätze für den Informatikunterricht .....	50
1.2.1 Die Hardware als Ausgangspunkt .....	50
1.2.2 Der Algorithmus als Maß aller Dinge.....	51
1.2.3 Die vom Algorithmus beherrschte Anwendung .....	51
1.2.4 Der Benutzer im Mittelpunkt.....	52
<b>2 Wozu Informatikunterricht? .....</b>	<b>55</b>
2.1 Wozu überhaupt Unterricht?.....	55
2.1.1 Die gesetzlichen Aufgaben der Schulen .....	55
2.1.2 Allgemeinbildung .....	57
2.2 Bildungsauftrag und Informatikunterricht .....	57
2.2.1 Für welche Welt bilden wir unsere Schüler aus?.....	58
2.2.2 Ist Medienerziehung nicht genug?.....	59
2.2.3 Der allgemein bildende Wert informatischer Bildung .....	62
2.2.4 Informatik zur Berufsvorbereitung .....	64
2.2.5 Allgemeine Studienvorbereitung .....	65
<b>3 Entwurf einer Unterrichtsmethodik .....</b>	<b>67</b>
3.1 Lernpsychologisches Fundament.....	67
3.2 Methodische Prinzipien .....	68
3.2.1 Problemorientierung .....	68
3.2.2 Modellbildung und Simulation .....	69

---

3.3	Organisationsrahmen für den Informatikunterricht.....	70
3.3.1	Verankerung im Pflichtfachbereich .....	70
3.3.2	Zeitliche Grobstruktur.....	70
3.3.3	Feinstruktur der Projekte .....	71
3.4	Bemerkungen zu Unterrichtsmedien.....	72
3.4.1	Bürosoftware.....	73
3.4.2	Hypertextsysteme .....	74
3.4.3	Programmiersprachen .....	74
3.4.4	Programmoberflächen .....	75
3.4.5	Code-Generatoren und Simulatoren.....	75
<b>4</b>	<b>Die Lerninhalte.....</b>	<b>77</b>
4.1	Wozu Lerninhalte systematisieren? .....	77
4.2	Informationszentrierung.....	78
4.2.1	Der Informationsbegriff.....	78
4.2.2	Das Paradigma der Informationsverarbeitung .....	79
4.2.3	Die Grundmenge informatischer Lerninhalte .....	81
4.2.4	Vergleich mit anderen Ansätzen .....	82
4.3	Didaktische Auswahlkriterien für Lerninhalte .....	82
4.3.1	Allgemeine Bedeutung .....	83
4.3.2	Lebensdauer .....	84
4.3.3	Vermittelbarkeit .....	84
4.3.4	Exemplarische Auswahl und Einflechtung .....	84
4.4	Modellierung als inhaltlicher Kern .....	85
4.4.1	Begriffsklärung .....	86
4.4.2	Programmierung und Modellierung.....	87
4.4.3	Unterricht auf der Grundlage von Modellierungstechniken .....	90
<b>5</b>	<b>Ein Gesamtkonzept .....</b>	<b>99</b>
5.1	Die Rahmenbedingungen.....	99
5.2	Die Unterrichtsmodule.....	100
5.2.1	Das Fundamentum .....	100
5.2.2	Die Wahlmodule .....	103
5.2.3	Informatische Allgemeinbildung .....	104
5.2.4	Oberstufe.....	106
5.3	Vorschläge für andere Schularten .....	107
5.3.1	Realschule.....	108
5.3.2	Hauptschule .....	108

## Teil C: Unterrichtsbeispiele

<b>1</b>	<b>Anfangsunterricht in Informatik.....</b>	<b>113</b>
1.1	Datenstrukturen.....	113
1.1.1	Lernziele .....	114
1.1.2	Notation .....	114
1.1.3	Software.....	115
1.1.4	Aufgabenstellung .....	115
1.1.5	Objekte, Klassen und Instanzen.....	116

1.1.6 Attribute und Attributwerte .....	117
1.1.7 Klassen und Attributstrukturen.....	118
1.1.8 Methoden und Botschaften .....	118
1.1.9 Übungsaufgaben und Lernzielkontrollen.....	120
1.1.10 Objektstruktur von Textverarbeitungssystemen .....	120
1.1.11 Beziehungen zwischen Objekten.....	121
1.2 Dateien und Ordner .....	121
1.2.1 Lernziele.....	122
1.2.2 Dateien und Dokumente .....	122
1.2.3 Ordnerstrukturen.....	123
1.2.4 Methoden.....	124
1.3 Versand von Dokumenten .....	124
1.3.1 Lernziele und Zeitrahmen.....	124
1.3.2 Systemanforderungen .....	125
1.3.3 Aufgabenstellung.....	125
1.3.4 Erste Schritte mit dem System.....	125
1.3.5 Der Weg einer elektronischen Nachricht.....	127
1.3.6 Das Format der Adressen .....	128
1.3.7 Anhängen von Anlagen .....	129
1.3.8 Aufgaben .....	130
1.4 Hypertext .....	130
1.4.1 Lernziele.....	130
1.4.2 Die Aufgabenstellung .....	131
1.4.3 Verweise auf andere Dokumente.....	131
1.4.4 Datenwege .....	132
1.4.5 Datenschutzaspekte .....	133
1.5 Verarbeitung von Information .....	134
1.5.1 Software.....	134
1.5.2 Lernziele .....	135
1.5.3 Aufgabenstellung .....	135
1.5.4 Umsetzung .....	135
1.5.5 Aufgaben .....	136
<b>2 Repräsentation von Information .....</b>	<b>137</b>
2.1 Formen der Repräsentation von Information .....	137
2.2 Aufgabenstellung .....	137
2.3 Problemanalyse.....	138
2.3.1 Eine Tabelle als Rastergrafik.....	138
2.3.2 Mathematische Objekte .....	139
2.4 Datenstrukturen .....	141
2.4.1 Das Datenmodell .....	141
2.4.2 Rastergrafik .....	142
2.4.3 Vektorgrafik .....	142
2.5 Verarbeitungsprozesse.....	143
2.5.1 Transformation einer Vektorgrafik in eine Rastergrafik.....	143
2.5.2 Transformation einer Rastergrafik in eine Vektorgrafik.....	143

2.6	Arbeit mit den Modellen .....	143
2.6.1	Transformationszyklus einer Vektorgrafik .....	143
2.6.2	Rastergrafik und Fotoretusche .....	144
2.7	Diskussion und Ausblick .....	145
2.7.1	Graphics Interchange Format (GIF).....	146
2.7.2	Joint Photographic Experts Group (JPG).....	146
<b>3</b>	<b>Funktionale Modellierung, Teil 1 .....</b>	<b>147</b>
3.1	Modellierung mit Hilfe von Funktionen .....	147
3.1.1	Datenflussdiagramme .....	147
3.1.2	Funktionen .....	148
3.1.3	Tabellenkalkulationssysteme .....	149
3.1.4	Spezielle Aspekte von Funktionen.....	150
3.1.5	Die WENN-Funktion .....	151
3.2	Funktionale Modelle von Hardware.....	152
<b>4</b>	<b>Datenmodellierung und Datenbanken .....</b>	<b>155</b>
4.1	Beschreibung der Anforderungen .....	155
4.2	Datenmodellierung.....	156
4.2.1	Das Entity-Relationship Modell .....	156
4.2.2	Relationale Modellierung .....	160
4.2.3	Normalformen des relationalen Modells.....	163
4.2.4	Umsetzung von ER-Modellen in relationale Modelle.....	167
4.3	Abfragen und Berichte .....	168
4.3.1	Funktionsprinzipien .....	168
4.3.2	Relationale Algebra .....	169
4.3.3	Abfragen mit SQL .....	172
4.4	Datenmodellierung eines Fahrplansystems.....	174
4.4.1	Problemstellung .....	174
4.4.2	Informelle Beschreibung.....	174
4.4.3	Datenmodellierung .....	174
4.4.4	Realisierung .....	175
<b>5</b>	<b>Zustandsorientierte Modellierung.....</b>	<b>177</b>
5.1	Programmierung als Dilemma .....	177
5.2	Zustandsmodellierung .....	178
5.2.1	Einführung von Zustands-Übergangsdiagrammen .....	178
5.2.2	Exkurs: Beschreibung abstrakter Maschinen .....	179
5.2.3	Ein Getränkeautomat als Anschauungsobjekt.....	180
5.3	Simulation von Automaten .....	182
5.3.1	Algorithmen und Programme .....	182
5.3.2	Zustände und Variable .....	183
5.3.3	Imperative Programmierung .....	184
5.3.4	Variablen- und Modellzustände .....	186
5.3.5	Automaten mit Ein- und Ausgabe.....	188
5.3.6	Bedingte Übergänge .....	189
5.3.7	Wiederholungen.....	194
5.4	Ausbau und Wertung .....	197

<b>6 Funktionale Modellierung, Teil 2 .....</b>	<b>201</b>
6.1 Problemstellung .....	201
6.2 Problembeschreibung .....	201
6.3 Modellierung .....	202
6.3.1 Datenflüsse und Prozesse .....	203
6.3.2 Der Verschlüsselungsalgorismus .....	203
6.3.3 Die Datenstruktur der Zeichenketten.....	204
6.4 Implementierung.....	206
6.5 Wertung und Ausblick.....	207
<b>7 Objektorientierte Modellierung .....</b>	<b>209</b>
7.1 Problemstellung .....	209
7.2 Modellierung .....	211
7.2.1 Das Objektmodell .....	211
7.2.2 Zeitliche Abläufe .....	213
7.3 Implementierung.....	214
7.4 Wertung .....	217
7.5 Nebenläufigkeit .....	219
7.5.1 Begriffsklärungen .....	220
7.5.2 Implementierung paralleler Prozesse.....	221
7.5.3 Wertung .....	222
<b>8 Rekursive Datenstrukturen .....</b>	<b>225</b>
8.1 Aufgabenstellung und Lernziele .....	225
8.2 Problembeschreibung .....	226
8.3 Formale Beschreibung .....	227
8.4 Implementierung.....	229
8.5 Wertung und Ausblick .....	235
<b>9 Formale Sprachen.....</b>	<b>237</b>
9.1 Aufgabenstellung und Lernziele .....	237
9.2 Beschreibung formaler Sprachen .....	239
9.3 Erkennung formaler Sprachen durch Automaten.....	240
9.4 Implementierung.....	240
<b>10 Rechnerkommunikation.....</b>	<b>245</b>
10.1 Aufgabenstellung .....	245
10.2 Postprotokolle .....	246
10.3 Erzeuger, Verbraucher und Petrinetze .....	248
10.4 Simulation elektronischer Post .....	252
10.4.1 Das Protokoll .....	252
10.4.2 Die Implementierung .....	254
10.5 Wertung und Diskussion.....	259
<b>11 Das Halteproblem .....</b>	<b>261</b>
11.1 Aufgabenstellung .....	261
11.2 Unlösbarkeit des allgemeinen Halteproblems .....	262

<b>12 Das Musterprojekt InfoBank .....</b>	<b>265</b>
12.1 Funktionen, Datenflüsse und Tabellenkalkulation.....	265
12.2 Datenmodelle .....	267
12.3 Zustandsmodelle .....	269
12.4 Objektmodelle.....	270
<b>Literatur .....</b>	<b>275</b>

## **Teil A**

# **Grundlagen**

Im ersten Teil werden die grundlegenden Erkenntnisse der Lernpsychologie und der allgemeinen Didaktik sowie ihre Anwendung zur Planung, Vorbereitung und Durchführung von Unterricht vorgestellt. Diese knappe Erläuterung soll einerseits studentischen Lesern die handwerkliche Mindestausstattung eines Lehrers vor Augen führen, andererseits als Argumentationsgrundlage für unseren Vorschlag zu einer Neukonzeption des Informatikunterrichts (siehe Teil B) dienen.

# 1 Lernpsychologische Fundierung

Die Lernpsychologie untersucht menschliche Lernvorgänge. Sie bildet daher neben den Fachwissenschaften die wichtigste Bezugswissenschaft der Didaktik. An dieser Stelle kann allerdings nur eine kurze, schlaglichtartige Vorstellung ihrer wichtigsten Erkenntnisse erfolgen. Eine ausführliche Darstellung der Lernpsychologie aus unterschiedlichen Blickwinkeln findet man bei Anderson (1989), Lefrancois (1994) und Edelmann (1986).

## 1.1 Grundlegende Strömungen

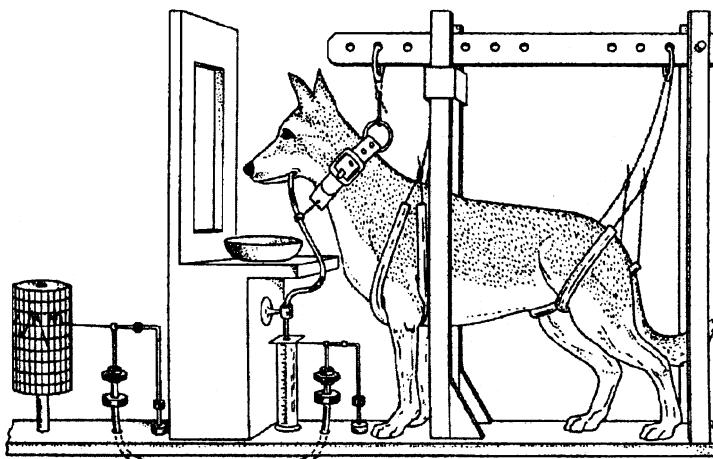
Zur Begründung der später formulierten didaktischen Prinzipien sollen anfangs die Hauptströmungen der Lernpsychologie kurz dargestellt werden. Aus den wichtigsten Erkenntnissen dieser Schulen lässt sich ein zeitgemäßes Modell des Lernvorgangs kombinieren, das als Leitbild für unser Vorgehen im Unterricht geeignet ist.

### 1.1.1 Behaviourismus

Die psychologische Schule des Behaviourismus erhebt den Anspruch, dass alle psychologischen Erkenntnisse in Experimenten verifizierbar sein müssen. Man beschränkt sich daher auf die Erklärung beobachtbarer Phänomene. In Bezug auf die Lernpsychologie bedeutet das eine Konzentration auf die Veränderung von Verhaltensweisen durch Lernprozesse. Das Hauptziel des Behaviourismus liegt in der Bereitstellung von Theorien zur *Vorhersage* bestimmter Reaktionen in einer gegebenen Situation.

Die Grundlagen dafür wurden von den berühmten Experimenten des Russen Pawlow (1849–1936) gelegt. Er erforschte als Erster die Kopplung von neutralen Reizen (*Glockenton*) und *unbedingten Reizen* (Vorlage von Futter), die bei gleichzeitigem Auftreten zur Ausbildung von *bedingten Reaktionen* (Speichelfluss bei Glockenton) führen (siehe Abb.1.1).

Diese ursprünglich für die Erklärung von Tierverhaltensweisen gedachten Ergebnisse wurden dann von Watson (1913) auf die Lernpsychologie übertragen. Sein Verdienst liegt in der Definition der Psychologie in objektiven Begriffen, wobei er forderte, sich auf das Beobachtbare zu beschränken. Er versuchte, alles menschliche Verhalten in pawlowschen Termen zu erklären. Emotionales Verhalten betrachtete er dabei als Subkategorie der klassischen Konditionierung.



**Abb.1.1.** Der berühmte Versuch von Pawlow. Aus Lefrancois (1994)

Durch Thorndike (1913) wurde das Konzept der Verstärkung pavlowscher Koppellungen eingeführt, das schließlich von Skinner (1938) zu seiner Theorie der operanten Konditionierung systematisiert wurde (vgl. Tabelle 1.1). Verstärkung kann demnach in den folgenden Formen auftreten:

**Tabelle 1.1.** Verstärkung nach Thorndike und Skinner

	angenehmer Reiz	unangenehmer Reiz
hinzugefügt	positive Verstärkung	Bestrafung
entfernt	Bestrafung	negative Verstärkung

Wie empirische Versuche zeigten, ist *Bestrafung* weit weniger wirksam als Verstärkung. Erstere führt meist nur zu einer Unterdrückung des Verhaltens in Gegenwart des Bestrafenden.

Die Erkenntnisse der Behaviouristen können uns vor allem zur Erklärung der Auslösemechanismen von Gefühlen und relativ primitiven Verhaltensweisen mit niedrigem Bewusstheitsgrad dienen (Angst, Freude, instinktive Ablehnung).

☞ Praxistip:

- angenehme Lernumgebungen mit entspannter, aufmerksamkeitsfördernder Atmosphäre schaffen,
- kontinuierlich, aber differenziert positiv verstärken (loben),
- Bestrafungen vermeiden,
- Abwehrreaktionen und Angsterzeugung vermeiden.

### 1.1.2 Kognitivismus

Als Gegenströmung zu den behaviouristischen Theorien entstanden parallel in den USA und Europa Lerntheorien, die sich mehr für die lernbedingten (inneren) Änderungen der Strukturen im Gehirn des Lernenden interessierten als für die Beobachtung seiner (äußereren) Verhaltensweisen.

Im Gegensatz zum Behaviourismus beschäftigt sich der Kognitivismus vor allem mit *höheren geistigen Prozessen*. Er gibt die Forderung nach unbedingter Brauchbarkeit für Vorhersagen zugunsten der Erklärbarkeit von Verhaltensweisen durch Modellierung innerer Vorgänge auf.

Wegbereiter dafür waren die Lernmodelle von Hebb (1949), der versuchte, Lernen durch Modellierung elektrochemischer Vorgänge im Gehirn zu erklären. Die wichtigste Rolle in diesen Modellen spielen *Neuronen* (ca. 12,5 Milliarden Nervenzellen in Gehirn und Rückenmark), die *Rezeptoren* (z.B. Sinnesorgane) und *Effektoren* (z.B. Muskelzellen) verbinden. Diese Neuronen übertragen elektrochemische Impulse, wobei sie zwischen zwei Impulsen eine gewisse Erholungspause benötigen. Zur Speicherung von Impulsen sind daher *Kreisläufe* von Impulsen nötig, die als stabile *Erregungskreise* (siehe Abb. 1.2) Ergebnisse von elementaren Lernvorgängen im Gehirn repräsentieren.

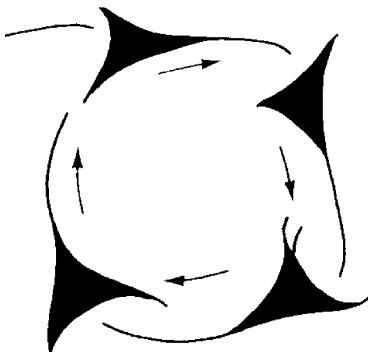


Abb.1.2. Ein Erregungskreis nach Hebb. Aus Lefrancois (1994)

Darauf aufbauend führten Tolman u. Honzik (1930) und die deutschen Gestaltpsychologen Köhler (1921), Koffka (1922) und Wertheimer (1945) kognitive Zwischenprozesse wie Erwartung oder Einsicht ein, um zielgerichtetes Verhalten bei Versuchstieren zu erklären (*kognitive Landkarte*). Der Amerikaner Jerome Bruner (1957) baute diese Erkenntnisse schließlich zu seiner Theorie des Erwerbs von Konzepten aus.

John R. Anderson (1976) und andere schlugen dann Mitte der 70er Jahre vor, menschliches Wissen mit Hilfe *propositionaler Netzwerke* (vgl. Abb. 1.3) zu strukturieren.

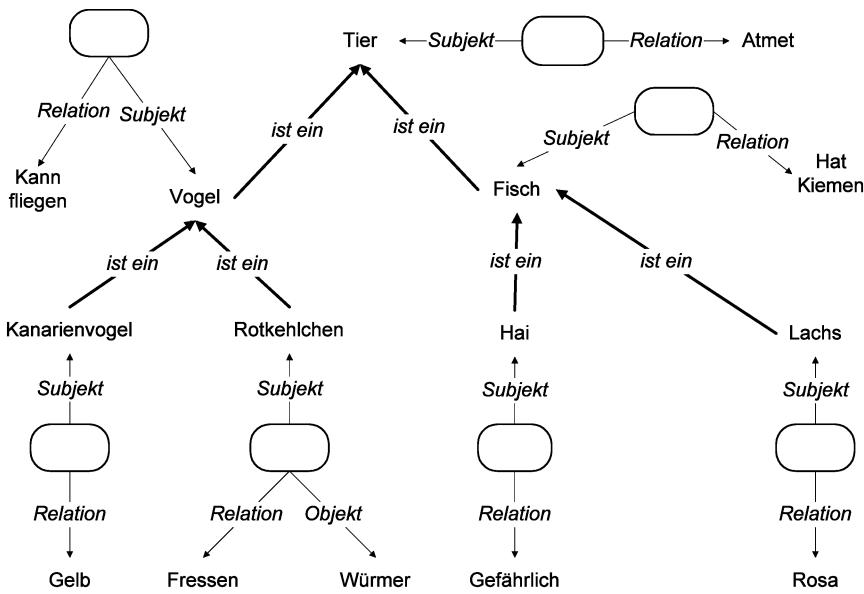


Abb.1.3. Propositionales Netzwerk nach Anderson (1989)

☞ Praxistip:

- Den Lernenden sollte vor dem Beginn der Unterrichtseinheit das Ziel und der Sinn des Lernvorgangs deutlich gemacht werden (Zielangabe).
- Der Lehrstoff ist in übergeordnete Sinnzusammenhänge einzuordnen.
- Die Lerninhalte müssen so strukturiert dargeboten werden, dass die Bildung bzw. Übernahme von Kategorien erleichtert wird.
- Es sollen möglichst viele Anknüpfungspunkte an bekanntes Wissen angeboten werden.

## 1.2 Integrative Theorien

Während sich die bisher besprochenen lernpsychologischen Ansätze relativ klar voneinander abgrenzen lassen, liegt die Leistung der nun folgenden vor allem in der Integration von Erkenntnissen diverser Schulen in ganzheitliche Theorien.

### 1.2.1 Bandura

A. Bandura (1969) fasste, teils in Zusammenarbeit mit R. Walters (Bandura u. Walters (1963)), einige der bisherigen Standpunkte in einem System zusammen, das für die Kontrolle menschlichen Verhaltens drei Möglichkeiten beschreibt.

**Reizkontrolle.** Ein äußerer Reiz bestimmt das Verhalten, darunter fallen autonome (reflektorische) Handlungen wie Niesen, Zurückzucken usw.

**Operante Kontrolle.** Die Handlungen stehen unter der Kontrolle ihrer Konsequenzen (z.B. Verstärkungen).

**Symbolische Kontrolle.** Dieser Bereich von Aktionen wird durch interne Prozesse („Vermittlung“) beeinflusst. Als Beispiele könnten Selbstinstruktion durch verdeckte Verbalisierung oder das Verhalten unter Vorstellung der Konsequenzen dienen.

Ein weiterer zentraler Punkt der Theorie von Bandura ist das Lernen von komplexen Verhaltensweisen durch *Nachahmung* des Verhaltens von besonders ausgezeichneten *Modellpersonen* wie Eltern (emotional), Lehrer (sozial), Medien (Prestige). Gelernt werden auf diese Weise zum Beispiel Kommunikationsmuster, Bewegungsabläufe in bestimmten Sportarten oder auch *spezielles Lehrverhalten*.

☞ Praxistip:

- Beweistechniken, Lehrverhalten oder Problemlösen können auch über Imitation gelernt werden.
- Der Lehrende nimmt eine Vorbildfunktion ein und muss sein Verhalten danach ausrichten.

## 1.2.2 Gagné

Robert Gagné (1985) entwickelte eine weitere Kategorisierung der bis dahin verfolgten Lerntheorien. Lernvorgänge kann man danach in acht verschiedene Klassen einteilen, die aufeinander aufbauen (siehe Tabelle 1.2).

## 1.3 Entwicklungspsychologie nach Piaget

Die wohl bedeutendste Darstellung der zeitlichen Entwicklung menschlicher Denk- und Lernfähigkeiten stammt von Jean Piaget (1975). Er teilte die geistige Entwicklung junger Menschen im Wesentlichen in fünf Phasen auf:

**Sensumotorisches Stadium.** Kinder im Alter bis zu zwei Jahren leben beinahe ohne Sprache ganz im Hier und Jetzt. Objekte existieren für sie anfangs nur, wenn sie wirklich wahrgenommen werden können. Es gibt zunächst keine Vorstellung von Permanenz und Identität. Das Kind perfektioniert und erweitert das kleine Verhaltensrepertoire, mit dem es geboren wurde. Es hat eine durch und durch egozentrische Einstellung zur Welt und ist vollkommen unfähig, sich die (physischen) Sichtweisen anderer Personen zu Eigen zu machen. Im Laufe dieser Phase erwirbt das Kleinkind die Fähigkeit zu symbolisieren und zu kommunizieren (Sprache) sowie ein einfaches Gegenstandskonzept in der Erkenntnis, dass es auch Objekte außerhalb der direkten Wahrnehmung geben kann.

**Tabelle 1.2.** Gagnés Kategorien für Lernvorgänge nach Lefrancois (1994)

Kategorie	Beschreibung	Beispiel
Signallernen	Einfaches Pawlow'sches Konditionieren: ein Signal (Reiz) löst eine Reaktion (Reflex) aus	Speichelfluss bei Glöckenton
Reiz-Reaktionslernen	Bildung einer einzelnen Verbindung zwischen einem Reiz und einer Reaktion	Schüler schweigen, wenn der Lehrer die Hand hebt, nachdem er früher öfter gemahnt hatte
Kettenbildung: motorische Ketten	Verbindung einer Abfolge motorischer Reiz-Reaktions-Verhaltensweisen	Auf den Befehl „Unterstreiche“ nimmt der Schüler seine Farbstifte aus der Mappe und unterstreicht die soeben geschriebene Zeile
Kettenbildung: sprachliche Assoziation	Verbindung einer Abfolge verbaler Reiz-Reaktions-Verhaltensweisen	Lernen der Bedeutung von Autodidakt über Automobil
Multiple Diskrimination	Unterscheidung hochgradig ähnlicher Reizinputs	Lernen verbaler Ketten in der Muttersprache und in einer anderen Sprache
Begriffslernen	Gegenteil von Diskriminationslernen, Ordnen von Objekten zu Klassen und das Reagieren auf Klassen	Bilden von Klassen aus einer Ansammlung von Objekten im Rahmen der objektorientierten Modellierung
Regellernen	Eine Regel ist eine erschlossene Fähigkeit, auf eine Klasse von Reizsituationen mit einer Klasse von Leistungen zu reagieren	Lernen der Syntaxregeln einer Programmiersprache
Problemlösen	Entwicklung von Lösungsstrategien aus bekannten Regeln, Produktion von Regeln höherer Ordnung	Entwicklung eines Algorithmus zur Lösung eines speziellen Problems

**Präkonzeptuelles Denken.** Zwischen der Vollendung des zweiten und des vierten Lebensjahres kann das Kind Objekte und ihre Zugehörigkeit zu Klassen erkennen. Es ignoriert dagegen oft abweichende Eigenschaften von Mitgliedern einer Klasse.

Alle ähnlichen Objekte werden so behandelt, als ob sie völlig identisch wären. Das Denken ist vorwiegend transduktiv: Schlussfolgerungen werden unreflektiert von einem Spezifikum auf das andere übertragen.

**Intuitives Denken.** Im Vorschulalter (4–7 Jahre) ist das Denken des Kindes vor allem von der Wahrnehmung dominiert. Es lässt sich leicht von irreführenden Wahrnehmungsmerkmalen täuschen. Es kann zwar physikalische Sichtweisen anderer Personen nachvollziehen, jedoch nicht mentale. In diesem Bereich argumentiert es weiter rein egozentrisch. Das Kind kann zwar mit Klassen umgehen, ist jedoch mit der Behandlung von Unterklassen überfordert.

**Konkrete Operationen.** In der Gegend der schulischen Primarstufe (7–11/12 Jahre) erwirbt das Kind neue Fähigkeiten vor allem in drei Bereichen:

- **Klassen:** Es kann Klassen kombinieren und dissoziieren sowie Objekte in Klassenhierarchien einordnen.
- **Serien:** Es kann Reihungen erkennen, aufstellen und vergleichen.
- **Zahlen:** Abgeleitet von Klassen und Reihungen lernt das Kind mit Kardinalität und Ordinalität umzugehen.

Das Denken ist in dieser Stufe allerdings immer noch konkret orientiert, d.h. an wirklichen Objekten verhaftet. Kombinatorische Analysen sind noch nicht möglich.

**Formale Denkoperation.** Etwa in der 6. Jahrgangsstufe zeichnet sich erstmals die Fähigkeit zum propositionalen Denken ab, das nicht mehr auf konkrete Wirklichkeiten beschränkt ist. Das Gebiet des Hypothetischen wird einbezogen. Das Kind kann nun vom Wirklichen zum Möglichen und vom Möglichen zum Tatsächlichen folgern, Transitivitäten erkennen und über zukünftige Entwicklungen der Gesellschaft spekulieren.

☞ Praxistip:

- In der Primarstufe muss Unterricht immer von konkreten Dingen ausgehen. Diese sind im Unterricht soweit möglich real zu präsentieren.
- In der Unterstufe weiterführender Schulen sind abstrakte theoretische Konzepte fehl am Platz.
- Formale Operationen können frühestens in der 7. Jahrgangsstufe erfolgreich vermittelt werden.
- Es ist sehr fragwürdig, Kinder vor dem Erreichen der letzten Stufe nach Schularten zu „sortieren“, da spätere Entwicklungen ausbleiben oder auch umgekehrt verstärkt eintreten können.

## 1.4 Konstruktivismus

Im Gegensatz zum radikalen Konstruktivismus, nach dessen Auffassung sich der Mensch seine Realität aus der Interpretation seiner Wahrnehmungen vollständig selbst konstruiert, gewinnt in letzter Zeit ein gemäßigter Konstruktivismus immer mehr an Einfluss im Bereich der pädagogischen Psychologie. Mit Reinmann-Rothmeier u. Mandl (1996) kann man diesen Standpunkt in der Forderung nach einem Primat der *Konstruktion* anstelle von *Instruktion* wie folgt zusammenfassen:

Prozessmerkmale des Lernens:

- Lernen ist nur über die aktive Beteiligung des Lernenden möglich. Dazu gehört, dass der Lernende zum Lernen motiviert ist und dass er an dem, was er tut und wie er es tut, Interesse hat oder entwickelt.
- Bei jedem Lernen übernimmt der Lernende Steuerungs- und Kontrollprozesse. Wenn auch das Ausmaß eigener Steuerung und Kontrolle je nach Lernsituation variiert, so ist doch kein Lernen ohne jegliche Selbststeuerung denkbar.
- Lernen ist in jedem Fall konstruktiv: Ohne den individuellen Erfahrungs- und Wissenshintergrund und eigene Interpretationen finden im Prinzip keine kognitiven Prozesse statt.
- Lernen erfolgt stets in spezifischen Kontexten, sodass jeder Lernprozess auch als situativ gelten kann.
- Lernen ist schließlich immer auch ein sozialer Prozess: Zum einen sind der Lernende und all seine Aktivitäten stets soziokulturellen Einflüssen ausgesetzt, zum anderen ist jedes Lernen ein interaktives Geschehen.

Nach den grundlegenden historischen Vorschlägen zur Arbeitsschule von Kerschensteiner (1950), zum entdeckenden Lernen von Bruner (1961), zur Projektmethode von Dewey (1964) und zum Epochenunterricht von Wagenschein (1970), die hauptsächlich in reformpädagogischen Ansätzen aufgegriffen wurden, haben sich mittlerweile verschiedene konstruktivistische Strömungen herausgebildet (siehe auch Reinmann-Rothmeier u. Mandl (1996), Gerstenmaier u. Mandl (1995), Dubs (1996)).

**Situierte Erkenntnis** (Situated Cognition). Zentral für alle Schattierungen dieser Richtung ist die Annahme, dass bei der Konstruktion von Wissen stets die soziale Umgebung und der inhaltliche Kontext eine tragende Rolle spielt.

**Narrativer Anker** (Anchored Instruction). Das Lernwissen wird bei diesem Ansatz an einem *narrativen Anker* aufgehängt. Dabei handelt es sich um Geschichten, die in authentische und interessante Problemsituationen eingebettet sind. Unter Verwendung des im folgenden Abschnitt 1.6. verwendeten Gedächtnismodells könnte man sagen, dass hierbei eine Querverbindung zwischen dem episodischen und dem semantischen Gedächtnis hergestellt und damit der Lehrstoff doppelt fixiert wird.

**Kognitive Flexibilität** (Cognitive Flexibility). Insbesondere bei der Behandlung von Lerninhalten zu komplexen, wenig strukturierten Themenbereichen sollen unangemessene Vereinfachungen vermieden werden. Stattdessen kann den Lernenden die tatsächliche Komplexität vor Augen geführt werden, indem man ihnen Zugänge zu verschiedenen Zeiten in verschiedenen Kontexten unter veränderter Zielsetzung und aus verschiedenen Perspektiven anbietet. Damit schafft man gleichzeitig mehrere unabhängige Zugänge zum Gelernten, wodurch Erinnerung und Anwendung stark vereinfacht werden.

**Kognitive Handwerkslehre** (Cognitive Apprenticeship). Nach dem Vorbild der traditionellen Handwerkslehre soll dem Lernenden durch Präsentation der Vorgehens- und Problemlösungsmethoden authentischer Vorbilder die Arbeitsweise von Experten auf dem jeweiligen Gebiet vor Augen geführt werden. Die Umgebung, innerhalb derer der Lernprozess stattfindet, sollte dabei möglichst nahe an der Wirklichkeit liegen.

☞ Praxistip:

- Aktive Auseinandersetzung mit dem Stoff ist soweit möglich Pflicht.
- Die Schüler sollen sich Problemlösemethoden selbst erschließen.
- Der Lehrer fungiert als Berater – statt als Präsentator.
- Während des Unterrichts ist genügend Zeit für die Konstruktionsvorgänge zu lassen.
- Lernumgebungen müssen so nahe wie möglich an der Wirklichkeit liegen.
- Derselbe Stoff ist aus unterschiedlichen Perspektiven zu erschließen.

## 1.5 Das Gedächtnis

Zwei Faktoren beeinflussen den Lernenden wesentlich: sein Gedächtnis und seine Aufmerksamkeit. Diese beiden Einflussbereiche sollen nun näher betrachtet werden. Die Darstellung folgt im Wesentlichen Lefrancois (1994).

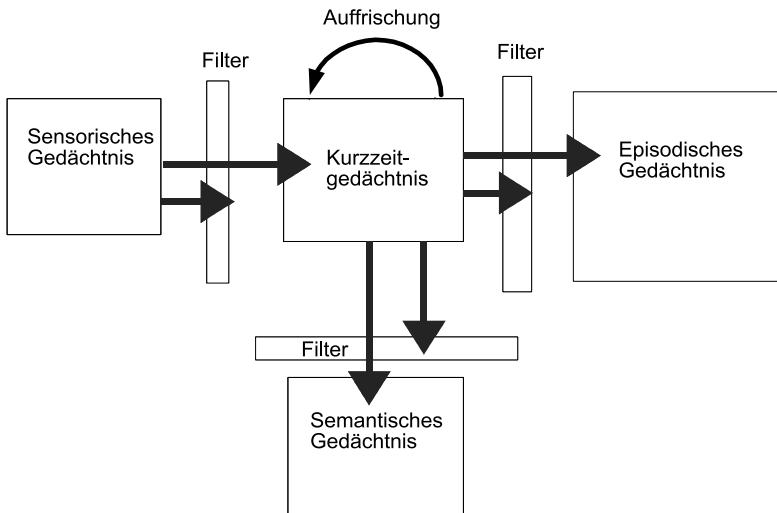
Unter Gedächtnis versteht man die Menge der im Gehirn gespeicherten Informationen, die für die entsprechende Person erreichbar sind. Aufgrund der Untersuchungsergebnisse scheint unser Gedächtnis aus den folgenden drei (logischen) Einheiten zu bestehen:

**Sensorisches Gedächtnis.** Darunter verstehen wir eine ikonische oder *echoische* Form der Speicherung, sie bewirkt eine „fotografische“ Kurzzeitpufferung von aufgenommenen Daten. Eine *große, relativ unstrukturierte Datenmenge* wird für einige Sekundenbruchteile gespeichert.

**Kurzzeitgedächtnis.** Physiologisch gesehen kreisen hier elektrochemische *Informationsströme* in Nervenzellenstrukturen (Hebb: Erregerkreise). Eine *kleine Menge* von Informationen (ca. 7 Einheiten wie Buchstaben oder Ziffern) kann für einige Sekunden oder Minuten behalten werden. Durch *Wiederholen* der Daten kann die Zeitspanne verlängert werden. Weitere Informationen können die gespei-

cherten verdrängen. Durch Bündeln von Informationen (Chunking) kann die Aufnahmekapazität erhöht werden.

**Langzeitgedächtnis.** *Chemische Veränderungen* in den Nervenzellen speichern dauerhaft Informationen. Dies zeigen Messungen einer Erhöhung des Gehirngewichts nach entsprechendem Training sowie einer Erhöhung des Anteils an Ribonukleinsäure nach Lernvorgängen. Es werden *hochstrukturierte* Informationen gespeichert.



**Abb. 1.4.** Ein Modell des Gedächtnisses

Nach Tulving u. Donaldsen (1972) kann man zwischen einem *semantischen Gedächtnis*, in dem das stabile Wissen über die Bedeutung von Worten und Regeln sowie das Verständnis über den Gebrauch von Sprache, Verhaltensstrategien usw. gespeichert ist, und dem *episodischen Gedächtnis*, das spezifische, an Ort und Zeit gebundene persönliche Erinnerungen enthält, unterscheiden.

In jedem Fall muss das Langzeitgedächtnis als ein *Netz* zusammenhängender Informationen gesehen werden. Ein zusammenfassendes Modell für das menschliche Gedächtnis zeigt Abb. 1.4.

 Praxistip:

- Neue Stoffe möglichst bald wiederholen, solange sie noch im Kurzzeitgedächtnis liegen,
  - dargebotene Sachverhalte klar, eventuell mehrfach strukturieren,
  - semantische und episodische Speicherung durch Anknüpfung an persönliche Erlebnisse der Schüler koppeln.

## 1.6 Aufmerksamkeit

Unter Aufmerksamkeit versteht man das Bewusstsein, dass ein Prozess abläuft oder ein Objekt vorhanden ist. Die Aufnahmefähigkeit des Gedächtnisses wird stark vom Grad dieser Aufmerksamkeit beeinflusst.

Abgesehen von elementar wichtigen Daten (z.B. unserem Namen) registrieren wir vor allem solche, auf die sich unsere Aufmerksamkeit richtet (selektive Aufmerksamkeit). Werden zu viele Daten bei zu geringer Aufmerksamkeit dargeboten, reagiert der Organismus in der Regel mit Aggression (Hintergrundlärm, „Lärm macht krank“).

Die *Filtertheorie* von Broadbent (1958) geht von sequentieller Verarbeitung der Input-Daten aus. Die *Filter-Amplituden-Theorie* nach Deutsch u. Deutsch (1963) und Treisman (1964) lässt dagegen auch parallele Verarbeitung zu. Beide gehen davon aus, dass die Unmenge der auf einen Menschen einströmenden Daten vor dem Erreichen des Bewusstseins nach bestimmten Kriterien gefiltert und u.U. untereinander in Beziehung gesetzt wird.

Nach dem *Kapazitätsmodell* von Kahneman (1973) ist die Menge an Informationen, die zu einem Zeitpunkt behandelt werden kann, eine Funktion der Anstrengung. Verschiedene Input-Reize können also leichter gleichzeitig bearbeitet werden als sehr ähnliche.

☞ Praxistip:

Der Lehrende muss das größtmögliche Maß an Aufmerksamkeit aufbauen und aufrecht erhalten, also

- Störungen vermeiden,
- Ermüdung berücksichtigen (Pausen),
- Wichtiges gegenüber Unwichtigerem betonen (Medieneinsatz),
- eine ruhige Atmosphäre schaffen.

## 1.7 Lernstörungen

Bei der Vielzahl von Entscheidungen, die im täglichen Unterrichtsbetrieb getroffen werden müssen, kann vorkommen, dass man aus anderweitigen Gründen (z.B. Strukturierung) ein Vorgehen wählt, das den Schülern letztlich die Aufnahme des neuen Stoffes erschwert. Tabelle 1.3 zeigt einige solche Transfer- oder Speicherungsprobleme, die den Lernvorgang ernsthaft behindern können.

Besonders interessant ist dabei die Ranschburgsche Hemmung, die ein starkes Argument gegen allzu systematisierten Unterricht darstellt: Die aufeinander folgende analoge Darstellung ähnlicher Fälle kann zu deren dauerhafter Verweichlung führen.

**Tabelle 1.3.** Lernstörungen

Name	Beschreibung	Ursache/Zweck	Vermeidung
passives Vergessen	Erregungen oder biochemische Muster verschwinden von selbst	Freigabe veralteter Informationen	Wiederholung, Übung
aktive Hemmungen (retroaktiv / rückwirkend bzw. proaktiv/ vorauswirkend)	unmittelbar aufeinanderfolgende Lernvorgänge können sich gegenseitig stören	Schutz vor der Überlastung des Gehirns	Pausen, Abwechslung, Portionierung
Ranschburgsche Hemmung	das zeitlich benachbarte Lernen ähnlicher Gegenstände kann zu deren dauerhafter Verwechslung führen	Einordnungsprobleme	Abwechslung in den Themen und Methoden
Verzerrung	beim Einordnen in das Langzeitgedächtnis können Umstrukturierungen in Richtung einer Ähnlichkeit mit Bekanntem auftreten	Einordnungs erleichterung	Erleichterung der Einordnung durch Anschau lichkeit, Bezeichnung neuer Sachverhalte
Verdrängung	für das „Seelenheil“ gefährliche Informationen werden versteckt	Schutz vor seelischen Problemen	positive emotionale Belegung der Unterrichtsthemen, Verbalisierung von Tabus und Problemen, keine Strafen

## 2 Prinzipien didaktischen Handelns

Aus den in Kapitel 1 skizzierten grundlegenden Erkenntnissen der Lernpsychologie kann man einige wichtige Prinzipien ableiten, die Lehrkräfte bei Planung und Durchführung von Unterricht beachten sollten. Die Darstellung folgt im Wesentlichen Seibert u. Serve (1992), denen in einer Reihe von Aufsätzen eine bemerkenswert systematische und ausführliche Erklärung der Prinzipien gelang.

### 2.1 Motivierung

Ohne den durch eine angemessene Motivierung erzeugten Lernwillen ist jedes unterrichtliche Bemühen sinnlos. Deshalb ist Motivierung das vordringlichste Ziel didaktischen Handelns.

Zunächst wollen wir einige Begriffe klären. Unter *Motivation* versteht man einen kurz andauernden Zustand des Angetriebenseins, unter *Motivierung* dagegen das aktive Bemühen um die Herstellung eines solchen Motivationszustands. Länger andauernde Initiierungs- und Lenkungsfaktoren wie Funktionslust, Schaffensfreude, Erfolg, Ehrgeiz, usw. bezeichnet man als *Motive* (vgl. Abb. 2.1).

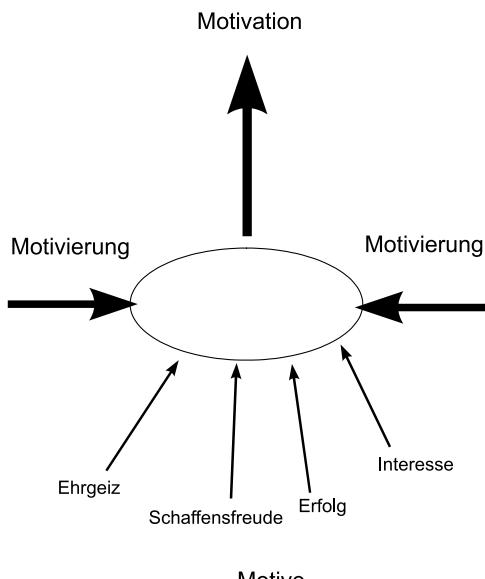


Abb. 2.1. Motivierung, Motive und Motivation

Der zu erreichende Endzustand (*Motivation*) kann nach verschiedenen Aspekten differenziert werden:

- *Lernmotivation* (Auseinandersetzung mit der Welt) vs. *Leistungsmotivation* (Beweis der eigenen Tüchtigkeit),
- *intrinsische* (zielt auf die Beschäftigung mit der Sache selbst) vs. *extrinsische* (zielt auf äußere Begleitumstände wie Lob, gute Noten) Motivation,
- *Eingangs-* (am Anfang der Unterrichtseinheit) vs. *Verlaufsmotivierung* (im weiteren Verlauf der Unterrichtseinheit).

### 2.1.1 Erzeugung von Motivation

Im Gegensatz zu naiven Vermutungen können sich erstaunlicherweise gleichzeitig auftretende intrinsische und extrinsische Motivationselemente durch *Interferenzeffekte* gegenseitig stören, wie Kruglanski et al. (1972) zeigen konnten. Es empfiehlt sich deshalb, in den unumgänglichen Fällen, in denen man extrinsisch motivieren muss, auf eine zusätzlichen intrinsische Motivierung zu verzichten.

Die Beziehung zwischen Motivation und Lernen ist immer *wechselseitig*: Motivation ist Voraussetzung, aber auch Ergebnis von Lernprozessen. Erfolg beim Lernen kann über Motivationsverstärkung nach den Erkenntnissen von Fraser et al. (1987) die Lernleistung mehr als verdoppeln.

☞ Praxistip:

- Erwartungshaltung der Schüler beachten,
- Nähe zum Erfahrungsbereich der Lernenden suchen,
- Überraschungsmomente anbieten,
- keine überhöhten Erwartungen wecken,
- möglichst intrinsisch, nur in unbedingt nötigen Fällen extrinsisch motivieren,
- Freude am Tun vermitteln,
- spontane Interessen berücksichtigen,
- Entscheidungssituationen schaffen,
- Hygienefaktoren berücksichtigen (Luft, Temperatur, Licht, soziales Klima).

### 2.1.2 Verlaufsmotivierung

Es gibt eine Vielzahl von Ursachen, die für ein Nachlassen der Motivation im Verlauf des Unterrichts in Frage kommen.

Das Fragebedürfnis des Lernenden spielt nach Serve (1992) dabei eine wichtige Rolle. Wenn Fragen zum bloßen Nachweismittel von bereits Gelerntem degradiert werden, kann die Motivation stark absinken. Ähnlich verhält es sich mit Antworten auf Lehrerfragen, die sich der Schüler zurechtgelegt hat, ohne sie anbringen zu können.

Einen großen Beitrag zum Motivationsverlust kann der Lehrer selbst leisten, wenn er negative Persönlichkeitsmerkmale (Phlegma) oder eine schlechte Tagesform (Müdigkeit) aufweist, einen stark dominanten oder intoleranten Führungsstil pflegt oder die Schüler andauernd unter- oder überfordert.

Auch die Missachtung anderer, in den folgenden Abschnitten dargestellter Unterrichtsprinzipien wie Differenzierung oder Anschaulichkeit kann sich negativ auswirken, ebenso wie das Ausbleiben von Lernerfolg (siehe oben) oder Anerkennung, methodische Eintönigkeit, fehlender Neugigtsgehalt, unbefriedigte Primärbedürfnisse wie Hunger und Müdigkeit oder störende Umwelteinflüsse wie Lärm oder Hitze.

### 2.1.3 Das ARCS Modell

Von Keller (1987) stammt das sehr anschauliche ARCS-Modell (für Attention – Relevance – Confidence – Satisfaction):

- Attention: Emotionale oder persönliche Information anbieten, Fragen stellen, Mentale Herausforderungen und Interessante Beispiele aufzeigen.
- Relevance: Das Lernziel muss im Kontext der Lebensumstände des Lerners als relevant erkennbar sein
- Confidence: einen passenden Grad von Zuversicht erzeugen; das Ziel erscheint erreichbar, aber dennoch herausfordernd.
- Satisfaction: während der Lernphase muss bereits Lernerfolg sichtbar sein; der Lernprozess muss den Erwartungen entsprechen

☞ Weitere Praxistipps:

- Die Lehrkraft sollte tragende Fragen stellen, besser noch von Schülern stellen lassen,
- auf Folgerichtigkeit des Stundenverlaufs achten,
- eine klare Strukturierung anbieten,
- eine gründliche und vernetzte Bearbeitung des Themas zulassen,
- angemessenes Tempo und Rhythmisierung einhalten,
- Teilerfolge möglichst früh verdeutlichen und Teilergebnisse sichern,
- Erfolgsbilanzen aufstellen.

## 2.2 Kreativitätsförderung

Der Begriff Kreativität wird sehr unterschiedlich gebraucht. Wir verzichten hier auf eine Begriffsklärung, die man sehr ausführlich bei Serve (1992a) nachlesen kann. Aus der Vielzahl der dort gesammelten Ausführungen zur Bedeutung des Begriffs sei hier exemplarisch die von Beer (1970) angeführt:

Kreativität ist der modernere, aber bescheidenere Ausdruck für das, was man früher anspruchsvoll, aber unbestimmt Schöpferkraft nannte: jene geheimnisvolle Fähigkeit, die den Menschen zu originellen, bisher nie da gewesenen Leistungen und Werken befähigt und so die Menschheit bereichert und den Fortschritt der Kultur garantiert. Die Kreativität bezeichnet einen Komplex produktiver Kräfte.

Die Kreativität der Schüler ist jedenfalls Voraussetzung für die Neukonstruktion von Wissen. Deshalb gehört ihre Förderung zu den wichtigsten Zielen des Unter-

richts. Dafür ist die Beachtung der anderen didaktischen Prinzipien Motivierung, Übung und Individualisierung unerlässlich.

Kreativität kann nur in einer freien und offenen Atmosphäre gedeihen, in der Schülerinnen und Schüler als eigenständige Persönlichkeiten akzeptiert und gelegentliche Fehlschläge in Kauf genommen werden. Alle Behinderungen einer solchen Atmosphäre stören auch die Kreativität. Dazu gehören Konformitätsdruck, autoritäre Haltungen, spöttische und zynische Bemerkungen, rigidisierende Persönlichkeitseigenschaften (des Lehrers), Überbetonung von Belohnungen, übermäßiges Streben nach Gewissheit oder Genauigkeit, Feindseligkeit gegenüber andersartigen Persönlichkeiten oder Intoleranz gegen unübliche Lösungswege.

 **Praxistip:**

- Förderung von Kreativität heißt vor allem Förderung von Aktivität.
- Kreativität kann in einer fest vorgeplanten Umgebung nicht gedeihen, ein gewisser chaotischer Bereich ist unabdingbar (partielle Ordnungsstrukturen).
- Indirekter Lehrereinfluss fördert die Kreativität.
- Beiträge, Ideen und Kritiken der Schüler müssen immer positiv aufgenommen werden. Ein freies, offenes Klima ist Voraussetzung.
- Stillarbeit, Schülervorträge oder Gruppenarbeit sind angemessene Arbeitsformen.
- Längere Lehrervorträge sind möglichst zu vermeiden.

## 2.3 Strukturierung

Die moderne Lernpsychologie betont mit ihrem Konzept der kognitiven Netze (siehe Anderson 1976) die Bedeutung der Einordnung des Lernstoffes in übergeordnete Zusammenhänge. Durch eine klare, innere Gliederung des Gelernten wird

**Tabelle 2.1.** Mögliche Gliederungskonzepte

Typ	Erklärung	Beispiele
das logische Beziehungsgefüge	kausale Abhängigkeiten, Gedankenketten, Funktionszusammenhänge dienen zur Strukturierung	vom Problem zum Programm, vom Element über die Menge zur Rechenstruktur
die erlebnisgebundene Ganzheit	das Bezugssystem liegt im Konkreten, die Sinne werden beteiligt	der Rechner hier vor mir mit seiner Tastatur und seinem Bildschirm, unsere Reise durch die Alpen im letzten Sommer
das zweckgerichtete Beziehungsgefüge	die Struktur spiegelt eine zielbewusste Ausrichtung auf einen Zweck wider	die Struktur eines Textverarbeitungssystems anhand seiner Anwendung zum Schreiben eines Briefes

dieser Vorgang unterstützt. In der Darbietung des Lehrstoffes sollen also inhaltliche Unterteilungen, Teilschritte, Abhängigkeiten und Abstraktionen ersichtlich werden, die dem Lernenden die Aufnahme erleichtern.

Nach Kopp (1970) gibt es drei mögliche Arten von „innerem Gefüge“ des Lehrstoffs (vgl. Tabelle 2.1).

Die notwendige Strukturierung kann nach Seibert (1992) grundsätzlich auf zwei Arten vorgenommen werden:

- nach *inhaltlichen* Kriterien: Eingrenzung, Konzentration auf das Wesentliche, Zerlegung in Teilschritte, Konzentration durch Abstraktion, Begriffs- und Regelbildung oder
- nach *methodischen* Kriterien: Stoffverteilungsplan, Querverbindungen, Artikulation (Zergliederung), Medien (Tafelbild, Arbeitsblatt, Folie).

## 2.4 Übung

Übung heißt, das Gelernte durch Wiederholung zu festigen. Datenwissen wird dabei vom Kurzzeitgedächtnis in das Langzeitgedächtnis übertragen und dort in bestehende Strukturen eingebettet.

Je nach Organisationsform des Übungsprozesses unterscheiden wir verschiedene Übungsformen, die in Tabelle 2.2 aufgeführt sind.

**Tabelle 2.2.** Übungsformen

Typ	Erklärung	Beispiele
Apponierte Übung	Jedem Lernschritt wird eine Übung beigeordnet.	Übungen zu den Kapiteln eines Lehrbuchs
Direkte Übung	Neugelerntes soll vor dem Vergessen bewahrt werden.	Hausaufgabe zum soeben behandelten Lernstoff
Disponierte Übung	Kurze, planmäßige, gleichmäßig wiederkehrende Einheiten werden eingestreut.	Tutorübungen zu einer Vorlesung
Latente Übung	In anderen Tätigkeiten sind Übungsformen versteckt.	Erstellen eines Vortrags

☞ Praxistip:

- Der Wert des zu Übenden muss einsichtig sein (möglichst intrinsisch motivieren).
- Der weitere Zusammenhang muss als Vernetzungshilfe erkennbar sein (Ausblicke geben).
- Positive Begleitumstände fördern die Einprägung (angenehmes Unterrichtsklima schaffen).

- Assoziationsmöglichkeiten unterstützen die Speicherung (Querverbindungen, Hinweise auf Bekanntes, Ausblicke).
- Es sind möglichst viele Sinne einzuschalten, um die Verankerung zu verbessern (Hören, Sehen, Mitschreiben, Aktivität allgemein).
- Alle Lerntypen (z.B. visuell, auditiv) sollten berücksichtigt werden.

## 2.5 Veranschaulichung

Als reine Datenmenge gesehen kann die Fülle der auf einen Schüler einströmenden Reize nicht verarbeitet werden. Anschaulichkeit soll ihn durch die Erzeugung von Assoziationen bei der Auswahl von Wichtigem und beim Erkennen der immanenten Strukturen unterstützen. Seibert u. Serve (1992) schreiben dazu:

Veranschaulichung ist das Bemühen des Lehrenden, einen Lerninhalt so aufzubereiten, dass bei aller Wahrung der Sachgemäßheit die *Vorstellungsfähigkeit* des Lernenden unterstützt wird, um zur intendierten Begriffsbildung zu gelangen. Die Anschaulichkeit der Unterrichtsgestaltung zielt auf Anschauung. Dies ist ein aktiver Prozess, der nur vom Lernenden vollzogen werden kann.

Ganzheitlichen Lernkonzepten zufolge sollten dabei natürlich möglichst viele verschiedene Sinnesorgane eingesetzt werden. In van Lück (1996) findet sich dazu eine eindrucksvolle Positronenemissionstomografie, die belegt, dass beim Hören, Sprechen und Sehen derselben Wörter ganz unterschiedliche Gehirnregionen aktiviert werden.

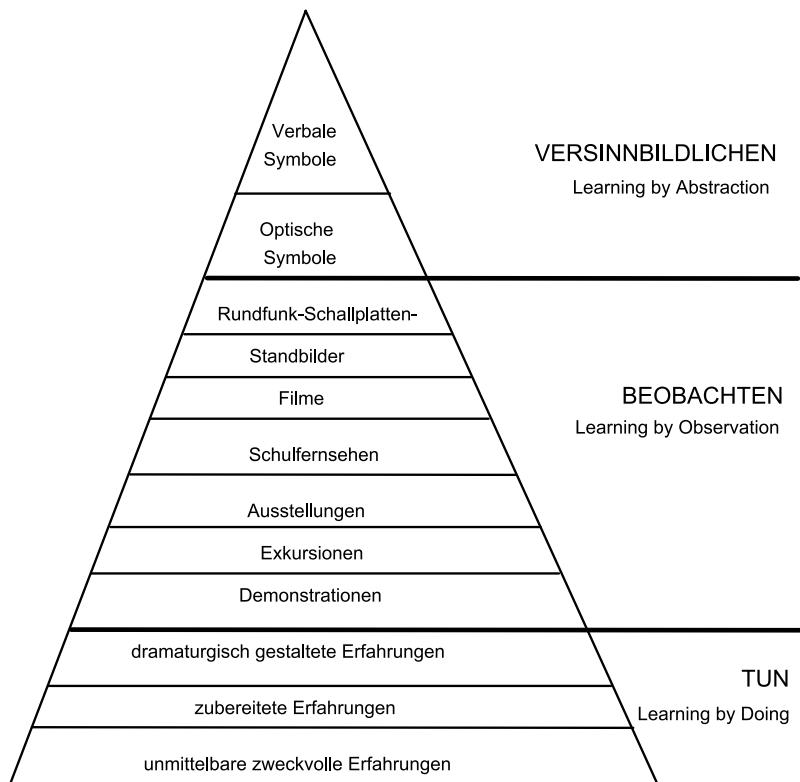
Bei der Darbietung von Unterrichtsgegenständen spielt nach Glöckel (1996) vor allem die Sprache eine wichtige Rolle:

Sprache benennt die Erscheinungen und macht sie so erst zu wahrnehmbaren Sachverhalten.

Bestimmende Faktoren beim Einsatz von Sprache sind nach Seibert (1992a):

- *Verbaler Ausdruck*: Bestimmte Wörter lösen Assoziationsketten, Gedankenketten oder Gefühle aus.
- *Paralinguistische Momente*: Lautstärke, Sprechtempo, Artikulation, Klangfarbe, Intonation, u.a..
- *Körpersprache*: Gestik und Mimik.

Zur Veranschaulichung von Lerninhalten können auch Medien eingesetzt werden. Den höchsten Grad an Anschaulichkeit hat dabei die originale Begegnung. Falls diese nicht eingesetzt werden kann, muss man auf andere Medien bzw. auf den Einsatz von geeigneten Symbolen zurückgreifen. Dale (1954) ordnete zahlreiche Medien in einem Erfahrungskegel nach dem Grad der Anschaulichkeit (von unten nach oben) an (siehe Abb. 2.2). Beim Einsatz von Medien gilt es in jedem Fall, ein Überangebot an Reizen zu vermeiden.



**Abb. 2.2.** Erfahrungskegel von Dale (1954)

## 2.6 Bewertung und Erfolgssicherung

Lernen soll eine Verhaltensänderung bewirken, die für die Abschätzung des Unterrichtserfolgs kontrolliert werden muss. Dies setzt beobachtbares Verhalten voraus. Wenn man Unterricht als interaktiven, geregelten Prozess begreift, so ist dazu Rückkopplung nötig, die vor allem aus solchen Lernzielkontrollen stammt. Dafür kommen in Betracht:

- Fragen zu Aufgaben (auf Antwort bestehen!),
- Beiträge der Schüler,
- Ergebnisse von Stillarbeit, Gruppenarbeit, Schülervorträgen,
- Ergebnisse schriftlicher Prüfungen,
- Ergebnisse von Hausaufgaben (auch von nicht abgegebenen!),
- Unruhe (wegen Über- oder Unterforderung).

Das Größte praktische Problem liegt in der Einschätzung des Schwierigkeitsgrades der anlässlich von Lernzielkontrollen gestellten Aufgaben. Für diese Einschätzung benötigt man eine Skala der Schwierigkeitsgrade (*Lernzieltaxonomie*). Eine ausführliche Darstellung der wichtigsten Taxonomien folgt in Kapitel 4.

☞ Praxistip:

- Formulieren Sie für jede Einheit die wichtigsten Lernziele: Was sollen meine Schüler lernen?
- Beantworten Sie nach jeder Einheit die Fragen:
  - 1) Was haben meine Schüler gelernt?
  - 2) Woher beziehe ich mein Wissen darüber?
- Das Ergebnis einer Bewertung muss auf jeden Fall wieder in den Unterricht einfließen.

## 2.7 Variabilität und Flexibilität

Um Langeweile zu vermeiden, die Strukturierung zu unterstützen, Belastung unterschiedlicher Leistungsbereiche zu ermöglichen, auf unerwartete Situationen eingehen zu können, alle von der Lernpsychologie aufgezeichneten Lernmöglichkeiten und -hilfen zu fördern, und schließlich die Bevorzugung eines bestimmten Lerntyps zu vermeiden, ist Unterricht grundsätzlich so variabel und flexibel wie möglich zu gestalten. Im Hinblick darauf ist vor allem die Präsentation ähnlich strukturierter Lerninhalte durch ähnliche Methoden eine verführerische Falle, in die man allzu leicht geraten kann.

☞ Praxistip

- Setzen Sie viele unterschiedliche Lehrmethoden ein,
- wechseln Sie beim Medieneinsatz ab,
- nutzen Sie variable, dynamische Sprache und Gestik,
- variieren Sie Ihre Sozial- und Interaktionsformen (Lehrervortrag, Schülervortrag, Stillarbeit, Gruppenarbeit, Unterrichtsgepräch).

## 2.8 Differenzierung

Zur Begriffsklärung finden wir bei Schröder (1990):

Differenzierung ist die Auflösung des heterogenen Klassenverbands zugunsten homogener Gruppen in Bezug auf die Leistungsfähigkeit oder die Interessenrichtung der Schüler.

Man unterscheidet dabei *äußere Differenzierung* als formale Auflösung des Klassenverbandes in getrennt unterrichtete Gruppen (Kurse, Lehrgänge) und *innere*

*Differenzierung* als die gruppeninterne Differenzierung unter einer gemeinsamen Lehrkraft.

Differenzierung soll insbesondere die unterschiedlichen Lernvoraussetzungen der Lernenden berücksichtigen, je nach Lerntyp, kognitiven Lernvoraussetzungen, Denkvermögen, logischen Fähigkeiten, Kreativität, Lernvoraussetzungen, individuellen Misserfolgen.

Organisatorische Schwierigkeiten bereitet dabei vor allem die innere Differenzierung. Möglichkeiten dazu sind u.a.

- die Verfolgung unterschiedlicher Lernziele (z.B. Trennung in Fundamentum und Additum),
- die Nutzung spezieller Organisationsformen (Gruppenarbeit, Partnerarbeit, Stillarbeit),
- Wiederholungs- und Besinnungsphasen,
- Auswahl eines geeigneten Medienangebots,
- ein angepasstes Angebot an Lernzeit.

### 3 Theoretische Ansätze der allgemeinen Didaktik

Aus der Sicht eines aktiven Lehrers, der vom Schulalltag voll in Anspruch genommen wird, ist der Nutzen vieler Überlegungen aus dem Bereich der allgemeinen Didaktik oft nicht auf Anhieb erkennbar. Erst auf den zweiten Blick wird an vielen Stellen deutlich, wie sehr prominente Vertreter dieser Richtung wie Klafki oder Heimann unsere Auffassung von Sinn und Zweck der Unterrichtsplanung und -gestaltung geprägt haben. Deshalb sollen nun, bevor wir auf Aspekte der konkreten Unterrichtsplanung eingehen, einige der wichtigsten theoretischen Ansätze der allgemeinen Didaktik kurz vorgestellt werden. Eine ausführliche, kritische Darstellung der Ansätze findet sich z.B. in Peterßen (1982).

#### 3.1 Bildungstheoretischer Ansatz

Das zentrale Leitbild dieses auch *Göttinger Schule* genannten Ansatzes (siehe Klafki (1958) und (1963), Derbolav (1960), und Weniger (1956)) ist die *bildende Begegnung* des Menschen mit der kulturellen Wirklichkeit. Daraus resultiert eine *doppelseitige Erschließung*: Die Wirklichkeit erschließt sich dem Menschen und umgekehrt. Diese Erschließung führt zum Prozess *kategorialer Bildung*. Nach Klafki (1963):

Die Aufnahme und Aneignung von Inhalten ist stets verbunden mit der Formung, Entwicklung und Reifung von körperlichen, seelischen und geistigen Kräften.

Diese Auffassung steht im krassen Gegensatz zur historischen Trennung von *materialer Bildung* auf der Objektseite (Erwerb von Wissen, Erschließung der Wirklichkeit) und *formaler Bildung* auf der Subjektseite (Formung, Reifung von Kräften, Aneignung von Methoden usw.).

Das vorrangige Ziel des Unterrichts ist nach der Göttinger Schule die Allgemeinbildung. Die Auswahl der Inhalte kann dabei allerdings nicht (wie zuvor) den Fachwissenschaften überlassen werden. Sie sollte nur solche Themen erfassen, die nach Klafki (1958) als *Besonderes das Allgemeine* enthalten.

Die Methodik tritt zugunsten der Lehrinhalte in den Hintergrund, die Unterrichtsplanung beschränkt sich im wesentlichen auf das Modell der *Didaktischen Analyse* (vgl. Klafki (1958)). Dabei bestimmen 5 Fragen die Auswahl der Unterrichtsgegenstände:

1. Welche exemplarische Bedeutung hat der Unterrichtsgegenstand?
2. Wie bedeutend ist er für die Gegenwart?
3. Welche Bedeutung für die Zukunft lässt sich vermuten?

4. Wie ist die Struktur des Inhalts?
5. Wie steht es mit der unterrichtlichen Zugänglichkeit?

Für die Belange unserer Informatikdidaktik, die ich in Teil B dieses Buches entwerfen werde, leistet dieser Ansatz vor allem eine solide Fundierung des Begriffes *Allgemeinbildung*, der nach wie vor das zentrale Ausbildungsziel unserer weiterführenden Schulen ist. Auch beim Entwurf von Lehrplänen sind Klafkis Fragen immer noch von großer Bedeutung.

## 3.2 Lerntheoretischer Ansatz

In dieser Schule, oft auch mit *Berliner Didaktik* (siehe Heimann (1962), Heimann, Otto, Schulz (1965)) bezeichnet, wird Didaktik als *Theorie des Lehrens und Lernens* verstanden. Der Lehrende hat unter Berücksichtigung der Elemente verschiedener *Bedingungsfelder* seine *Entscheidungen* in den jeweiligen *Entscheidungsfeldern* so zu treffen, dass die gewünschten Folgen erreicht werden können. Im Gegensatz zur Göttinger Schule, die sich ausschließlich mit der Frage „Was ist zu lehren?“ beschäftigte, geht es hier also auch um die Frage „Wie soll man den Stoff vermitteln?“.

Die Bedingungs- und Entscheidungsfelder (vgl. Tabellen 3.1 und 3.2) der Berliner Didaktik liefern uns eine sehr ansprechende Struktur für unsere Überlegungen zur Unterrichtsplanung in Kapitel 4. Dort findet sich eine systematische Befragung der wichtigsten Aspekte zu diesen Feldern.

**Tabelle 3.1.** Bedingungsfelder der Berliner Didaktik

---

### 1. soziokulturelle Voraussetzungen

sozio-ökonomische	finanzielle und wirtschaftliche Rahmenbedingungen	Klassenstärken, Ausstattung mit Lernmitteln, etc.
sozio-ökologische	Einlagerung des Unterrichts in ein räumliches Umgebungsgeflecht	Stadt- oder Landschule, Verkehrsverbindungen, Lärmbelästigung etc.
sozi-kulturelle	aus der geschichtlich-geistigen Situation erwachsende Strömungen, Einstellungen etc.	Tabus, Kommunikationsweisen, Sprachformen, Symbole etc.
ideologisch-normbildende	aus Interessenlagen einzelner gesellschaftlicher Gruppen und Mächte stammende Einflüsse	politische Richtziele im Wandel der Parteienlandschaft, Einfluss der Umweltschutzbewegung etc.

## 2. anthropologisch-psychologische Voraussetzungen

Schülerseite:	Lernfähigkeit Lernbereitschaft	Lernstand: Wissen, Können, Haltung, Lernstil, Lerntempo
Lehrerseite:	Lehrfähigkeit Lehrbereitschaft	Lehrstand: Wissen, Könen, Haltung, Lehrstil

**Tabelle 3.2.** Entscheidungsfelder

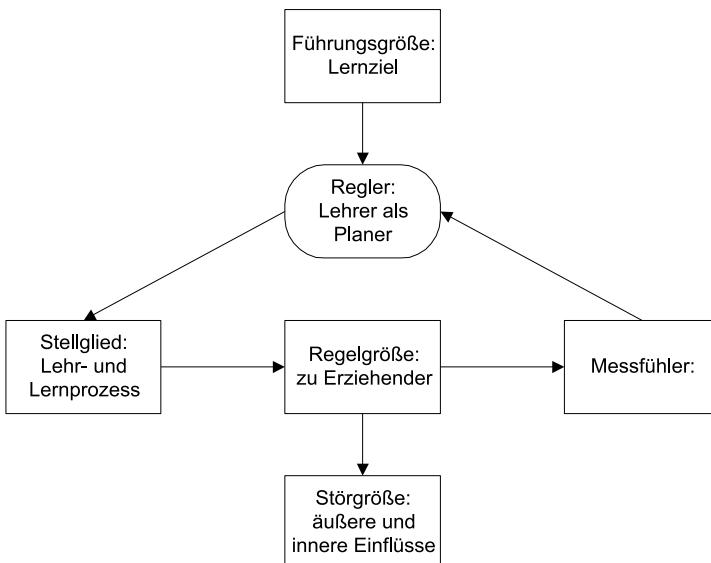
Intentionen	Welche Zielsetzung hat der Unterricht?
Lerninhalte	Was wird gelehrt ?
Methoden	Wie wird der Stoff vermittelt ?
Medien	Womit wird der Lernstoff transportiert?
Folgen des Unterrichts	Welche soziokulturellen und anthropologisch-psychologischen Auswirkungen wird der Lernvorgang haben?

## 3.3 Informationstheoretisch-kybernetischer Ansatz

Unter Anwendung von damals sehr aktuellen Methoden der Informationstheorie und der Kybernetik auf den Prozess des Lernens reduzierten Cube (1968) und Frank (1974) die Didaktik auf *reine Methodik*. Von der Auswahl des Lehrstoffes ist hier nicht mehr die Rede.

Entscheidende Faktoren sind dabei einerseits der *pädagogische Raum*, der sich in die Dimensionen Lehrziel, Lehrstoff, Medium, Soziostuktur und Psychostruktur gliedert, und andererseits die *Struktur des Lernprozesses*, der als *Regelkreis* modelliert wird (siehe Abb.3.1).

Aus heutiger Sicht erscheint es geradezu naiv, eine derart komplexe Domäne wie den Lernprozess mit so einfachen Modellen behandeln zu wollen. Dennoch hatte dieser Ansatz bemerkenswerte Auswirkungen auf Modelle zur Evaluierung von Lehrveranstaltungen oder die Entwicklung von maschinellen Lehr- und Lernsystemen.



**Abb. 3.1.** Regelkreis nach dem Kybernetischen Ansatz

### 3.4 Kommunikative Didaktik

Der letzte hier besprochene Ansatz von Schäfer u. Schaller (1971) behandelt den Unterricht als einen *kommunikativen und edukativen Prozess*, wobei zwei Aspekte des Kommunikationsgeschehens entscheidend sind:

1. die *Inhaltsdimension*, die durch das Curriculum weitgehend festgelegt ist, und
2. die *Beziehungsdimension*, die das Verhältnis zwischen Lehrendem und Lernendem beschreibt.

Besonderer Wert wird auf die fortschreitende *Emanzipation* der Lernenden einerseits gegenüber dem Lehrer, andererseits gegenüber der Umgebung gelegt. Neben der Kommunikation im Unterrichtsprozess wird hier auch die Bedeutung der *Metakommunikation* betont.

Das Verdienst dieses Ansatzes liegt vor allem in der Klarstellung der Bedeutung des sozialen Wechselwirkungsprozesses zwischen Lehrer und Schüler.

## 4 Unterrichtsplanung und -gestaltung

Nach den vorausgegangenen theoretischen Überlegungen ist es nun an der Zeit, sich der Planung und Gestaltung praktischem Unterrichts zuzuwenden. Die Bedeutung soll dabei auf dem Bereich des Pflichtunterrichts an öffentlichen Schulen liegen.

### 4.1 Was ist Unterricht?

Zunächst stellt sich natürlich die Frage nach dem Wesen von *Unterricht*. Der Versuch einer Definition des Begriffes würde in den Entwurf einer allgemeinen Didaktik ausarten. Für meine folgenden Ausführungen wollen wir als Arbeitsgrundlage die folgende Interpretation verwenden:

Unterricht ist ein hochkomplexer *Prozess* mit zahlreichen, auch zyklischen Wechselwirkungen, bei dem Lehrende und Lernende unter gewissen gesellschaftlichen, bürokratischen und materiellen Vorgaben und Rahmenbedingungen im Hinblick auf eine bestimmte Zielsetzung interagieren. Langfristig hat dieser Prozess wiederum Auswirkungen auf die gesamte Gesellschaft und die von ihr formulierten Vorgaben.

Vergleicht man diese Aussage mit dem jeweiligen Unterrichtsmodell der im vorigen Kapitel kurz beschriebenen vier didaktischen Ansätze, so erkennt man, dass jeder dieser Ansätze einen Teil zur Systematisierung dieses Prozesses beiträgt. Die Göttinger Schule beschreibt vor allem Lernziele und mit dem Konzept der *bildenden Begegnung* die Wechselwirkung zwischen den Lerninhalten und den Lernenden, der kybernetische Ansatz dagegen vor allem Regelungs- und Rückkopplungsaspekte. Die kommunikative Didaktik widmet sich vorzugsweise den Interaktionen zwischen Lehrenden und Lernenden. Einer ganzheitlichen Sicht kommt die Berliner Didaktik wohl am nächsten, sie idealisiert jedoch die Entscheidungsfreiheit der Lehrkraft.

Meiner Ansicht nach bringt aufgrund der intensiven Wechselbeziehungen zwischen allen Aspekten des Unterrichts jede Kategorisierung ihre Probleme mit sich. Dennoch scheinen mir *Entscheidungsfelder* von Heimann (1962) als strukturelles Gerüst für unsere Zwecke gut brauchbar, wie sie auch in der ausführlichen Befragung von Peterßen (1982) verwendet wird. Im Gegensatz zu Peterßen, der von *Planungsdimensionen* spricht, betrachte ich die folgenden Abschnitte eher als Sichten aus unterschiedlichen Blickwinkeln auf denselben Bereich.

## 4.2 Lerninhalte

Auch wenn die tatsächlich vermittelten Lerninhalte keineswegs so unabhängig von den Lehrmethoden sind, wie es auf den ersten Blick erscheint, stellt der Unterrichtsstoff doch meist den Ausgangspunkt des Planungsprozesses dar. Dennoch darf die starke Wechselwirkung mit der Methodik und den Lernzielen nicht außer Acht gelassen werden. Meyer (1987) schreibt dazu:

Eine hierarchisierende Überordnung der Inhaltsfrage über die Methodenfrage ist also ebenso verkehrt wie die Umkehrung der Reihenfolge. Es gibt keine lineare Über- und Unterordnung; Inhalte und Methoden des Unterrichts stehen vielmehr in einer komplizierten, von vielen Faktoren abhängigen Wechselwirkung zueinander.

In der Praxis ist die Auswahl der Lerninhalte zumindest im groben meist Sache spezieller Expertenkommissionen. Im Hinblick auf den in Teil B folgenden Entwurf eines Unterrichtskonzeptes für den Informatikunterricht hat dieser Punkt für uns jedoch eine besondere Bedeutung.

### 4.2.1 Berliner Didaktik

Nach Heimann (1962) können Lerninhalte in drei Kategorien eingeteilt werden:

- *Wissenschaften*: aus den (eigentlichen) Wissenschaften resultierende Schulwissenschaften, die deren Erkenntnisse und Verfahren auf die Ebene der Schüler transferieren.
- *Techniken*: alle Fertigkeiten, in denen die Schüler geschult werden.
- *Pragmata*: alle auf ein konkretes Ergebnis ziellenden Handlungsfähigkeiten.

Das Prinzip des *Exemplarischen* hat bei der Auswahl der Inhalte eine tragende Bedeutung, da die Fülle der mit einem Unterrichtsthema verbundenen Informationen nicht in ihrer vollen Breite, sondern nur in der Repräsentation durch Beispiele dargestellt werden kann.

### 4.2.2 Göttinger Schule

Nach dem bildungstheoretischen Ansatz (Klafki (1964)) muss der Lehrer zunächst vier Fragen bezüglich der projektierten Lehrinhalte beantworten:

1. Lässt der Inhalt zu, dass meine Schüler eine allgemeine Kenntnis bzw. Einsicht erwerben können?
2. Ist der Inhalt so strukturiert, dass er neben seiner Besonderheit auch ein über sich hinausweisendes Merkmal aufweist?
3. Lässt sich das Allgemeine an diesem Inhalt auch von meinen Schülern in dieser Lernsituation erfassen?
4. Sollten meine Schüler dieses Allgemeine überhaupt erwerben?

Weiter führt Klafki (1964) 7 Grundformen elementarer Inhalte auf (s. Tabelle 4.1).

**Tabelle 4.1.** Grundformen elementarer Inhalte nach Klafki (1964)

Grundform	Beschreibung	Beispiel
das Fundamentale	nur als Erlebnis erfahrbar	in einer Grenzsituation sich selbst erfahren
das Exemplarische	Allgemeines wird am Besonderen erfahrbar	an einem fallenden Stein das Fallgesetz
das Typische	Allgemeines wird im Besonderen erfahrbar	funktionaler Programmierstil in einem Programm einer speziellen Sprache
das Klassische	Allgemeines wird als Wert erfahren	Klarheit rekursiver Programmierung an Baumoperationen
das Repräsentative	Allgemeines wird als Vergegenwärtigung erfahrbar	an der Stadtmauer wird die Vergangenheit lebendig
die einfache Zweckform	Allgemeines (Form) und Besonderes (Zweck) fallen zusammen	Programmieren mittels einer Programmiersprache durch Programmieren lernen
die einfache ästhetische Form	Allgemeines und Besonderes fallen zusammen	der <i>goldene Schnitt</i> am gleichnamigen Bild

### 4.2.3 Wagenschein

Eine weitere Klassifizierung stammt von Wagenschein (1970), der exemplarische Inhalte in vier Ebenen gliedert:

1. Auf der ersten, untersten Ebene sollen Exempla einen größeren Sach- bzw. Wissensbereich erschließen, z.B. den Aufbau einer imperativen Programmiersprache am Beispiel Pascal.
2. Auf der zweiten Ebene sollen Exempla Einsicht in den Erkenntnisprozess der zugehörigen Wissenschaft vermitteln, z.B. wird anhand eines Problems ein Algorithmus abgeleitet.
3. Auf der dritten Ebene wird die Relativität wissenschaftlicher Erkenntnisbildung einschabbar, z.B. die Erkenntnis, dass eine Modellbildung das Problem in seiner Allgemeinheit einschränken kann.
4. Auf der vierten Ebene schließlich kann die Relativität menschlichen Erkennens überhaupt eingesehen werden, z.B. bei der Frage, wo das Weltall zeitlich oder räumlich endet.

### 4.3 Zeitliche Planung

Die nächsten Überlegungen nach der Auswahl der Lerninhalte betreffen meist den zeitlichen Ablauf. Dazu muss der Lernstoff auf bestimmte Zeiteinheiten verteilt werden. Dies geschieht in der Regel in verschiedenen Planungsstufen mit unterschiedlicher Zuständigkeit (siehe Tabelle 4.2).

**Tabelle 4.2.** Stufenmodell der zeitlichen Unterrichtsplanung

	verantwortlich	zeitlicher Umfang	Inhalt
Lehrplan	Kommissionen	Ausbildungsabschnitt	Lernziele und Inhalte
Jahresplan	Lehrender	Jahr	Lernziele und Inhalte
mittelfristige Planung	Lehrender	Woche	Inhalte und Methoden
Unterrichtsentwurf	Lehrender	Unterrichtseinheit	Inhalte, Methoden, Sozialformen und Medien

**Lehrpläne.** Unter einem Lehrplan verstehen wir eine Vorgabe für den Unterrichtsstoff, der in einem bestimmten *Ausbildungsabschnitt* vom Lehrenden zu behandeln ist. Lehrpläne für öffentliche Schulen werden meist von speziellen Expertenkommissionen erstellt und vom Kultusministerium erlassen. Meist existieren mehrere Ebenen mit unterschiedlicher Zielsetzung. In Bayern fanden sich früher vier Ebenen:

1. allgemeine Vorgaben, Leit- und Erziehungsziele für alle Schulfächer,
2. Charakterisierung des jeweiligen Unterrichtsfachs für die gesamte Schulart,
3. Kurzbeschreibung der Zielsetzungen eines Faches in einer Jahrgangsstufe mit kurzer Auflistung der Lerninhalte,
4. ausführliche Beschreibung der Lerninhalte einer Jahrgangsstufe, oft mit methodischen Hinweisen und Beispielen versehen.

Die vierte Ebene wurde jedoch mit dem Lehrplan für das achtjährige Gymnasium abgeschafft.

Im Allgemeinen können Lehrpläne in Form von *Maximal-*, *Minimal-* oder *Richt-*plänen vorliegen und den Stoff mit *linearer* oder *spiralförmiger* (zyklischer) Themenfolge auflisten. Sie können für den Lehrenden absolut bindend sein oder nur einen Anhaltspunkt für den Stoffumfang darstellen. Die weiteren Planungsstufen übernimmt nun die jeweilige Lehrkraft.

**Jahresplan.** Auf der Basis des vorgegebenen Lehrplans erstellt der Lehrende in der Regel einen Jahresplan, der unter Berücksichtigung der speziellen Gegebenhei-

ten des jeweiligen Schuljahres (Anzahl der Feiertage, Ferienzeiten, Exkursionen, Schulveranstaltungen etc.) und des Ausgangszustandes der Schüler die konkrete zeitliche Verteilung des Stoffes und der Lernziele über *ein Schuljahr* beschreibt. Erfahrungsgemäß sollten dabei reichlich *zeitliche Reserven* für unvorhergesehene Unterrichtsausfälle (z.B. Krankheit der Lehrkraft) eingeplant werden.

**Planung von mittelfristigen Unterrichtseinheiten.** Eine inhaltlich zusammenhängende *Stoffsequenz* (z.B. der Komplex Bruchrechnen in der 6. JGSt.) wird auf einzelne Unterrichtsstunden verteilt. Dabei sollten die resultierenden Stoffportionen als inhaltlich abgeschlossene Einheiten erkennbar bleiben.

**Unterrichtsentwurf.** Schließlich wird das Vorgehen in jeder einzelnen *Unterrichtsstunde* entworfen, wobei Entscheidungen in allen relevanten Bereichen (Lernziele, Methoden, Interaktionen, Medien) zu treffen sind. Alle wichtigen Elemente des Entwurfs sollten schriftlich fixiert werden.

## 4.4 Lernziele

Neben Auswahl und zeitlicher Verteilung des Stoffes ist die Planung der Lernziele der wichtigste Vorgang der Unterrichtsplanung. Die Beantwortung der Frage „Was will ich mit *dieser* Unterrichtseinheit *genau* erreichen?“ regelt in Verbindung mit den Lerninhalten die Planung der restlichen Dimensionen.

### 4.4.1 Lernzieltaxonomien

Mit Hilfe spezieller Taxonomien wird versucht, Lernziele in Ordnungsschemata einzubetten. Eine erste Einteilung lieferte Heimann (1962) in seiner *Berliner Didaktik* (s. oben). Die angestrebten Lernziele können demnach nach Lernbereichen (Klassen) und graduellen Stufen in ein Raster eingeteilt werden (siehe Tabelle 4.3).

**Tabelle 4.3.** Lernzielklassifikation nach Heimann (1962)

Klasse	kognitiv-aktiv	affektiv-pathisch	pragmatisch-dynamisch
Daseins	-erhellung	-erfüllung	-bewältigung
Anbahnung	Kenntnis	Anmutung	Fähigkeit
Entfaltung	Erkenntnis	Erlebnis	Fertigkeit
Gestaltung	Überzeugung	Gesinnung	Gewohnheit

Daran angelehnt teilt man mittlerweile Lernziele im Allgemeinen in drei Bereiche ein, die in Tabelle 4.4 aufgeführt sind.

In Anlehnung an die Stufung der kognitiven Lernziele hat der Deutsche Bildungsrat (1970) eine generelle vierteilige Stufung vorgeschlagen, die sich in vielen ministeriellen Empfehlungen wieder findet:

1. *Problemlösung*: Selbstständiges Kombinieren verschiedener erlernter Kenntnisse und Fähigkeiten zur Lösung noch nicht behandelter Probleme,
2. *Transfer*: Übertragung von Gelerntem auf einen anderen Anwendungskontext,
3. *Reorganisation*: Wiedergabe in gegenüber dem Lernvorgang veränderter Form,
4. *Reproduktion*: Wiedergabe in derselben Form wie beim Lernvorgang.

**Tabelle 4.4.** Lernzielbereiche

Bereich	Stufung nach	Stufen (absteigend)	Quelle
1. <i>kognitiv</i> : Denken, Wissen, Problemlösen, intellektuelle Fähigkeiten	Komplexität	Beurteilung Synthese Analyse Anwendung Verständnis Kenntnis	Bloom (1956)
2. <i>affektiv</i> : Gefühle, Wertungen, Einstellungen und Haltungen	Verinnerlichung	Charakterisieren Organisieren Werten Reagieren Beachten Aufmerksam werden	Krathwohl, Bloom, Masia (1984)
3. <i>psychomotorisch</i> : Bereich von erwerbbaren Fertigkeiten	Koordination	Naturalisierung Handlungsgliederung Präzision Manipulation Imitation	Dave (1968)

#### 4.4.2 Operationalisierung von Lernzielen

Zum Zwecke der Überprüfung müssen die Lernziele als beobachtbares Schülerverhalten formuliert werden, etwa: „Die Schüler sollen eine lineare Gleichung mit zwei Unbekannten lösen können“. Nach Mager (1969) kann man diese Operationalisierung in drei Komponenten realisieren:

1. Beschreibung des erwarteten Endverhaltens. Brauchbare Begriffe dazu sind etwa: „schreiben, identifizieren, unterscheiden“. Nicht brauchbar wären dagegen „wissen, verstehen, vertrauen“.
2. Angabe der Mittel, derer sich die Schüler dabei bedienen dürfen: „Anhand des Graphen einer Funktion“, „mit Hilfe einer Formelsammlung“.
3. Definition des Beurteilungsmaßstabes: „Es genügt, wenn die Schüler zwei einfache Verfahren anwenden können.“

## 4.5 Lehr- und Lernmethoden

Um etwas Ordnung in den vielfältigen Bereich der Methodik zu bringen, möchte ich anfangs zwei Klassifizierungen dafür vorstellen. Nach Heimann (1962) hat der Lehrende bei der Planung des Unterrichts 5 methodische Einzelentscheidungen zu treffen:

1. Artikulation (Phasen, Stufen, Stadien etc.)
2. Gruppen- und Raumorganisation (Art der sozialen Kommunikation)
3. Lehr- und Lernweisen (die einzelnen Aktionen von Lehrenden und Lernenden)
4. methodische Modelle (Gestaltung nach bekannten Modellen)
5. Prinzipien

Meyer (1987) identifiziert dagegen 5 andere Klassen, die er als *methodische Ebenen* bezeichnet (siehe Tabelle 4.5).

**Tabelle 4.5.** Methodische Ebenen nach Meyer (1987)

Ebene	Beispiele
Handlungssituationen	Fragen stellen, Arbeitsaufträge formulieren
Handlungsmuster	Lehrervortrag, Schülerreferat
Unterrichtsschritte	Einstieg, Erarbeitung
Sozialformen	Frontalunterricht, Gruppenunterricht
methodische Großformen	Lehrgang, Projekt

Wir wollen eine Mischform aus den beiden Systematiken wählen.

### 4.5.1 Artikulation

Bei der Artikulation geht es um die Unterteilung einer Unterrichtseinheit in kleinere Abschnitte mit unterschiedlicher Zielsetzung. Herbart (1965) hat dafür bereits Anfang des 19. Jahrhunderts die in Tabelle 4.6 dargestellten Formalstufen postuliert.

**Tabelle 4.6.** Formalstufen nach Herbart

<i>Vertiefung</i>	<i>Klarheit:</i> Der Gegenstand wird vor Augen geführt.
	<i>Assoziation:</i> Alle Erkenntnisse aus der Erinnerung werden in Beziehung zum Gegenstand gesetzt.
<i>Besinnung</i>	<i>System:</i> Die Verbindung einzelner Erkenntnisse wird systematisch aufbereitet, eine Einordnung findet statt.
	<i>Methode:</i> Die Erkenntnis wird angewendet, wobei sie sich verifiziert.

Nach Roth (1963) kann man den Lernvorgang dagegen in 6 Stufen einteilen (siehe Tabelle 4.7).

**Tabelle 4.7.** Lernstufen nach Roth (1963)

Stufe	Inhalt
Motivation	Anstoß des Lernprozesses
Schwierigkeiten	das bisherige Wissen reicht nicht aus
Lösung	der Lösungsweg wird klar
Tun und Ausführen	die neue Strategie wird ausprobiert
Behalten und Einüben	durch Variation der Anwendungen wird die Strategie gefestigt
Bereitstellen, Übertragung, Integration	Übertragung auf andere Situationen, Integration in größere Zusammenhänge

Zudem unterscheidet derselbe Autor noch nach

- *indirektem Lernen*: als Rückwirkung von Handlungen,
- *direktem Lernen*: mit bewusster Lerneinstellung,
- *Lernen mittels Lehrendem*: mit Hilfe eines Lehrers.

Aus heutiger Sicht muss (besonders im Informatikunterricht) diesen klassischen Stufen noch eine Stufe der Unterrichtsevaluierung und -kritik folgen, deren Ergebnisse als Grundlage der folgenden Unterrichtssequenzen herangezogen werden können.

## 4.5.2 Lehrformen

Uhlig (1953/54) ordnete die Vielzahl der möglichen Lehr- und Lernformen in jeweils drei sich entsprechenden Kategorien an (siehe Tabelle 4.8).

**Tabelle 4.8.** Lehr- und Lernmethoden nach Uhlig (1953/54)

Lernmethoden	Lehrmethoden	Beispiele
rezeptiv	darbietend	Vortrag, Anschreiben von Beweisen, Demonstrationsversuch
geleitet-produktiv	anleitend	Gesprächsführung, Begutachtung, Richtigstellung, Beispiele geben
selbstständig-produktiv	anregend	Stellen einer Aufgabe oder eines Problems, Angabe von Informationsquellen oder Arbeitshilfen

### 4.5.3 Sozialformen

Die äußere Form der Interaktion zwischen Lehrer und Schülern wird als Sozialform bezeichnet. In Fortschreibung der Einteilung von Aschersleben (1984) unterscheiden wir zwischen Klassenunterricht, bei dem der gesamte Klassenverband an den gleichen Aktionen teilnimmt, und differenzierterem Unterricht, bei dem der Klassenverband in unterschiedlich handelnde Gruppen aufgelöst wird. Diese Aufteilung kann man dann noch weiter verfeinern (vgl. Tabelle 4.9).

**Tabelle 4.9.** Sozialformen

Unterricht im Klassenverband	lehrerzentriert	Frontalunterricht, Unterrichtsgespräch mit Lehrer
	schülerzentriert	Schülervortrag Unterrichtsgespräch ohne Lehrer, Schüler als Gesprächsleiter
differenzierter Unterricht	Gruppenunterricht	arbeitsgleich, arbeitsteilig
	Einzelunterricht	programm-, lehrer-, schülergesteuert

Da wissenschaftliche und industrielle Tätigkeiten mittlerweile überwiegend im Team durchgeführt werden, sollten alle kommunikationsunterstützenden Sozialformen ausgiebig praktiziert werden, insbesondere Gruppenarbeit und Unterrichtsgespräche.

### 4.5.4 Lehrerverhalten

Der Verhaltenstypus der Lehrkraft kann eine entscheidende Wirkung auf den Unterrichtserfolg haben. Weinert (1971) teilt Muster für Lehrerverhaltensweisen nach ihrer Herkunft ein:

- aus Lehrtradition (vom eigenen Lehrer) übernommen,
- aus sozialer Lernerfahrung hergeleitet („billiger“ Altruismus),
- durch philosophische Traditionen begründet,
- durch Bedürfnisse des Lehrers bewirkt (Selbstbestätigung),
- durch Bedingungen der Lehrinstution erzwungen (Lärm),
- durch Erforschung des Lernens fundiert.

Besonders einflussreich auf Motivation, Kreativität und Aufmerksamkeit der Lernenden ist der Führungsstil der Lehrperson. Tabelle 4.10 zeigt die 3 Kategorien von Lewin (1953).

Der für die spezielle Situation jeweils angemessene Führungsstil lässt sich nicht streng einer der drei Kategorien zuordnen. Er beinhaltet meist Elemente aus allen drei. „Im Durchschnitt“ sollte der Führungsstil in jedem Fall demokratisch sein.

**Tabelle 4.10.** Führungsstile nach Lewin (1953)

Führungsstil	Lehrerverhalten	Auswirkungen
autoritär	Die Lehrkraft <ul style="list-style-type: none"> <li>- legt alle Richtlinien fest,</li> <li>- schreibt Techniken/Tätigkeiten vor,</li> <li>- lobt und tadelt nach persönlichen Gesichtspunkten,</li> <li>- hält sich abseits von der Gruppe.</li> </ul>	größere Leistungsquantität, geringere Qualität geringere Arbeitsmoral mehr Konflikte und Aggressionen kaum Arbeitseinsatz bei Abwesenheit des Lehrers
demokratisch	Die Lehrkraft <ul style="list-style-type: none"> <li>- lässt Richtlinien nach Diskussion entscheiden,</li> <li>- hilft beim Entscheiden,</li> <li>- schlägt evtl. Alternativen vor,</li> <li>- orientiert sich bei der Bewertung an objektiven Kriterien,</li> <li>- versucht, Mitglied der Gruppe zu sein.</li> </ul>	höhere Arbeitsmoral weniger Aggressionen besserer Arbeitseinsatz bei Abwesenheit des Lehrers geringere Produktionsmenge höhere Qualität
laissez-faire	Die Lehrkraft <ul style="list-style-type: none"> <li>- überlässt alle Entscheidungen den Schülern,</li> <li>- beschafft lediglich Material,</li> <li>- gibt Informationen nur auf Befragen,</li> <li>- zeigt keine Teilnahme, keine Beurteilung,</li> <li>- verzichtet auf jede Regelung.</li> </ul>	geringe Arbeits- und Gruppenmoral geringe Produktivität

Neben der Vermittlung von Kenntnissen und Fähigkeiten besteht ein wesentlicher Teil der Aufgaben eines Lehrers aus der Erziehung seiner Schüler. Das allgemeine Erziehungsverhalten von Lehrpersonen liegt nach Tausch u. Tausch (1965) zwischen den zwei Extremen:

- *autokratisch*: Unfreundlichkeit, Unhöflichkeit, Verständnislosigkeit, Pessimismus, Erregung, Verärgerung, Befehlen, Strafen lösen bei den Schülern Ablehnung sowie Gefühle der Unfreiheit aus, die seelische Reifung wird behindert,
- *sozialintegrativ*: Freundlichkeit, Höflichkeit, Verständnis, Ruhe, Optimismus verursachen bei den Schülern annehmende Reaktionen und fördern ihre seelische Reifung.

Der Lehrer versucht auch, die Schüler in das soziale Gefüge der Lerngruppe zu integrieren, wobei er sich nach Gordon u. Wayne (1965) unterschiedlich orientieren kann (siehe Tabelle 4.11).

**Tabelle 4.11.** Integrationsweisen nach Gordon u. Wayne (1965)

Instrumental	Orientierung an den Zielen des Systems Integration beruht auf Autoritätsanwendung	hohe Produktivität schlechte Gruppenmoral
Instrumental-expressiv	Orientierung in Richtung Anpassung der Systemziele an Schüler und Lehrer Kombination von Autorität mit Empfehlung, Überredung, persönlichem Einfluss	hohe Produktivität gute Gruppenmoral
Expressiv	Orientierung an den Zielen und Bedürfnissen von Schülern und Lehrern Integration durch Empfehlung, Überredung und persönlichen Einfluss Bedürfnisse sind das Kriterium des Handelns	niedrige Produktivität gute Gruppenmoral

Insgesamt ergibt sich für den Lehrer ein Dilemma:

- einerseits soll er beim Schüler positive Emotionen auslösen, um ein günstiges Lernklima zu erzeugen,
- andererseits muss er dem Klassenverband gegenüber eine gewisse Autorität darstellen, um den organisatorischen Rahmen aufrechtzuerhalten.

## 4.6 Medien

Medien sollen vor allem der Veranschaulichung von Lerninhalten dienen. An den in Kapitel 2 (Prinzipien didaktischen Handelns) beschriebenen Erfahrungskegel von Dale (1969) sei hier nochmals erinnert.

Die eingesetzten Medien können je nach der mit ihnen gekoppelten Aktivität überwiegend mit bestimmten Lehr- und Lernverfahren kombiniert werden (siehe Tabelle 4.12).

**Tabelle 4.12.** Lehr- und Lernverfahren und geeignete Medien

gekoppelte Aktivität	Lehrverfahren	Lernverfahren
tun	anregend	selbstständig-produktiv
beobachten	anleitend	geleitet-produktiv
versinnbildlichen	darbietend	rezeptiv

Bei der Auswahl der einzusetzenden Medien sollte man sich in jedem Fall einige Fragen stellen:

1. Ist dieses Medium eindeutig genug, um das intendierte Lernziel unmissverständlich und klar erscheinen zu lassen?
2. Repräsentiert dieses Medium den intendierten Inhalt derartig isomorph, dass es eine optimale Erfahrungsquelle für den Lernprozess schafft?
3. Ist dieses Medium attraktiv genug, um Aufmerksamkeit zu erregen?

## **Teil B**

# **Konzepte**

Im zweiten Teil sollen, ohne Anspruch auf Vollständigkeit zu erheben, die wichtigsten didaktischen Konzepte für den Informatikunterricht vorgestellt werden. Nach Klärung der Frage, was wir eigentlich unter Informatikunterricht zu verstehen haben, folgt ein kurzer Abriss seiner geschichtlichen Entwicklung, gefolgt von einer ausführlichen Begründung der Notwendigkeit einer systematischen informatischen Bildung. Anschließend erläutern wir unseren didaktischen Ansatz im Hinblick auf Methodik und Lerninhalte. Schließlich stellen wir exemplarisch ein Gesamtkonzept für den Informatikunterricht an Gymnasien vor, das sich jedoch in abgewandelter Form auch auf andere Schularten übertragen lässt.

# 1 Informatische Bildung und Informatikunterricht

Bevor wir unsere Vorschläge zur Gestaltung des Informatikunterrichts erläutern, müssen wir die zentralen Begriffe „Informatische Bildung“ und „Informatikunterricht“ klären. Dazu gehört auch ein Abriss der kurzen, aber wechselhaften Geschichte des Informatikunterrichts. Hierbei erheben wir allerdings nicht den Anspruch einer vollständigen Darstellung, wie sie von Baumann (1996) versucht wurde. Wir wollen lediglich ein solides begriffliches Fundament für unsere späteren Überlegungen zum Informatikunterricht legen.

## 1.1 Informatiksysteme und Schulen

Durch die offensichtliche Dominanz *moderner Informations- und Kommunikationssysteme* in unserer Gesellschaft werden zunehmend auch unsere Schulen gezwungen, sich mit solchen Systemen auseinander zu setzen und sie soweit möglich auch im Unterricht zu verwenden (siehe z.B. Rüttgers (1997)). Dabei erhebt sich natürlich zuallererst die Frage, von welchen Systemen hier die Rede ist. Schließlich könnte man auch Rauchzeichenketten eines Indianerstammes oder prähistorische Höhlenmalereien als *Informations- und Kommunikationssysteme* bezeichnen. Sind die Briefpost, ein relaisbetriebenes analoges Telefonsystem oder ein TV-Gerät nicht sogar *moderne* Informations- und Kommunikationssysteme? Zur Vermeidung einer allzu weiten Auslegung des Begriffes wollen wir uns in diesem Buch auf Systeme beschränken, die drei Eigenschaften aufweisen (siehe auch Hauf-Tulodziecki (1999)):

- automatische Verarbeitung von Daten,
- Vernetzung,
- Interaktion mit menschlichen Benutzern.

Solche Systeme werden wir im Folgenden als *Informatiksysteme* (siehe Schulz-Zander et al. (1993)) bezeichnen. Natürlich sind einzelne Computer als zentrale Komponenten solcher Systeme ebenso durch diesen Begriff abgedeckt wie Mobiltelefone, elektronische Steuerungsanlagen oder eingebettete Systeme, z.B. in Kraftfahrzeugen.

In der schulischen Auseinandersetzung mit Informatiksystemen finden sich derzeit drei verschiedene Schwerpunktsetzungen. Die Betonung liegt dabei

- auf dem Nutzen solcher Systeme zur Unterstützung von Lernvorgängen aller Art oder
- auf der Schulung von Bedienerfertigkeiten für konkrete Systeme oder

- auf der Vermittlung allgemeinerer und langlebiger Grundlagen der Informatstechnik.

Daneben gibt es ein breites Einsatzspektrum für Informatiksysteme außerhalb des Unterrichtsgeschehens, von der Textverarbeitung im Sekretariat über die häusliche Arbeit der Lehrer und die Personalstatistik bis zur Erstellung der Zeugnisausdrucke, das jedoch wenig mit dem Thema dieses Buches zu tun hat.

### **1.1.1 Unterstützung von Lernprozessen**

Den größten zeitlichen Umfang unter den drei oben genannten unterrichtlichen Schwerpunktsetzungen dürfte wohl mittlerweile der Einsatz von konkreten Informatiksystemen als Lernhilfen ausfüllen. Die rasche Verbreitung des Internet hat dazu sicher einen wichtigen Beitrag geleistet. Dabei werden vorwiegend Lerninhalte aus Bereichen außerhalb der Informatik („across curriculum“) vermittelt. Als Beispiele könnte man nennen:

- die Beschaffung der letzten Rede unseres Bundespräsidenten aus dem Internet für den Sozialkundeunterricht,
- den Einsatz eines Übungsprogrammes zur Primzahlzerlegung im Fach Mathematik,
- die Simulation der Funktionsweise eines Kernkraftwerkes mit Hilfe geeigneter Unterrichtssoftware für die Physik.

In diesen Bereich fällt auch die Nutzung spezieller Lernsoftware aus dem Bereich „computerbasiertes Training“ (CBT), deren Behandlung den Rahmen dieses Buches sprengen würde. Dennoch soll dem Leser die interessante Kategorisierung von Schulmeister (1994) nicht vorenthalten werden, der Lehr- und Lernsysteme nach der jeweils dominierenden psychologischen Sicht des Lernvorgangs klassifiziert (siehe Tabelle 1.1).

**Tabelle 1.1.** Kategorisierung von Lehr- und Lernsystemen nach Schulmeister (1994)

Lernpsychologische Sicht	Lernprogrammtypus
behaviouristisch	Drill & Practice
kognitivistisch	Tutorielle Systeme
konstruktivistisch	Offene Lernumgebungen

Der Einsatz von Informatiksystemen zur Veranschaulichung abstrakter Vorgänge durch Simulation sowie der unterrichtliche Einsatz globaler Netze könnte dabei im weitesten Sinne der dritten Kategorie zugerechnet werden.

Für diesen Anwendungsbereich von Informatiksystemen wird in letzter Zeit häufig der Begriff „Medieneinsatz“ verwendet. Im Zusammenhang damit wird vor allem von Politikern aller Schattierungen meist eine intensive „Medienerziehung“ mit dem Ziel der Vermittlung von „Medienkompetenz“ gefordert. Die Bund-Länderkommission für Bildungsplanung und Forschungsförderung (1995) schreibt dazu:

Die medienerzieherische Arbeit in der Schule umfasst vor allem drei Aufgabenbereiche:

- a) Nutzung von Medien und nichtmedialen Möglichkeiten für unterschiedliche Aufgaben. Ziel ist die Kenntnis unterschiedlicher Medienangebote und nicht-medialer Möglichkeiten sowie die Fähigkeit zu einer bewussten Auswahl und Auswertung in den Bereichen
  - Unterhaltung und Vergnügen ... ,
  - Information ... ,
  - Kommunikation ... ,
  - Problemlösung und Bildung
- b) Einblick in Wirkungsweise und Produktionsbedingungen von Medien. Ziel ist es, eine Haltung kritischer Aufmerksamkeit gegenüber der Beeinflussung von Wahrnehmen, Denken und Handeln zu entwickeln. Dazu gehören:
  - das Aufarbeiten von Medienerlebnissen über spielerisch-kreative Formen der Auseinandersetzung ... ,
  - das Aufarbeiten von Medienerlebnissen im Gespräch ... ,
  - das Verstehen und Unterscheiden von Medienangeboten ... ,
  - die Analyse und Bewertung von Medien aufgrund von Kenntnissen und Einsichten in institutionelle Bedingungen von Medienproduktion und -distribution.
- c) Praktisch-gestalterische Medienarbeit. Ziel ist es, die persönlichen Ausdrucks- und Gestaltungsmöglichkeiten zu erweitern, die Fähigkeit zu genauer Wahrnehmung und zu sozial verantwortlichem Medienverhalten auszubilden.

Bei genauerer Betrachtung entpuppen sich jedoch beinahe alle Unterrichtshilfsmittel von der Tafel bis zur menschlichen Sprache als potentielles Medium. Deshalb ist es aufgrund der besonderen Fähigkeiten von Informatiksystemen angebracht, sehr sorgfältig zwischen herkömmlichen und *computerbasierten* Medien zu unterscheiden. Die Gesellschaft für Informatik schreibt dazu in ihren jüngsten „Empfehlungen zur informatischen Bildung und Medienerziehung“ (siehe Annemarie Hauf-Tulodziecki (1999)):

Wie traditionelle technische Medien, so ist auch das Medium Computer ein Artefakt, das die Wahrnehmung, Speicherung, Darstellung und Übertragung von Informationen unterstützen kann. Für multimediale Anwendungen des Computers werden ganz unterschiedliche Daten, z.B. Texte, Bilder, Grafiken, Audio- oder Videosequenzen digitalisiert und dadurch auf dem selben Medium speicherbar. Gleichzeitig besitzt das Medium Computer als Informatiksystem jedoch Besonderheiten und neue Funktionalitäten:

- (1) Durch Computerprogramme können Daten automatisch verknüpft, neu zusammengesetzt, verarbeitet werden. In diesem Sinn kann der Computer auch als instrumentales Medium bezeichnet werden.
- (2) Computer werden als interaktive Medien bezeichnet. Die so genannte Interaktion zwischen Mensch und Computer beruht darauf, dass durch die Reaktionen des Systems auf Eingaben ein Eindruck von Kommunikation erzeugt werden kann.
- (3) Computer können Informationen durch lokale und globale Netze an beliebige Orte übertragen bzw. von beliebigen Orten empfangen. Sie können für die Individualkommunikation ebenso gut genutzt werden, wie als Massenmedium.

...

Diese Merkmale des Mediums Computer bewirken, dass Computer noch in ganz anderer Weise an der Herstellung und Rezeption von Inhalten beteiligt sind, als dies bei traditionellen Medien der Fall ist: Was digitalisiert gespeichert wird, kann über Software auch verarbeitet und mit dem selben Medium übertragen und dargestellt werden. Sogar die Inhalte selbst können von einem Programm erzeugt worden sein.

...

Für eine reflektierte Nutzung computerbasierter Medien ist es notwendig, sich bewusst zu machen, welchen spezifischen Beitrag Informatiksysteme – gegenüber anderen Medien sowie im Vergleich mit nichtmedialen Möglichkeiten – in dem jeweiligen Zusammenhang leisten können. Insbesondere aber ist auch die Frage zu stellen, wo und für welche Aufgaben automatische Prozesse genutzt werden sollen und wofür ausdrücklich nicht.

Obgleich die Untersuchung der Anwendungsmöglichkeiten informatischer Systeme als Lernhilfe oder Unterrichtsmedium nicht in der Absicht dieses Buches liegt, kann der Bereich der *informatischen Bildung*, um den es uns hier geht, nicht völlig unabhängig losgelöst vom Bereich des *Medieneinsatzes von Informatiksystemen* betrachtet werden. In Abschnitt 2.2.2 wird deshalb die Beziehung zwischen diesen beiden Bereichen nochmals näher beleuchtet werden.

### **1.1.2 Bedienerschulung**

Eine zweite mögliche Schwerpunktsetzung bei der Behandlung von Informatiksystemen im Unterricht ist die Schulung von Bedienerfertigkeiten an konkreten Informatiksystemen. Als Beispiele könnte man eine Einführung in die Handhabung eines speziellen Textverarbeitungssystems oder in die Bedienung eines bestimmten E-Mail-Programms anführen. Oft sind derartige Kurse sogar nach den verwendeten Produkten benannt: „Einführung in Excel 7®“. Bei solchen Bedienerschulungen werden in der Regel sehr spezielle Kenntnisse und Fertigkeiten über das jeweilige System gelehrt, häufig in Form von „Kochrezepten“, die den Weg durch die Menüstruktur des jeweiligen Produktes weisen. Leider sind solche Kenntnisse selten übertragbar, wodurch bei jedem Softwarewechsel der Besuch eines „Auffrischungskurses“ notwendig wird. Andererseits kann im Sinne exemplarischen Lernens implizit auch eine begrenzte Vermittlung von durchaus übertragbarem Wissen stattfinden, wenn nicht die Struktur der *Software* in Vordergrund steht, sondern die der *Aufgabenstellung*. Dann wird das erzeugte Wissensnetz (siehe Teil A, Abschnitt 1.1.2) mehr von der Problemdomäne geprägt anstatt vom zufälligen Aufbau eines konkreten Softwareprodukts. Ein Wechsel der verwendeten Software fällt dann wesentlich leichter, da man in den Begriffen des Problemraums statt in denen der Softwareoberfläche denkt. Beispielsweise ist es bei einer Schulung in Textverarbeitung anzuraten, von der allgemeinen Datenstruktur von Textdokumenten und typischen Operationen darauf auszugehen, als die Menüstruktur eines bestimmten Softwareproduktes durchzusprechen.

Mit ihren Richtlinien zur Konstruktion guter Software („a direction for design“) haben Winograd u. Flores (1986) in beispiellos tiefgründiger Weise zugleich implizit auch Zielsetzungen für wirksame Bedienerschulung an realer (das heißt in ihrem Sinn nicht optimaler) Software vorgegeben. Für die Bedienerfreundlichkeit

von Software setzen sie drei (ursprünglich auf Konzepten von Martin Heidegger (1977) basierende) Kriterien an:

**Zuhanden sein** („readiness-to-hand“). Gute Software muss in dem Sinn zuhanden sein, wie ein Handwerker während selbstvergessener Arbeit an einem Werkstück einen Hammer nutzt: Ohne sich der Existenz des Hammers als eines eigenständigen Objektes bewusst zu sein, hat er nur sein entstehendes Werk im Sinn. Von der Software wird analog dazu gefordert, dass der Benutzer gedanklich in der Begriffswelt seiner Aufgabenstellung verbleiben kann, sich also nicht um Konzepte kümmern muss, die nur für die Struktur der Software von Belang sind. Da derzeit reale Programme weit von diesem Ziel entfernt sind, muss in Schulungen deshalb eine möglichst enge Beziehung zwischen den Begriffswelten von Aufgabenbereich und Softwareoberfläche hergestellt werden. Die Schulung sollte also anhand derselben Aufgabenstellungen stattfinden, wie sie später beim tatsächlichen Einsatz der Software auftreten werden. Dies führt erneut zur Forderung einer problemnahen Ausbildung.

**Vorwegnehmen von Pannen** („anticipation of breakdown“). Ebenso wie Winograd u. Flores für die Konstruktion von Software fordern, dabei alle vorhersehbaren Fälle des Versagens zu berücksichtigen und jeweils einen Handlungsräum für entsprechende Reaktionen bereit zu stellen, sollten auch bei der Bedienerschulung möglichst viele Fälle von Fehlfunktion angesprochen werden. Damit wird die Umgebung der Systeme ebenso wie ihre innere Struktur durchleuchtet. Dem Nutzer werden die Grenzen ihrer Anwendbarkeit und ihre Leistungsfähigkeit klar. Bei der Schulung an einer Textverarbeitung sollte man beispielsweise die Frage klären, wie sie auf eine Überfüllung der Festplatte reagiert und wie dieses Problem gelöst werden kann.

**Bewusstsein von Blindheit** („blindness“). Nach Winograd u. Flores tritt in jeder Situation, die wir im Hinblick auf beteiligte Objekte und deren Eigenschaften analysieren, überhaupt bei jeder Erstellung einer formalen Repräsentation, eine gewisse Blindheit in Bezug auf Phänomene auf, die außerhalb dieses Schemas liegen. Insbesondere führt die Einführung jedes neuen Informatiksystems dazu, dass bestimmte Sachverhalte außerhalb seines unmittelbaren Anwendungsbereichs keine Beachtung mehr finden. Dazu gehört zum Beispiel die Frage, ob es sich überhaupt um den richtigen Systemtypus handelt oder ob sich das Problem nicht generell einer Behandlung mit dem Rechner entzieht. Bei der Konstruktion guter Software ist man sich dieses unvermeidlichen Problems bewusst und versucht, die Auswirkungen soweit möglich abzumildern. Ebenso gehört zu einer guten Ausbildung auch die ausführliche Diskussion der möglichen „Blindheit“, die das behandelte System erzeugen könnte. Das kann anhand einer Besprechung seines Umfeldes, seiner Auswirkungen oder durch eine eingehende Reflexion von Sinn, Zweck, Effizienz oder Alternativen erfolgen.

Aus den letzten beiden Forderungen kann man unschwer ableiten, dass eine Bedienerschulung ohne die Vermittlung grundlegender Prinzipien der Konstruktion und Funktionsweise der behandelten Software wohl kaum zu einem befriedigenden Ergebnis führen kann. Davon wird später noch ausführlich die Rede sein.

### 1.1.3 Informatikunterricht

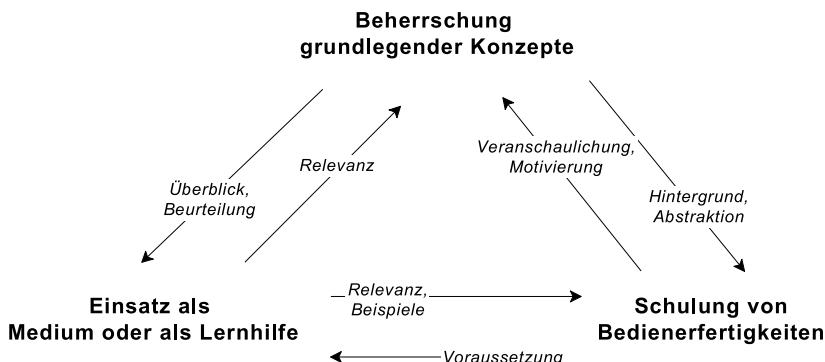
Neben der Unterstützung von Lernprozessen oder der Bedienerschulung können im Unterricht auch Prinzipien, Konzepte und Strategien zur Planung, Konstruktion, Beschreibung und Bewertung abstrakter Informatiksysteme thematisiert werden. Vor allem um diesen *Informatikunterricht* im eigentlichen Sinne geht es in diesem Buch. Dazu gehört auch die Behandlung von Anwendungen informatischer Prinzipien aus Bereichen ohne Beteiligung elektronischer Rechenanlagen, wie etwa die Strukturierung eines Großbetriebs. Als weitere Beispiele dafür könnte man die Modellierung einer Domäne mit Hilfe eines Entity-Relationship-Diagramms und dessen Umsetzung in Tabellen eines relationalen Datenbanksystems, die Behandlung der Datenstruktur einer Tabellenkalkulation oder die Entwicklung eines kleinen Algorithmus für ein Sortierverfahren anführen.

Leider wird im schulischen Umfeld der Begriff *Informatik* häufig für jede Art der Beschäftigung mit dem Computer missbraucht. Das Spektrum reicht dabei vom computergestützten Videokurs bis zu Fingerübungen auf der Tastatur.

### 1.1.4 Die Synthese: informatische Bildung

Die drei genannten Bereiche sind natürlich in vielfältiger Weise miteinander verflochten. Jeder Unterricht aus einem der drei Ausbildungsbereiche nützt Kenntnisse und Fertigkeiten aus den anderen beiden Bereichen (siehe Abb 1.1):

- Ein sinnvoller, ökonomischer Einsatz von Informatiksystemen als Lernhilfe verlangt natürlich sowohl gewisse Bedienerfertigkeiten als auch die Kenntnis einiger grundlegender Konzepte zur Beurteilung der Möglichkeiten und zur effizienten Nutzung des Systems.
- Die Schulung von Bedienerfertigkeiten gerät zur bloßen Vermittlung von „Rezepten“, wenn dabei nicht auch grundlegende Konzepte vermittelt werden. Diese Konzepte sichern auch die Übertragbarkeit der angeeigneten Bedienerfertigkeiten auf andere, ähnliche Systeme. Der nächstliegende Anwendungsbereich von Informatiksystemen ist für den Schüler wiederum deren Einsatz als Unterrichtshilfe.
- Die Vermittlung grundlegender Konzepte der Informatik wäre eine abschreckend abstrakte Veranstaltung ohne die Arbeit am Rechner. Dies ist aus meiner Sicht zur Veranschaulichung der Lerninhalte und zur Motivierung der Lernenden absolut unerlässlich. Dies setzt wiederum einige Fertigkeiten in der Bedienung von Hard- und Software voraus, so dass man auch hier nicht ohne ein gewisses Maß an Bedienerschulung auskommt.



**Abb. 1.1.** Verflechtung von Unterrichtshilfen, Bedienerschulung und Vermittlung grundlegender Konzepte

Aus dem Bewusstsein dieser Verflechtungen entstand auch das Konzept der *informationstechnischen Grundbildung* (ITG), wie es die Bund-Länderkommission (BLK) für Bildungsplanung und Forschungsförderung (1984) in ihrem „Rahmenkonzept Informationstechnische Bildung“ beschrieb:

Aufgaben der informationstechnischen Grundbildung sind:

- Aufarbeitung und Einordnung der Erfahrungen, die Schüler in ihrer Umwelt mit Informationstechniken machen,
- Vermittlung von Grundstrukturen, die den Informationstechniken zugrunde liegen,
- Einübung von einfachen Anwendungen der Informationstechniken,
- Vermittlung von Kenntnissen über die Einsatzmöglichkeiten und die Kontrolle der Informationstechniken,
- Darstellung der Chancen und Risiken der Informationstechniken,
- Einführung in Probleme des Persönlichkeits- und Datenschutzes,
- Aufbau eines rationalen Verhältnisses zu den Informationstechniken.

Leider hat die Erfahrung mit der Umsetzung dieses Rahmenkonzeptes inzwischen gezeigt, dass der darauf ausgerichtete Unterricht dabei in der Regel zu techniklastig ausfällt. Koerber u. Peters (1993) bemerken dazu:

Während es früher allein der Informatikunterricht war, der in der Schule sich dem modernen Thema der Informationstechnik zuwandte, ist es heute das, was als „Informationstechnische Grundbildung“ bezeichnet wird, das den Interessen der Schülerinnen und Schüler näher zu sein scheint. Es fehlt allerdings auch hier an einheitlichen Konzepten im Sinne einer echten Grund-Bildung. Beim Betrachten entsprechender Rahmenpläne entsteht der Eindruck, dass entweder produktbezogene Anwenderschulungen oder Programmierkurse im Kleinen in diesem Unterricht durchgeführt werden.

In Abgrenzung zu dieser „real existierenden informationstechnischen Bildung“ verwendet man heute zur Bezeichnung einer Ausbildungsform, in der die Betonung auf grundlegenden Konzepten von Informatiksystemen anstatt auf techni-

schen Einzelheiten liegt, meist den Begriff *informatische Bildung*, der von Norbert Breier (1994) geprägt wurde:

Informatische Bildung ist ... jener Teil der Allgemeinbildung, der die Welt unter informationellem Aspekt betrachtet, während die naturwissenschaftlichen Fächer den stofflichen oder energetischen Aspekt in den Mittelpunkt ihres Unterrichts stellen.

Informatische Bildung nach dieser Definition findet also auch außerhalb des Informatikunterrichts statt, nämlich immer dann, wenn mit allgemein bildenden Intentionen über „informationelle Aspekte“ gesprochen wird. Ein systematischer Informatikunterricht, der seinem Namen gerecht wird, ist allerdings nach meiner Auffassung ein unverzichtbarer Teil dieser informatischen Bildung.

## 1.2 Historische Ansätze für den Informatikunterricht

Bevor wir uns im nächsten Kapitel dem Versuch zuwenden, die Notwendigkeit eines systematischen Informatikunterrichts zu begründen, wollen wir zur weiteren Erläuterung der informatischen Bildung einen kurzen Blick in die Vergangenheit werfen. Dabei sollen einige der bisher im Informatikunterricht verfolgten didaktischen Ansätze aus heutiger Sicht beschrieben und gewertet werden. Für eine eingehendere Darstellung und Kritik der genannten Ansätze verweisen wir auf Schulz-Zander (1978), Forneck (1990), Baumann (1990) u. (1994).

### 1.2.1 Die Hardware als Ausgangspunkt

Der Ansatz stammt aus den späten 60er- und frühen 70er Jahren. Damals befanden sich die elektronischen Rechenanlagen noch am Anfang ihrer Entwicklung. Obwohl eigentliche Anwendungen noch kaum realisiert waren, gab es dennoch einige spektakuläre Leistungen der Rechnertechnik, die bei einem breiten Publikum große Beachtung fanden. Man denke nur an die erste bemannte Mondlandung im Jahre 1969. Vom Charakter her handelte es sich um eine der Kybernetik verpflichtete „Rechnerkunde“ mit dem Ziel, die mathematisch-technischen Grundlagen der Datenverarbeitung zu vermitteln (siehe Meißner (1972), (1975)).

Dass dieser Ansatz für das Gymnasium im Allgemeinen nicht der geeignete sein kann, wurde bereits hinlänglich diskutiert. So antworteten Wilfried und Ute Brauer (siehe Brauer u. Brauer (1973)) auf die Frage „Wie lässt sich Informatik im Unterricht verwirklichen?“ bereits im Jahre 1972:

Also soll das Hauptgewicht nicht gelegt werden auf das Kennenlernen des technischen Aufbaus von Rechenanlagen oder auf das Erlernen einer Programmiersprache. Die Schüler sollen vielmehr eingeführt werden in Methoden der Strukturierung, Mathematisierung und Algorithmisierung von Problemkreisen aus verschiedensten Gebieten (d.h. Methoden der Modellbildung und Problemlösung) sowie vor allem in die Methoden des systematischen Programmierens und Möglichkeiten des Einsatzes von Datenverarbeitungssystemen zur Behandlung komplexer Aufgaben (Informationssysteme, Simulation, heuristische Programmierung).

Weitere Kritiken dieses Ansatzes finden sich bei Schulz-Zander (1978), Forneck (1990) oder Baumann (1990). Dennoch bestimmt die Hardwareorientierung immer noch, wenn auch oft unausgesprochen, in nicht geringem Maße die Denkweise von Lehrkräften und damit zahlreiche Einzelentscheidungen über Methodik und Lerninhalte, insbesondere im Wahlunterricht der Sekundarstufe I. Diese Ausrichtung zwingt zu ständigem Umlernen beim Verfolgen der neuesten Hardware- und Betriebssystementwicklungen. Ein bleibender Beitrag zur Allgemeinbildung kann daraus wohl nicht entstehen.

### **1.2.2 Der Algorithmus als Maß aller Dinge**

Ab Mitte der 70er Jahre entstand unter dem Eindruck allgemeiner Anerkennung der Informatik als neuer wissenschaftlicher Disziplin ein Ansatz mit nun auch fachwissenschaftspropädeutischem Anspruch, der sogar die Abhaltung des Unterrichts in einer eigenen Sprache postulierte (siehe Brenner (1982)). Die Schüler sollten Algorithmen formulieren und programmieren, Probleme mit algorithmischem Hintergrund analysieren und Algorithmen in Programme umsetzen können. Methodisch folgte der Erkennung einer Problemstellung der Entwurf eines Lösungsplans, darauf die eigentliche Problemlösung mit Überlegungen zur Korrektheit und schließlich eine Diskussion möglicher Alternativen (siehe Balzert 1980)). Dabei kam vorzugsweise das Top-Down-Verfahren (siehe Bauer (1979)) zum Einsatz.

Im Zusammenhang mit dem allgemeinen didaktischen Prinzip altersgemäßer Vermittelbarkeit zeigte sich jedoch, dass die Forderung nach vollständiger Algorithmisierung der Problemstellungen diese auf relativ einfache Beispiele mit geringer Komplexität beschränkte. Gesellschaftliche Auswirkungen der Informatik oder die Beherrschung komplizierter Informationsstrukturen waren im Unterricht ohnehin nicht vorgesehen. Den Ansprüchen eines allgemein bildenden Schulfaches, das zu traditionellen Fächern in Konkurrenz treten könnte, wurde dieser Ansatz deshalb auf die Dauer nicht gerecht. Die Krise des Informatikunterrichts anfangs der 90er Jahre ist zu einem guten Teil auf die Erfahrungen mit überzogener Algorithmusorientierung zurückzuführen. So schreibt Burkert (1994):

Die Konstruktion von Software – im Großen wie im Kleinen – gehört zum ingenieurwissenschaftlichen Teil der Informatik und ist nicht allgemein bildend. Eine Orientierung am Algorithmus als zentralem Begriff der Informatik kann keine Legitimation für ein Fach der gymnasialen Oberstufe sein.

Auch Rechenberg (1994) warnt vor einer Überbetonung des algorithmischen Denkens im Unterricht:

Algorithmisches Denken ist nützlich und wertvoll, aber ist nicht das Höchste, wozu der Mensch imstande ist. Es ist nicht schöpferisch, sondern ein Denken zweiter Klasse.

### **1.2.3 Die vom Algorithmus beherrschte Anwendung**

Kurz nach dem algorithmischen Ansatz entwickelte sich, ausgelöst von der Forderung Robinsohns (siehe Robinson (1971)) nach einer Ausrichtung des Unterrichts an

konkreten Lebenssituationen anstatt an wissenschaftlichen Disziplinen, ein neuer Ansatz. Dieser wollte neben der Lösung praktischer Probleme auch die gesellschaftlichen, kulturellen, psychologischen Dimensionen dieser Lösungsfindung miteinbeziehen. Ausgehend von einer den Schülern bekannten Anwendung der Informatik sollte die Lösung einer bestimmten Problemstellung entwickelt werden, wobei wiederum die Algorithmik als Werkzeug zur Problemlösung betont wurde. Anschließend sollten dann möglichst alle Konsequenzen dieser Lösung beleuchtet werden.

Als Lernziele werden genannt (nach Brauer et al. (1976)): Die Schüler sollen in der Lage sein, algorithmische Lösungen von Problemen systematisch zu finden, algorithmische Lösungen als Programm zu formulieren, das Gelernte durch Anwendung auf praxisorientierte Probleme zu vertiefen, Auswirkungen der elektronischen Datenverarbeitung auf die Gesellschaft zu erkennen. Methodisch ist dabei zu beachten, dass Unterrichtsgegenstände nur dann Geltung beanspruchen können, wenn sie sich in einen Anwendungsbezug einbetten lassen (siehe Schulz-Zander (1978)). Nach Koerber u. Peters (1988) ergibt sich ein typischer Unterrichtsablauf, indem sich eine problembezogene, eine modellbezogene und eine informatikbezogene Ebene abwechseln, wobei diese Ebenen bei der Software-Erstellung von oben nach unten, bei der Programmierung von unten nach oben durchschritten werden.

Die Problematik dieses Ansatzes liegt wiederum im Postulat der Algorithmisierung aller behandelten Probleme. Die Fülle der intendierten Lernziele ist so nicht erschließbar, da für komplexere Probleme in der Schule oft kein Lösungsalgorithmus entwickelt werden kann. Somit überfordert dieser Ansatz durch seinen umfassenden Anspruch das Fach Informatik sowie Lehrer und Schüler, wie Forneck (1990) kritisch anmerkt.

#### 1.2.4 Der Benutzer im Mittelpunkt

Unter dem Eindruck des Vordringens der Mikroelektronik in Freizeit und Familienleben, der Weiterentwicklung kommerzieller Software mit ihren verminderten Einarbeitungszeiten und der neuartigen Vernetzung von Informations- und Kommunikationstechnologien entstand in den späten 80er Jahren ein neuer Ansatz. Dieser will unter Verzicht auf Programmierung mittels Benutzung von Anwendersystemen ausschließlich lebenspraktische Orientierung vermitteln. Er geht dabei nicht von Prinzipien der Fachdisziplin Informatik aus, sondern von den Auswirkungen der technologischen Entwicklung auf Individuum und Gesellschaft (siehe LISW (1987)).

Primäre Zielsetzungen sind informationstechnische Allgemeinbildung, Qualifizierung zum rationalen Umgang mit den Kommunikations- und Informationstechnologien, Steigerung der Beurteilungsfähigkeit ihrer Anwendungen und Auswirkungen sowie die Vermittlung der Fähigkeit, durch Ausbreitung und Weiterentwicklung der Technologien entstehende Probleme zu bewältigen.

Es gibt kein ausgezeichnetes methodisches Vorgehen, man durchläuft die folgenden Tätigkeitsgebiete in unterschiedlicher Reihenfolge (manche auch mehrfach): Finden, Erkennen und Analysieren eines Problems, Strukturieren des Problems und Entwickeln modellhafter Lösungsmöglichkeiten, Nutzen von Anwendersystemen und Programmierumgebungen, Beurteilen der Ergebnisse, Reflektieren und Bewerten der Nutzung der Technologien.

Für das Gebiet der ITG mag sich dieser Ansatz eignen, für einen systematischen Informatikunterricht fehlt ihm jedoch die intellektuelle Tiefe, die bei den bisher beschriebenen Ansätzen durch die Algorithmisierung erreicht wird. Die Schüler sehen bei der Verwendung vorgegebener Standardsoftware nur die äußeren Strukturen der Programmoberfläche, die eigentlich interessanteren inneren Konzeptionen wie Datenstrukturen und Problemlösungsstil bleiben ihnen verborgen, was sie in der Entwicklung ihrer Kritik- und Abstrahierungsfähigkeit stark einschränkt.

## 2 Wozu Informatikunterricht?

Bekanntlich lassen sich die geistige Aufnahmekapazität unserer Schüler sowie die gesamte für eine Schullaufbahn verfügbare Unterrichtszeit nicht beliebig erhöhen, weshalb die Einführung eines neuen Schulfaches zwangsläufig auf Kosten anderer Fächer gehen muss. Damit tritt das neue Fach „Informatik“ in Konkurrenz einerseits zu etablierten Fächern mit zum Teil jahrhundertelanger Unterrichtstradition, andererseits zu weiteren Wissenschaftsdisziplinen, die ebenfalls in die Stundentafeln unserer Schulen drängen. Um sich gegen diese Konkurrenten durchzusetzen, bedarf es einer überzeugenden Argumentationslinie. In diesem Kapitel wollen wir versuchen, eine solche Linie zu skizzieren.

### 2.1 Wozu überhaupt Unterricht?

Natürlich gibt es in jeder Gesellschaft feste Vorstellungen von Sinn und Zweck der Schulausbildung. In demokratischen Staaten existieren dafür in der Regel gesetzliche Vorgaben, die den Bildungsauftrag der öffentlichen Schulen festlegen.

#### 2.1.1 Die gesetzlichen Aufgaben der Schulen

Am Beispiel des *Bayerischen Erziehungs- und Unterrichtsgesetzes* (BayEUG) wollen wir nun einige mögliche Aufgaben öffentlicher Schulen untersuchen. Die Grundschulen werden dabei ausgeklammert, da Schüler dieser Altersstufen nicht über die für den Informatikunterricht notwendige Abstraktionsfähigkeit verfügen. Aus den einschlägigen Bestimmungen des BayEUG (siehe Tabelle 2.1) lassen sich für die allgemein bildenden Schulen Bayerns die folgenden drei Zielsetzungen ableiten:

**Allgemeinbildung.** Art. 2 verlangt als Ziel aller allgemein bildenden Schulen u.a.: „.... zu selbständigem Urteil und eigenverantwortlichem Handeln zu befähigen, ... erschließen den Schülern das überlieferte und bewährte Bildungsgut und machen sie mit neuem vertraut.“ In den folgenden schulartspezifischen Vorschriften wird diese Allgemeinbildung dann differenziert. Während die Hauptschule nach Art. 7 eine „grundlegende Allgemeinbildung“ vermittelt, leistet die Realschule laut Art. „allgemeine und berufsvorbereitende Bildung“. Das Gymnasium ist dagegen mit dem Anspruch einer „vertieften allgemeinen Bildung“ besonders gefordert.

**Allgemeine Berufsvorbereitung.** Bereits in Art. 2 des BayEUG werden alle Schulen verpflichtet, „auf Arbeitswelt und Beruf vorzubereiten“. Die Hauptschule trägt „Hilfen zur Berufsvorbereitung“ bei und „schafft Voraussetzungen für eine qualifizierte berufliche Bildung.“ An der Realschule findet „.... berufsvorbereitende Bildung.“ statt: „Sie legt damit den Grund für eine Berufsausbildung und eine spätere qualifizierte Tätigkeit.“ Das Gymnasium „schafft auch zusätzliche Voraussetzungen für eine berufliche Ausbildung außerhalb der Hochschule.“

**Allgemeine Studienvorbereitung.** Diese Zielsetzung beschränkt sich auf das Gymnasium: „....die für ein Hochschulstudium vorausgesetzt wird“.

**Tabelle 2.1.** Auszüge aus dem Bayerischen Erziehungs- und Unterrichtsgesetz (BayEUG)

Artikel	Text
Art. 2	(1) Die Schulen haben insbesondere die Aufgabe: ... zu selbständigem Urteil und eigenverantwortlichem Handeln zu befähigen, ... auf Arbeitswelt und Beruf vorzubereiten. (2) Die Schulen erschließen den Schülern das überlieferte und bewährte Bildungsgut und machen sie mit neuem vertraut.
Art. 7	(6) <sup>1</sup> Die Hauptschule vermittelt eine grundlegende Allgemeinbildung, bietet Hilfen zur Berufsfindung und schafft Voraussetzungen für eine qualifizierte berufliche Bildung.
Art. 8	(1) <sup>1</sup> Die Realschule vermittelt eine zwischen den Angeboten der Hauptschule und des Gymnasiums liegende allgemeine und berufsvorbereitende Bildung. ... <sup>3</sup> Sie legt damit den Grund für eine Berufsausbildung und eine spätere qualifizierte Tätigkeit in einem weiten Bereich von Berufen mit vielfältigen theoretischen und praktischen Anforderungen.
Art. 9	(1) <sup>1</sup> Das Gymnasium vermittelt die vertiefte allgemeine Bildung, die für ein Hochschulstudium vorausgesetzt wird; es schafft auch zusätzliche Voraussetzungen für eine berufliche Ausbildung außerhalb der Hochschule.
Art. 11	(3) <sup>1</sup> Die Berufsschulen haben ... die für den Ausbildungsberuf oder die berufliche Tätigkeit erforderlichen fachtheoretischen Kenntnisse zu vermitteln und die fachpraktischen Kenntnisse zu vertiefen; ...

Die beruflichen Schulen sind dagegen naturgemäß vor allem auf *fachspezifische Berufsvorbereitung* ausgerichtet (siehe Art. 11 BayEUG in Tabelle 2.1). Von einer *fachspezifischen Studienvorbereitung* ist dagegen in den gesetzlichen Rahmenbedingungen nirgends die Rede. Dennoch wird sie zumindest am Gymnasium in den Leistungskursen durchaus betrieben.

Im Vergleich zu den Begriffen „allgemeine Berufsvorbereitung“ und „allgemeine Studienvorbereitung“ ist „Allgemeinbildung“ weitaus schwieriger zu fassen. Deshalb folgt eine kurze Begriffsklärung, die sich auf Aspekte beschränkt, die für den Informatikunterricht von Bedeutung sein könnten.

## 2.1.2 Allgemeinbildung

Ein sehr bekannter und besonders in der Informatikdidaktik breit akzeptierter Allgemeinbildungsbegriff (siehe etwa Engbrink (1995), Baumann (1996), Schubert (1997)) stammt von Bussmann u. Heymann (1987). Danach ist für allgemein bildende Unterrichtsbemühungen zu fordern (zitiert nach Engbrink (1995)):

1. Vorbereitung auf zukünftige Lebenssituationen. D.h. allgemein bildende Schulen sollen Qualifikationen vermitteln,
  - a) die zur Bewältigung realer und auf absehbare Zeit in unserer Gesellschaft verbreiteter Lebenssituationen beitragen,
  - b) die nicht auf die Ausübung eines bestimmten Berufes hin ausgerichtet sind,
  - c) von denen anzunehmen ist, dass sie nicht gleichsam automatisch, nebenher von jedem Heranwachsenden erworben werden und
  - d) die durch eine gewisse Universalität, also Anwendbarkeit in sehr verschiedenen Situationen gekennzeichnet sind.
2. Stiftung kultureller Kohärenz,
3. Aufbau eines Weltbildes,
4. Anleitung zum kritischen Vernunftgebrauch,
5. Entfaltung eines verantwortlichen Umgangs mit den erworbenen Kompetenzen,
6. Stärkung des Schüler-Ichs.

Klafki (1964) billigt in seiner *Göttinger Schule* (siehe Teil A, Abschnitt 3.1) im Vergleich zu anderen allgemeinen Didaktikern der Allgemeinbildung einen besonders hohen Stellenwert zu. Seiner Meinung nach hat man bei der Auswahl allgemein bildender Inhalte die folgenden Fragen zu beantworten:

1. Lässt der (geplante) Inhalt zu, dass meine Schüler eine allgemeine Kenntnis, Einsicht erwerben können?
2. Ist der Inhalt so strukturiert, dass er neben seiner Besonderheit auch ein über sich hinausweisendes Merkmal aufweist?
3. Lässt sich das Allgemeine an diesem Inhalt auch von meinen Schülern in dieser Lernsituation erfassen?
4. Sollten meine Schüler dies Allgemeine überhaupt erwerben?

An diesen Kriterien wird sich die Informatik messen müssen, wenn sie den Anspruch erhebt, in den Fächerkanon der allgemein bildenden Schulen aufgenommen zu werden. Im nächsten Abschnitt werden wir begründen, warum in unserer Gesellschaft ein systematischer Informatikunterricht unbedingt zur Allgemeinbildung gehört.

## 2.2 Bildungsauftrag und Informatikunterricht

Natürlich ist die Umsetzung des oben beschriebenen Bildungsauftrags unserer Schulen davon abhängig, in welcher gesellschaftlichen Umgebung unsere Schüler nach ihrem Abschluss leben und arbeiten werden. Dazu muss man berücksichtigen, wie lange ihre Ausbildung voraussichtlich dauern wird. Eine Schülerin der 5.

Jahrgangsstufe eines Gymnasiums wird beispielsweise bis zur Aufnahme ihrer Berufstätigkeit noch mindestens 13 Jahre warten müssen, falls sie beabsichtigt, ein wissenschaftliches Studium zu absolvieren. Ein Schüler der 10. Klasse einer Realschule dagegen könnte bereits ein Jahren später als Auszubildender im Berufsleben stehen.

### **2.2.1 Für welche Welt bilden wir unsere Schüler aus?**

Niemand wird heute ernsthaft bestreiten wollen, dass Informatiksysteme in unserer Gesellschaft eine dominante Rolle übernommen haben. Es gibt unzählige Symptome, mit denen man diese Behauptung belegen kann. Wir wollen hier nur einige wenige aufzählen.

**Informationsflut.** Bereits eine oberflächliche Betrachtung der explosiven Entwicklung des Internet lässt das gigantische Ausmaß der Informationsflut erahnen, die in nächster Zukunft über uns hereinbrechen wird. Das Internet-Software-Konsortium gibt auf seiner Webseite (siehe [www.isc.org](http://www.isc.org)) an, dass im Internet-Namensdienst (*Domain Name System*) im Januar 2007 ca. 433 Millionen Rechner registriert waren, das sind ca. 22 mal so viele wie zehn Jahre zuvor und ca. dreimal so viele wie im Jahr 2002.

Dieser immense technische Aufwand spiegelt sich auch inhaltlich in einer gigantischen Informationsmenge wider. Laut eigenen Angaben (<http://www.google.de/intl/de/corporate/index.html>) durchsucht die Suchmaschine *Google*® ([www.google.com](http://www.google.com)) nach eigenen Angaben regelmäßig 8 Milliarden Webseiten in mehr als 100 Sprachen. Bei diesen Datenmassen ist es kein Wunder, dass eine Anfrage nach Dokumenten zur Stichwortkombination „George Bush“ auf Google im April 2007 rund 126 Millionen Treffer erbringt. Die effiziente Suche nach brauchbaren Informationen wird in Zukunft wohl eine der größten Anforderungen der Berufswelt darstellen. Peter Glaser (1995) schrieb dazu bereits vor vielen Jahren:

Die großen Fragen einer Informationsgesellschaft lauten also: Wie schützt man sich vor Daten? Wie vor lebenszeitfressendem, unnötzen Wissen? Wie lernt man, Informationen nach seinen Interessen zu bewerten, auszuwählen und zu strukturieren? Und wie und mit welchen Instrumenten - also Programmen - kann man effizient, alltagstauglich und elegant an der Erkundung und Benutzung der elektronischen Welt teilhaben? Wer darauf Antworten hat, hat Klasse.

**Arbeitsmarkt.** Nach einer Phase der Konsolidierung und größeren Verschiebungen von Arbeitsplätzen ins Ausland bietet die IT-Branche mittlerweile wieder hervorragende Berufsaussichten für qualifizierte Fachkräfte. Nach einer Studie der BITKOM vom Februar 2007 ([http://www.bitkom.de/de/presse/8477\\_44206.aspx](http://www.bitkom.de/de/presse/8477_44206.aspx)) stehen in der ITK-Branche 20.000 offene Stellen zur Verfügung. 63 Prozent der Unternehmen schaffen dieser Studie zufolge im Jahr 2007 neue Jobs und für jede zweite Firma ist der Fachkräftemangel ein großes Problem. Gefragt sind vor allem Softwareentwickler, IT-Berater und IT-Projektmanager. Rund zwei Drittel der Unternehmen suchen Mitarbeiter mit einschlägigen Kenntnissen.

**Wirtschaftliche Bedeutung.** Die immense Bedeutung der Informationstechnik für die Wirtschaft wird z.B. durch eine aktuelle Studie des Bundesverbands Informationswirtschaft, Telekommunikation und neue Medien e.V. (BITKOM) vom März 2007 belegt, die das Marktvolumen im Informations- und Telekommunikationsbereich in Deutschland auf ca. 140 Milliarden Euro taxiert.

Zwei konkrete Marktsegmente verdeutlichen diese enormen wirtschaftlichen Möglichkeiten: Einer der aktuellen Trends in der Automobil-Elektronik geht Richtung Infotainment, das vom Navigationssystem über das Herunterladen von Musik aus dem Internet bis zum Video-Angebot für Fahrgäste reicht. Nach einer Studie von Frost & Sullivan aus dem Jahre 2003 (Frost&Sullivan Report B175, 09/03) soll der europäische Markt für Infotainment-Technologien im Auto bis zum Jahre 2010 auf mehr als neun Milliarden Euro ansteigen. Nach einer anderen Studie derselben Firma (Frost & Sullivan Report A276, 01/03) wird der Weltmarkt für Wireless-LAN Technologien im Jahr 2009 ca. 1,5 Milliarden US\$ betragen.

**Allgegenwart.** Der ungeheure Einfluss der Mikroelektronik auf unser Berufs- und Privatleben zeigt sich auch in ihrer Allgegenwart. Programmierbare Mikroprozessoren finden sich beinahe überall in unserer Umgebung. In einem Kraftfahrzeug der Oberklasse arbeiten beispielsweise weit mehr als 50 Prozessoren, die von der Lenkunterstützung über den Zugang per Fingerabdruck oder Stimmbild bis zum ABS, von der Scheibenwischersteuerung über die Sitzheizung bis zur Einspritzung unzählige Abläufe regeln.

Auf ähnlich dramatische Weise sorgen Mobiltelefone für die Verbreitung von hochkomplexen miniaturisierten Informatiksystemen: Laut einer Studie von Gartner aus dem Jahr 2005 sollen im Jahr 2009 mehr als 1 Milliarde Mobiltelefone verkauft werden.

Personal Computer erobern mehr und mehr unsere Umgebung: Das Bundesministerium für Bildung und Forschung zitiert in seinem Aktionsprogramm (siehe Bundesministerium für Bildung und Forschung (1999)) eine Untersuchung (Quelle FVIT), nach der in den USA auf 100 Einwohner bereits 67 Personal Computer kommen, in Deutschland immerhin bereits 37.

## 2.2.2 Ist Medienerziehung nicht genug?

Angesichts dieser drastischen Entwicklungstendenzen herrscht allgemein Konsens über die Verpflichtung der Schulen, sich mit modernen Informations- und Kommunikationssystemen zu beschäftigen, vor allem im Hinblick auf die Modebegriffe „Multimedia“ oder „Telekommunikation“. Dazu wurden unzählige politische Programme und Absichtserklärungen verfasst, die sich aber unglücklicherweise meist auf den Medienaspekt beschränken. Als Beispiel sei hier nur das Aktionsprogramm der Bundesregierung zum Thema „Innovation und Arbeitsplätze in der Informationsgesellschaft des 21. Jahrhunderts“ genannt (siehe Bundesministerium für Bildung und Forschung, Bundesministerium für Wirtschaft und Technologie (1999)). Darin heisst es unter der Überschrift „Multimedia in der Bildung fördern“:

Die Wissensgesellschaft kann nur der meistern, der über eine angemessene technische Ausstattung und eine vernetzte Infrastruktur verfügt und die modernen Informations- und Kommunikationstechniken beherrscht. Dabei bestehen im gesamten Bildungssektor nach wie vor Defizite. An den allgemein bildenden und berufsbildenden Schulen fehlt es oft an der Qualifikation der Lehrerinnen und Lehrer im Umgang mit den neuen Medien und der zugrundeliegenden IT-Technik sowie an bedarfsgerechter und flexibel nutzbarer Bildungssoftware. Darüber hinaus fehlt es in allen Schultypen an einer hinreichenden Ausstattung mit modernen Informations- und Kommunikationstechniken und ihrer Vernetzung.

Bezeichnenderweise werden auch in diesem Papier die Defizite hauptsächlich im Bereich der technischen Infrastruktur und bei den Kompetenzen der Lehrkräfte gesehen. Wie die verlangte „Beherrschung moderner Informations- und Kommunikationstechniken“ ohne systematischen Informatikunterricht vermittelt werden soll, wird leider nicht weiter erklärt. Überhaupt werden mit der „Medienerziehung“ sehr hohe Erwartungen verknüpft, wie etwa von der Bund-Länderkommission für Bildungsplanung und Forschungsförderung in ihrem Papier zur Medienerziehung (1995):

Medienerziehung ist als schulische Aufgabe im Zusammenhang mit den allgemeinen und verbindlichen Erziehungs- und Bildungsvorstellungen zu sehen. Geht man von dem Grundgesetz und den Länderverfassungen aus, so hat das Recht auf freie Entfaltung der Persönlichkeit in sozialer Verantwortung als Leitidee für den Erziehungs- und Bildungsbereich zu gelten. Nimmt man bildungspolitische Auslegungen und fachliche Konkretisierungen hinzu, so kann als allgemeine Leitvorstellung für Erziehung und Bildung ein sachgerechtes, selbstbestimmtes und kreatives Handeln in sozialer Verantwortung postuliert werden. An dieser Leitvorstellung muss sich auch die Medienerziehung orientieren und zugleich messen lassen.

Wenn es also genügt, sich im Unterricht aller Fächer mit den neuen Medien und ihren gesellschaftlichen Auswirkungen zu beschäftigen, um die Schüler in die Lage zu versetzen, sachgerecht, selbstbestimmt und kreativ zu handeln, wozu sollte man dann noch einen eigenen Informatikunterricht brauchen? Wie unrealistisch diese Erwartungen an die Medienerziehung sind, beschrieben Wilfried und Ute Brauer (siehe Brauer u. Brauer (1989)) ebenso weitsichtig wie eindrucksvoll bereits vor 10 Jahren, als sie sich über die Konsequenzen dreier unterschiedlicher Ausbildungsstrategien in einer (damals utopischen) Informationsgesellschaft Gedanken machten:

**Das bedienerfreundliche Werkzeug.** Die Behandlung des Computers als *bedienerfreundliches Werkzeug* („user friendly black-box tool“) zum „Entdeckenden Lernen“ oder als intelligentes Tutorsystem ohne Behandlung seiner grundlegenden Funktionsprinzipien führt ihrer Meinung nach zur Bildung eines *mental Modells*, das dem einer *ultimativen Autorität* entspricht. Die Menschen sind den Systemen auf Gedeih und Verderb ausgeliefert und werden abhängig von einer kleinen Gruppe von Spezialisten, die solche Systeme konstruieren kann.

**Der nette Gesprächspartner.** Die zweite Möglichkeit, die Behandlung als *netten Gesprächspartner* („partner for communication“) der Kinder führt u.a. dazu, dass die Kommunikation über die geschriebene Sprache mehr und mehr verschwindet. Die

leichte Verfügbar- und Konsumierbarkeit großer Wissensmengen via Multimedia könnte die Menschen zum Glauben verführen, dass sie keine tieferegreifende Ausbildung mehr benötigten, da alles Wissen so leicht zugänglich ist. Dadurch sind sie Manipulationen schutzlos ausgeliefert. Sie werden weniger rational als emotional handeln. Das resultierende mentale Modell des Computers wird *anthropomorph* sein, ähnlich der metaphorischen Erklärung von Naturphänomenen mittels Elfen, Zauberern oder Hexen in vorgeschichtlicher Zeit.

**Die informatische Maschine.** Zur Vermeidung der oben genannten Probleme schlagen die Autoren vor, den Rechner als eine *reine Konstruktion von Menschenhand* („informatical machine“) zu behandeln, die nach den Gesetzen der Physik unter Benutzung von mathematischen und informatischen Methoden als Werkzeug zur Steigerung der Effizienz menschlicher Tätigkeiten, zur Behandlung komplexer Informationen und zur Befreiung von ermüdenden, oft zu wiederholenden Handlungsabläufen geschaffen wurde. Nur Menschen mit einem grundlegenden Verständnis des Computers und seiner Konstruktionsprinzipien können diesen in einfühlsamer, fruchtbarer und menschlicher Weise nutzen. Dazu benötigen sie geeignete mentale Modelle der verwendeten Informatiksysteme. Dies kann nicht mittels „learning by doing“ erreicht werden, es ist vielmehr eine sorgfältige und gründliche Ausbildung in Informatik und benachbarten Disziplinen vonnöten.

Auch die Gesellschaft für Informatik stellt klar, dass die Medienerziehung ihre Aufgaben ohne eine Vermittlung grundlegender Konzepte von Informatiksystemen nicht erfüllen kann (siehe Hauf-Tulodziecki (1999)). Zum Aufgabenbereich „b) Einblick in Wirkungsweise und Produktionsbedingungen von Medien“ des BLK-Konzeptes (siehe Abschnitt 1.1.1 auf Seite 44) merkt sie in ihren Empfehlungen „Informatische Bildung und Medienerziehung“ an:

Bezogen auf computerbasierte Medien umfasst das Durchschauen von Produktionsbedingungen ein Verständnis für prinzipielle Verfahren der Softwareentwicklung, für Digitalisierung, Strukturierung von Gegenstandsbereichen, Modellbildung und algorithmische Lösungsverfahren, die der Programmierung und Implementation ausführbarer Programme vorausgehen. Im Hinblick auf die Verbreitung von Online-Medien sind Grundkenntnisse im Bereich der Vernetzung erforderlich.

In einer detaillierten Fallstudie legen Friedrich u. Neupert (1997) sehr überzeugend anhand einiger Fehlermeldungen eines Webbrowsers dar, wie utopisch die Vorstellung einer bewussten, effizienten Handhabung solcher Systeme ohne entsprechende Grundlagenkenntnisse ist. Abschliessend konstatieren sie:

Es ergeben sich also aus der einfachen und vielfältigen Nutzung des Mediums Internet in verschiedenen Lernsituationen Probleme, die Ansatzpunkte für eine fachlich fundierte Informatikausbildung begründen. Ohne grundlegende Kenntnisse über informatische Inhalte der Arbeit mit Netzwerken können diese auch nicht in allen Alltagssituationen effektiv genutzt werden. Die dafür notwendige Kompetenz lässt sich eben nicht immanent vermitteln, sondern bedarf einer systematischen Aneignung der entsprechenden Grundlagen. Hier wandeln sich die Ansprüche an den Unterricht. Insbesondere bedarf es geradezu entsprechender Lerneinheiten im Rahmen einer informatischen Bildung, sinnvollerweise in einem Fach Informatik.

Insgesamt ergibt sich als Fazit: Die Medienerziehung kann ihre hoch gesteckten Ziele nur erfüllen, wenn sie von angemessener informatischer Bildung in Form eines systematischen Informatikunterrichts begleitet wird.

### **2.2.3 Der allgemein bildende Wert informatischer Bildung**

Natürlich gibt es auch unabhängig von den sich rasant verbreitenden neuen Medien weitere Gründe für eine planmäßige Informatikausbildung an unseren Schulen. Dazu muss vor allem ihr Beitrag zur Allgemeinbildung nachgewiesen werden. Wir wollen versuchen, solche Beiträge in den Kriterienkatalog von Bussmann und Heymann (siehe Abschnitt 2.1.2) einzuordnen:

**1. Vorbereitung auf zukünftige Lebenssituationen.** Hoppe u. Luther (1996) führen als zentrales Argument für die Informatik als Bestandteil der Allgemeinbildung an:

Informatik repräsentiert und transportiert mehr als jedes andere Fach, insbesondere auch mehr und besser als die Mathematik, das historisch und kulturell bedeutsame Bemühen um die Automatisierung geistiger Tätigkeiten. ... Das Wissen um die Automatisierung geistiger Tätigkeiten, ihre nachgewiesenen Grenzen und ihre vielfältigen Möglichkeiten ist für das Selbstverständnis des modernen Menschen ähnlich bedeutsam wie z.B. die Kontroverse um genetische versus soziale Determinierung im Bezug zur Biologie und zu den Sozialwissenschaften...

Peter Rechenberg (1994) betont als wesentlichen Bildungsgehalt der Informatik ebenfalls das Wissen um die prinzipiellen Grenzen der Automatisierbarkeit. Friedrich, Schubert u. Schwill (1996) geben unter anderem als Begründung für den allgemein bildenden Wert der Informatik an, dass die Schule gemäß ihres Bildungs- und Erziehungsauftrages:

zur Auseinandersetzung mit komplexen Denksystemen anleiten und Anwendungs- und Handlungsmöglichkeiten sicher ausprägen

soll. Auch Rechenberg (1994) sieht in der Anleitung zur Meisterung komplexer Zusammenhänge einen wichtigen Beitrag der Informatik zur Bildung.

Die Kriterien 1.c) und d) von Bussmann und Heymann sprechen übrigens einer reinen Bedienerschulung jeden Gehalt an Allgemeinbildung ab.

Zur näheren Beleuchtung der Lebenssituationen, auf die der Informatikunterricht vorbereiten soll, scheint es nützlich, sich mögliche Rollen des Menschen gegenüber einem Informatiksystem zu vergegenwärtigen (siehe Tabelle 2.2). Der Schulbildung kommt dabei umso mehr Bedeutung zu, je weniger zu erwarten ist, dass die Inhaber der Rollen in einer späteren Spezialausbildung auf ihre Aufgaben vorbereitet werden. Vor allem für Entwickler, Administratoren und (professionelle) Benutzer finden sich meist spezielle Aus- oder Weiterbildungsmassnahmen. Die Spannweite reicht dabei von der firmeninternen Produktschulung bis zum universitären Informatikstudium.

**Tabelle 2.2.** Menschliche Rollen gegenüber Informatiksystemen

Rolle	Beispiele	Beschreibung
Entscheider	Vorstand, Abteilungsleiter	Verantwortung für Geldmittel, Entscheidung über Kauf oder Eigenentwicklung eines Systems
Planer	Projektleiter	Planung und Leitung von Entwicklung oder Einrichtung der Systeme
Entwickler	Programmierer, Sachbearbeiter	Produktion von Systemen oder Teilen von Systemen
Administratoren	Netzwerk- betreuer	Installation, Wartung, Information
Nutzer	Sekretärin, Personalchef	Direkte oder indirekte Nutznießer
Betroffene	Sachbearbeiter, Setzer	Veränderung oder Wegfall des Arbeitsplatzes durch Einrichtung oder Betrieb der Systeme

**2. Stiftung kultureller Kohärenz.** Hierzu kann die Informatik z.B. durch eine Vereinheitlichung der beim Umgang mit Informatiksystemen verwendeten Fachsprache beitragen. Ein rechtzeitiger systematischer informatischer Pflichtunterricht für alle Schülerinnen und Schüler könnte auch dafür sorgen, dass die oft beklagten geschlechtlich oder sozial bedingten Unterschiede in der Beherrschung von Informatiksystemen gar nicht erst aufkommen können.

**3. Aufbau eines Weltbildes.** Norbert Breier (1994) sieht die Materie (= reale Welt) einerseits in Energie und andererseits in Information (= Struktur) aufgeteilt und leitet daraus die Notwendigkeit her, um den Begriff „Information“ eine neue Art von Informatikunterricht zu konzipieren. Zur Ausbildung von Sach-, Handlungs-, und Beurteilungskompetenz im Umgang mit Informationen und Informatiksystemen ist seiner Meinung nach eine ebenso tief greifende Behandlung des Themenkomplexes „Information“ nötig, wie die etwa in den Fächern Physik und Chemie mit der anderen Materiekomponente „Energie“ geschieht.

**4. Anleitung zum kritischen Vernunftgebrauch.** Zu einem kritischen Vernunftgebrauch in der Informationsgesellschaft gehört unbedingt auch das Wissen um die prinzipiellen Möglichkeiten und Grenzen dieser Systeme, also z.B. Grundkenntnisse über

- Berechenbarkeit und Komplexität,
- Funktionsweise von Rechenanlagen und Netzen,
- das Phänomen der „Blindheit“ (siehe Abschnitt 1.1.2).

**5. Stärkung des Schüler-Ichs.** Eine angemessene informatische Grundausbildung führt dazu, dass die Schülerinnen und Schüler mit nüchterner, selbstsicherer Gelassenheit mit Informatiksystemen umgehen. Die Sicherheit, diese Systeme zu durchschauen, gibt ihnen das Gefühl, sie zu beherrschen, anstatt von ihnen beherrscht zu werden. Weder „Computerangst“ noch „Computersucht“ können so auftreten.

Eine systematische Darstellung des Beitrags der Informatik zur Allgemeinbildung findet sich auch in den bisher letzten Empfehlungen der Gesellschaft für Informatik zum Thema Informatikunterricht (siehe Schulz-Zander et al. (1993)):

- Förderung eines verantwortungsvollen Umgangs mit Informationen und Erziehung zu verantwortungsvollem Handeln,
- Reflexion des Verhältnisses von Menschen zur Informationstechnik,
- Förderung eines gleichberechtigten Zugangs zur Technik,
- Vermittlung verschiedener Problemlösungs- und Gestaltungsmethoden und deren Beurteilung,
- Förderung schöpferischen Denkens,
- Förderung der Fähigkeit zur Kommunikation und Kooperation.

## 2.2.4 Informatik zur Berufsvorbereitung

Neben der allgemeinen Berufsvorbereitung in allen Schulararten kommt besonders in der Ausbildung zu den so genannten neuen IT-Berufen (wie Informations- und Telekommunikationselektroniker o.ä.) auf die beruflichen Schulen auch die Vermittlung spezieller informatischer Berufskenntnisse zu. In Analogie zur klassischen Handwerksausbildung könnte man argumentieren, dass man wahre Meisterschaft nur durch solide Kenntnisse von Werkzeug und Rohmaterial erlangen kann. In Bezug auf die informative Bildung sind die Werkzeuge *Informatiksysteme*, der Rohstoff *Daten oder Informationen*. Von einem guten Handwerker würde man erwarten, dass er die folgenden Fragen beantworten kann (siehe Hubwieser u. Broy (1999)):

- In welcher Situation, zu welchem Zweck soll ich welches Werkzeug und/oder welches Material verwenden?
- Wie kann ich die besten Resultate aus dem gegebenen Material holen, wie behandle ich seiner Struktur gemäß richtig?
- Was ist grundsätzlich möglich und was nicht? Bis zu welcher Belastung wird das Material halten? Welchen Zeitraum wird mein Werk überdauern?
- Wie kann man Konstruktion und Produktion der Werkzeuge optimieren ?
- Wie kann ich im Hinblick auf Material-, Zeit-, Energie- und Personalkosten optimal arbeiten?

- Gibt es ein besseres oder billigeres Werkzeug oder Material für diesen Zweck?
- Welchen Schaden kann meine Arbeit anrichten, mit welchen Gefahren muss ich rechnen?

In einer fachgerechten Informatikausbildung, die über die bloße Allgemeinbildung hinausgeht, müsste man demnach Anwendungen, Strukturen, Grenzen, Kosten, Alternativen und Auswirkungen der Verarbeitung von Informationen durch Informatiksysteme eingehend behandeln.

### **2.2.5 Allgemeine Studienvorbereitung**

Dieses Ausbildungsziel trifft vor allem auf Gymnasien und Fachoberschulen zu. Der Informatikunterricht kann hier vor allem Methoden und Kenntnisse

- zur Beschreibung und Lösung von Problemen,
- zur Beschaffung und Darstellung von Informationen,
- zur Beherrschung und effizienten Nutzung von Informatiksystemen

beitragen.

### 3 Entwurf einer Unterrichtsmethodik

Nachdem wir im ersten Kapitel geklärt haben, was wir unter Informatikunterricht verstehen wollen und im zweiten dargelegt wurde, warum dieser unserer Meinung nach eine unverzichtbare Komponente einer tragfähigen Allgemeinbildung darstellt, soll in diesem Kapitel nun eine geeignete Unterrichtsmethodik skizziert werden. Dies geschieht bewusst *vor* der Auswahl möglicher Lerninhalte, da die Beurteilung der Umsetzbarkeit dieser Inhalte stark von der geplanten Methodik abhängt. Überhaupt ist eine Abgrenzung zwischen dem *Wie* (der Methodik) und dem *Was* (den Lerninhalten) des Informatikunterrichts oft sehr schwierig. So stellt Modellierung und Simulation einerseits ein wichtiges methodisches Grundprinzip des Unterrichts dar, während andererseits gewisse Modellierungstechniken als zentrale Lerninhalte auftreten.

#### 3.1 Lernpsychologisches Fundament

Aufbauend auf den Erkenntnissen der Lernpsychologie (siehe Teil A, Kapitel 1) sollte der Unterricht die folgenden Forderungen erfüllen:

- Erzeugung einer entspannten Arbeitsatmosphäre, in der die Schüler ohne Stoffdruck Zeit haben, auch spezielle Interessen und Bedürfnisse zu befriedigen. Damit sollen Motivation und Aufmerksamkeit der Schüler gefördert und dauerhaft aufrechterhalten werden (vgl. Teil A, Abschnitt 1.1.1).
- Einordnung der Lerninhalte in größere Sinnzusammenhänge sowie eine deutliche Strukturierung der Stoffe, um den Schülern die Bildung *präpositionaler Netzwerke* zu ermöglichen (vgl. Teil A, Abschnitt 1.1.2).
- Förderung einer aktiven Auseinandersetzung mit dem Stoff, wobei unbedingt vor der Präsentation von Lösungen ein ausreichendes Problembewusstsein erzeugt werden muss. Diese Forderungen entstammen den gemeinsamen Erkenntnissen aller gemäßigt konstruktivistischen Lernansätze, die betonen, dass Wissen vom Lernenden aktiv konstruiert wird (vgl. Teil A, Abschnitt 1.4).
- Anbieten verschiedener Perspektiven und Zugänge zum selben Thema im Sinne von *Cognitive Flexibility* (vgl. Teil A, Abschnitt 1.4).
- Erzeugung möglichst authentischer Problemsituationen, um mit den Schülern Problemlöseverhalten in einer Umgebung zu trainieren, in der dieses Verhalten tatsächlich auch benötigt wird (*Cognitive Apprenticeship*, vgl. Teil A, Abschnitt 1.4).
- Altersgemäße Darbietung der Lerninhalte. Nach den Erkenntnissen der Entwicklungspsychologie Piagets (vgl. Teil A, Abschnitt 1.3) entwickelt sich

z.B. in der Gegend der 6. Jahrgangsstufe bei den Kindern die Fähigkeit, hypothetische Vorgänge in ihre Überlegungen mit einzubeziehen, die für die Erarbeitung von Problemlösestrategien im Rahmen des Informatikunterrichts (siehe nächster Abschnitt) von großer Bedeutung ist.

## 3.2 Methodische Prinzipien

Neben den allgemeinen didaktischen Prinzipien (siehe Teil A, Kapitel 2) gibt es weitere methodische Prinzipien, die speziell für den Informatikunterricht von besonderer Bedeutung sind, wie Problemorientierung oder Modellbildung und Simulation.

### 3.2.1 Problemorientierung

Nach den Erfahrungen mit der Umsetzung abstrakter Konzepte wie der Gruppentheorie oder der Abbildungsgeometrie im Mathematikunterricht des Gymnasiums scheint eine Vermittlung der naturgemäß abstrakten informatischen Lerninhalte nur dann erfolgversprechend, wenn durch konkrete, anschauliche Problemstellungen eine erhöhte Aufnahmefähigkeit der Schüler geschaffen wird.

Edelmann (1986) beschreibt den Begriff „Problem“ (aus lernpsychologischer Sicht) folgendermaßen:

Ein Problem ist also durch drei Komponenten gekennzeichnet:

- Unerwünschter Anfangszustand,
- erwünschter Zielzustand,
- Barriere, die die Überführung des Anfangszustandes in den Zielzustand im Augenblick verhindert.

Davon zu unterscheiden ist die *Aufgabe* (ebenfalls nach Edelmann (1986)):

Bei einer Aufgabe verfügen wir über Regeln (Wissen, Know-how), wie die Lösung zu erreichen ist.

Die Informatik kann in Form von Strukturierungshilfen und Simulationsmöglichkeiten besonders wertvolle Beiträge zur Problemlösefähigkeit der Schüler leisten, wie Brauer u. Brauer (1973) bereits zur Zeit der „Rechnerkunde“ feststellten:

Also sollte das Hauptgewicht nicht gelegt werden auf das Kennenlernen des technischen Aufbaus von Rechenanlagen oder auf das Erlernen einer Programmiersprache. Die Schüler sollten vielmehr eingeführt werden in

1. Methoden der Strukturierung, Mathematisierung und Algorithmisierung von Problemkreisen aus verschiedenen Gebieten <d.h. Methoden der Modellbildung und Problemlösung> sowie
2. vor allem die Methoden des systematischen Programmierens und

3. Möglichkeiten des Einsatzes von Datenverarbeitungssystemen zur Behandlung komplexer Aufgaben (Informationssysteme, heuristische Programmierung, Simulation).

Mittlerweile ist das *Problemlösen mit Informatiksystemen* eine der allgemein anerkannten Leitlinien der informatischen Bildung (siehe Friedrich (1995)). Die Struktur des Problemlöseprozesses stellt deshalb gerade für den Informatikunterricht ganz im Sinne von Roth (1963) (vgl. Teil A, Abschnitt 4.5.1) ein geeignetes Mittel zur *Artikulation* des Informatikunterrichts dar. Jeder neue Stoff sollte anhand von Problemen aus dem Erfahrungsbereich der Schüler (s.a. Robinson (1971)) eingeführt werden: Einer Motivationsstufe folgt eine Stufe der Schwierigkeiten, die dann in einer nächsten Stufe gelöst werden.

Die didaktischen Fähigkeiten des Lehrers zeigen sich dabei in der Auswahl geeigneter Probleme, deren Komplexität einerseits so hoch sein sollte, dass sie von den Schülern ohne die zu erlernenden Konzepte nicht oder nur unter erheblich höherem Aufwand gelöst werden können. Andererseits darf der intellektuelle Horizont der Schüler nicht überschritten werden. Im Sinne von Edelmann (1986) wären Probleme, die sich durch *Anwendung von Strategien* oder durch *Systemdenken* lösen lassen, optimal. Das Ergebnis des Problemlöseprozesses ist dabei im Vergleich zu konventionellen Unterrichtsfächern aus dem mathematisch-naturwissenschaftlichen Bereich meist wesentlich offener.

Eine strikte Problemorientierung kann den Informatikunterricht auch davor bewahren, in die Niederungen reiner Produktschulung abzufallen (siehe auch Abschnitt 1.1.2), wo oft das konkrete System zum Ausgangspunkt unterrichtlichen Handelns gemacht wird. Typischerweise führt eine solche Werkzeugorientierung, wie sie oft beim „Programmierunterricht“ vergangener Tage zu finden war, bei den Schülern zur Frage: „Jetzt habe ich ein schönes Werkzeug kennen gelernt, was soll ich nun damit anfangen?“

### **3.2.2 Modellbildung und Simulation**

Im Gegensatz zu anderen Konzeptionen (Fakultätentag (1996), Hoppe u. Luther (1996), Rechenberg (1997)) betrachten wir den Prozess von Modellbildung und Simulation nicht als *Lerninhalt*, sondern als durchgängiges *Prinzip* der Unterrichtsgestaltung. Deshalb findet man diese Schlagwortkombination auch nicht unter den später vorgeschlagenen Lerninhalten. Unser Informatikunterricht beschäftigt sich nicht nur mit Modellbildung und Simulation, unser Unterricht besteht im Wesentlichen aus Modellbildung und Simulation, wie wir in Abschnitt 3.3.2 ausführlich darlegen werden. Spezielle, schülergemäße *Modellierungstechniken* gehören dagegen durchaus zu den vorgeschlagenen Lerninhalten.

### 3.3 Organisationsrahmen für den Informatikunterricht

Trotz aller Verschiedenheiten bezüglich der Situation an verschiedenen Schularten und in den einzelnen Bundesländer gibt es einige Rahmenbedingungen, deren Einhaltung für alle Arten von Informatikunterricht anzuraten ist.

#### 3.3.1 Verankerung im Pflichtfachbereich

Unter Berücksichtigung der gegenwärtigen Schulsituation ist die Vermittlung der entscheidenden allgemein bildenden Lerninhalte aus dem Bereich der Informatik nur innerhalb eines eigenen, fest installierten Pflichtfaches möglich. Dafür kann man zumindest drei Begründungen anführen:

**Lehrerausbildung.** Nur im Rahmen eines speziellen Pflichtfaches „Informatik“ kann sichergestellt werden, dass der Unterricht von Lehrkräften mit adäquater Ausbildung durchgeführt wird. Darunter verstehen wir ein für das jeweilige Lehramt zugeschnittenes Universitätsstudium der Informatik mit angemessener Tiefe, wie dies für alle anderen Pflichtfächer seit langem selbstverständlich ist. Die Anforderungen an ein solches Studium wurden u.a. von der Gesellschaft für Informatik ausführlich beschrieben (siehe Gesellschaft für Informatik (1999)).

**Intellektuelle Ansprüche.** Die im folgenden Kapitel 4 vorgeschlagenen Lerninhalte stellen zum Teil hohe Anforderungen an die intellektuelle Leistungsfähigkeit der Schüler. Die Schüler sind zu derartigen Anstrengungen nur in einem Fach bereit und in der Lage, das formal anderen Fächern mit ähnlichen Ansprüchen (etwa Mathematik) gleichgestellt ist.

**Kontinuität.** Die vorgeschlagenen Lerninhalte sind auf keinen Fall innerhalb einer Jahrgangsstufe vermittelbar. Das Zustandekommen von Wahlkursen hängt aber stark von wechselnden Rahmenbedingungen wie Schülerinteresse, Verfügbarkeit von Räumen und Lehrkräften und ähnlichem ab. Ein kontinuierliches Arbeiten über mehrere Schuljahre hinweg ist deshalb nicht möglich.

Davon unberührt bleiben natürlich *zusätzliche* Angebote aus dem Wahlkursbereich, die insbesondere für Schüler mit speziellen Interessen eine willkommene und fruchtbare Ergänzung des Pflichtunterrichts darstellen können. Ein reguläres Pflichtfach kann jedoch durch solche Kurse niemals ersetzt werden.

#### 3.3.2 Zeitliche Grobstruktur

Die erste Begegnung mit neuen Lerninhalten soll nach den obigen Überlegungen zur Problemorientierung möglichst innerhalb von größeren Unterrichtsprojekten stattfinden. Dazwischen müssen Festigungsphasen eingeschoben werden, in denen der während der letzten Projekte erlernte Unterrichtsstoff systematisiert, eingeord-

net, wiederholt, zusammengefasst sowie direkt und apponiert (siehe Teil A, Kapitel 2) geübt werden kann. In diesen Phasen werden auch die Anforderungen für Leistungserhebungen definiert.

Die tatsächlich verfügbare Stundenzahl je Jahrgangsstufe für ein zweistündiges Fach beträgt ca. 50 Unterrichtsstunden. Je Halbjahr sollten etwa 4 Blöcke mit je 2-stündigen Festigungsphasen eingestreut werden. Damit verbleiben für Projektarbeit ca. 34 Stunden pro Schuljahr, also Raum für 3–4 umfangreiche Unterrichtsprojekte.

### 3.3.3 Feinstruktur der Projekte

Modellbildung und Simulation als Unterrichtsprinzip in Verbindung mit den bewährten Stufen des unterrichtlichen Vorgehens (siehe Teil A, Abschnitt 4.5.1) sowie die Berücksichtigung der lernpsychologischen Vorgaben aus dem vorangegangenen Abschnitt 3.1 führen uns zu den folgenden Vorschlägen für die Phaseneinteilung der Unterrichtsprojekte (siehe auch Hubwieser u. Broy (1996), (1997)). Die Ähnlichkeit mit der Struktur gängiger Vorgehensmodelle bei der Softwareentwicklung (siehe etwa Rechenberg u. Pomberger (1997)) ergibt sich zwangsläufig aus dem gemeinsamen Prinzip der Problemorientierung. Diese Struktur ist jedoch keineswegs als strenges Schema gedacht. Wir stellen uns eher vor, dass man versucht, bei jedem Projekt alle genannten Phasen zu streifen, wobei deren Reihenfolge u.U. von den Eigenheiten der Aufgabenstellung abhängen kann.

**Problembegegnung.** Zunächst erfolgt ein erster Kontakt mit einer Problemstellung aus der Praxis, die Notwendigkeit des Einsatzes von neuen Techniken der Informationsverarbeitung wird klar. Ziele dieser Phase sind Motivierung, Veranschaulichung, Wiederholung und Festigung von früher Gelerntem (siehe Teil A, Kapitel 2). Der Lehrer wird hierbei naturgemäß im Mittelpunkt stehen, Lehrervortrag, Schülervortrag und Unterrichtsgespräch dominieren. Als Medien sind originale Begegnungen, Film- und Bildmaterial sowie Simulationen am Rechnernetz denkbar.

**Informelle Problembeschreibung.** Die Schüler versuchen informell, das Problem mit allen seinen Randbedingungen verbal oder graphisch möglichst ausführlich zu beschreiben. Erziehung zu planmäßigem Handeln, Training von Sorgfalt und Umsicht stehen im Vordergrund. Die Schüler arbeiten alleine oder in Gruppen. Als technische Hilfen bieten sich Textverarbeitungs- oder Grafikprogramme an, am besten mit der Möglichkeit der Erstellung gemeinsamer Dokumente über das Schulnetz. In der Softwareentwicklung wäre das Ergebnis der entsprechenden Phase ein Pflichtenheft.

**Formale Modellierung.** Unter Anwendung der im folgenden Kapitel 4 beschriebenen Modellierungstechniken sollen die Schüler

- Techniken zur Strukturierung von Information erlernen,
- zu Sorgfalt, Genauigkeit, systematischem Denken und Handeln erzogen werden,
- im Hinblick auf ein späteres Studium standardisierte Modellierungsverfahren kennenlernen,

- Einblicke in Methoden zum Entwurf komplexer Informationssysteme gewinnen,
- sowie eine Förderung ihrer Abstrahierungsfähigkeit erfahren.

Die Gruppe ist die beherrschende Sozialstruktur, typisch wäre etwa die Entwicklung unterschiedlicher Modellklassen durch einzelne Gruppen mit abschließender gemeinsamer Diskussion der Ergebnisse. Ein willkommenes Hilfsmittel bei der Erarbeitung von Diagrammen wäre ein geeignetes Flow-Chart-Programm, das eine saubere Anordnung der Elemente auf dem Arbeitsblatt, leichte Korrekturen und eine Einbindung in ein Abschlussdokument ermöglicht.

**Implementierung und Realisierung.** Das Ziel dieser Phase ist es vor allem, die in der vorigen Phase konstruierten Modelle zu überprüfen und zu veranschaulichen. Außerdem trägt die Aussicht auf ein lauffähiges System entscheidend zur Motivation der Schüler bei. Die realisierte Lösung ist dabei immer die Simulation eines mehr oder weniger komplexen Systems. Dies kann sich sowohl auf ein real vorhandenes System als auch auf ein hypothetisches System beziehen. Die Vermittlung profunder Kenntnisse über ein spezielles Programm- oder Programmiersystem wird dabei nicht angestrebt. Es kommen verschiedene Informatiksysteme wie Standardsoftware, Programmiersprachen oder spezielle Simulatoren zum Einsatz, die im folgenden Abschnitt 3.4. besprochen werden. Auch hier sollte die Gruppenarbeit dominieren, etwa in der Programmierung einzelner Module oder in der Realisierung derselben Problemstellung mittels unterschiedlicher Standardsoftware.

**Bewertung.** Zur Wiederholung, Festigung, Einordnung und Förderung der Kritik- und Urteilsfähigkeit der Schüler folgt abschließend eine Bewertungsphase, ähnlich der Review-Phase der Softwaretechnik. In allgemeiner Diskussion werden die Beschreibungen und Realisierungen bewertet und mit Alternativen verglichen. Es werden Fragen beantwortet wie:

- Was könnte man verbessern?
- Welche Teilprobleme wurden nicht gelöst?
- Wo haben wir das Problem vereinfacht?
- Welche alternative Realisierungen gäbe es?
- Welche ähnlichen Probleme können wir mit unserer Lösung behandeln?
- Welche Fehlerfälle können auftreten, wie reagiert unser System darauf?
- Welche Konsequenzen hat das System auf sein Umfeld im Hinblick auf Datenschutz, Arbeitsplätze, etc.?

## 3.4 Bemerkungen zu Unterrichtsmethoden

Der Begriff „Unterrichtsmedium“ wird zwar häufig gebraucht, ist jedoch sehr schwer einzugrenzen. Meyer (1987) schreibt dazu:

Was Zweck und was Mittel bzw. Medium des Unterrichts ist, kann also auf unterschiedlichen Ebenen methodischer Reflexion und aus der Perspektive von Lehrern und Schülern je unterschiedlich bestimmt werden. Es gibt keine Medien „an sich“,

deren theoretischer Status unabhängig von realen Unterrichtsprozessen bestimmt werden könnte, sondern lediglich historisch überlieferte, mehr oder weniger gut begründete Hypothesen über den Mediencharakter bestimmter unterrichtlicher Erscheinungsformen. Deshalb muss der Medienbegriff theoretisch unbestimmt bleiben, auch wenn er in der Unterrichtspraxis durch Sprachkonventionen genau fixiert werden kann.

Diese Unbestimmtheit wird im Informatikunterricht besonders deutlich: Einerseits dient die Behandlung grundlegender informatischer Konzepte auch (und nicht zuletzt) dem besseren Verständnis der Arbeitsweise konkreter Informatiksysteme, die damit in gewisser Weise zum Unterrichtsgegenstand werden, andererseits werden solche konkreten Systeme zum Zwecke der Veranschaulichung von abstrakten Konzepten, also als Medium, eingesetzt.

Zur Veranschaulichung der im Unterricht erstellten Modelle sind alle Medien geeignet, die eine Simulation zulassen. Dabei ist stets im Auge zu behalten, ob der Nutzen des Einsatzes den Aufwand zur Einarbeitung in das System rechtfertigt. Den Schülern muss immer bewusst sein, dass das Ziel nicht im Erlernen der Feinheiten einer bestimmten Programmiersprache oder in der perfekten Beherrschung eines Programmsystems liegt, sondern im Erkennen übergeordneter Strukturen oder Strategien.

Nun sollen einige typische Medien (die aber auch zu Unterrichtsinhalten werden können) vorgestellt werden.

### 3.4.1 Bürosoftware

Textverarbeitung, Tabellenkalkulationen oder Datenbanksysteme, letztere insbesondere in relationaler Ausprägung, stellen für einfache Beispiele bereits durchaus geeignete Implementierungsmittel dar. Für anspruchsvollere Probleme könnte man makroprogrammierbare Software verwenden. Ideal dafür wären integrierte Programmepakete („Office“-Programme) mit einer durchgängigen, intuitiv verständlichen Makrosprache.

Im Informatik-Anfangsunterricht gewinnt die Untersuchung der typischen Strukturen kommerzieller Softwaresysteme immer mehr an Bedeutung (siehe Knapp u. Fischer (1998), Füller (1999), Hubwieser (1999a)). Mit ihrem sehr interessanten *Dekonstruktionsansatz* versuchen Hampel, Magenheim u. Schulte (1999), diese Unterrichtsbemühungen zu systematisieren und theoretisch zu hinterlegen, indem sie exemplarisch ein spezielles Softwaresystem analysieren lassen:

Programmierung als Teil des Konstruktionsprozesses eines Informatiksystems steht nicht mehr allein im Mittelpunkt des Informatikunterrichts, sondern ggf. eher am Ende eines Modellierungs-, Formalisierungs-, Abstraktions- und ggf. Dekonstruktionsprozesses. Objektorientierte Modellierung bewegt sich dann zwischen den Ebenen von Systemanalyse (reales Objekt), Formalisierung bzw. Abstraktion von Aufgaben und Kooperationen (abstraktes Objekt) sowie Klassendefinitionen (implementierte Klasse).

### 3.4.2 Hypertextsysteme

Anhand von Hypertextsprachen wie der im Internet sehr verbreiteten *Hypertext Markup Language* (HTML) bieten sich gute Gelegenheiten, die Schüler in Techniken der Informationsstrukturierung und -darbietung einzuführen. In der Regel, vor allem in der Unterstufe, wird man unter Umgehung der Sprachsyntax unter Verwendung von entsprechenden Hilfsprogrammen mit den Schülern Hypertextstrukturen aufzubauen. Ebenso gut können Hypertextsprachen aber auch als Beispiele für formale Sprachen mit einer unmittelbar sichtbaren, „visuellen“ Semantik dienen. Hier kann man die Bedeutung syntaktischer Konstrukte hervorragend veranschaulichen, ohne auf komplexe Zustands- oder Auswertungskonzepte zurückgreifen zu müssen.

### 3.4.3 Programmiersprachen

Die Implementierung der erstellten Modelle zwingt zumindest gelegentlich zur Verwendung einer Programmiersprache. Welche Aspekte dabei zu beachten sind, soll in Kapitel 4 besprochen werden. In jedem Fall stellt sich die Frage, welche der zahlreichen verfügbaren Sprachen man jeweils verwenden soll, weshalb wir kurz einige Vor- und Nachteile der verschiedenen Sprachparadigmen ansprechen wollen. Die Zuordnung einer Sprache zu einem bestimmten Paradigma ist dabei meist keineswegs eindeutig. So haben z.B. die meisten objektorientierten Sprachen einen stark imperativen Charakter. Dennoch hat jede Sprache ihr „Schwerpunktparadigma“. Wir wollen kurz auf einige wenige Vor- und Nachteile der einzelnen Paradigmen eingehen. Eine ausführlichere Besprechung findet sich z.B. bei Schwill (1995).

**Funktionale Sprachen** (z.B. Haskell, Gofer, ML). Diese Sprachen haben meist eine sehr knappe Syntax mit der Möglichkeit einer kompakten Darstellung rekursiver Datenstrukturen. Ein Programm ist einfach nur ein Term, in dem selbst definierte Funktionen vorkommen können. Andererseits kann die Wiederholung von Befehlen (unter Einhaltung des Sprachparadigmas) nur über Rekursion gesteuert werden, wodurch jüngere Schüler überfordert sein könnten.

**Imperative Sprachen** (z.B. Basic, Pascal, C, Modula 2, Oberon). Günstig ist die Analogie zur Funktionsweise des Von-Neumann-Rechners, wodurch ein Grundverständnis für dessen Funktionsweise erzeugt wird. Weniger positiv ist die Verführung zur „Ad-hoc Codierung“ sowie die für Schüler oft schwer durchschaubare Zustandssemantik.

**Objektorientierte Sprachen** (z.B. C++, Java, Smalltalk). Diese Sprachen liegen nach Schwill (1995) und Füller (1999) nahe am menschlichen Weltbild. Schwierigkeiten bereiten dagegen erfahrungsgemäß häufig die höheren Konzepte der Objektorientierung wie dynamische Bindung, Vererbung oder Polymorphie.

**Prädiktative Sprachen** (z.B. Prolog). Für spezielle Anwendungen wie die Modellierung von Wissensbasen sind diese Sprachen sehr geeignet. Die Syntax ist minimal. Problematisch ist das Grundverständnis der Semantik, das zumindest elementare Einblicke in die Prädikatenlogik voraussetzt.

Wie man sieht, gibt es (derzeit noch) kein optimales Sprachparadigma für schulische Zwecke. Man wird also nicht umhin kommen, im Lauf der Schulausbildung mehrere Sprachen für verschiedene Zwecke einzusetzen. Dabei ist allerdings zu berücksichtigen, dass die Schüler sehr viel Zeit zur Einarbeitung in eine neue Sprache benötigen. Ob sich dieses Problem mit Mischformen wie Logo oder Makrosprachen lösen lässt, muss durch entsprechende Unterrichtsversuche gezeigt werden.

### 3.4.4 Programmieroberflächen

Neben dem Paradigma der Sprache ist die verwendete Programmieroberfläche von großer Bedeutung für den Unterrichtseinsatz. Hier reicht die Spannweite vom spartanischen Texteditor über spezielle Programmieroberflächen mit Syntaxprüfung bis zur *visuellen Programmierung*, bei der man nur noch die Attributwerte der Objekte setzt. Bei letzterer spart man sich zwar das Eingeben zahlreicher Syntaxelemente, handelt sich jedoch im Gegenzug eine ziemlich undurchsichtige Semantik ein: Warum passiert wo genau was? Welche Information steckt in welchen Attributen und Dateien? Welcher Programmcode wird warum wann ausgeführt (siehe dazu auch Füller (1999))?

Eine schöne Möglichkeit für einen Einstieg in die Programmierung in der Sekundarstufe I bieten beispielbezogene Programmiersysteme wie *Karel* der Roboter (siehe Patti (1981)). Dabei kann man auf dem Bildschirm einen kleinen Roboter bewegen und zum Auf- und Abbauen von Ziegelsteinbauwerken veranlassen. Dies kann anfangs im Befehlsmodus durch Aufruf entsprechender Kommandos vor sich gehen. Später kann man dann dieselben Befehle zusammen mit grundlegenden Kontrollstrukturen zu einem Programm kombinieren und ablaufen lassen, etwa um ein Haus aufzubauen. Auf der Basis dieses System entwickelten Freiberger und Krsko im Jahre 2001 eine aktualisierte Version für MS-Windows® namens *Karol* (siehe <http://www.schule.bayern.de/karol>), die u.a. Syntaxprüfung, Einzelschrittmodus sowie die Unterstützung von Struktogrammen anbietet.

Für Informatik-Anfänger eignet sich auch das Grafik-Programmiersystem *EOS* („einfache objektorientierte Sprache“), das von Martin Pabst entwickelt wurde (siehe <http://berg.heim.at/anden/420971/eos>). Es ermöglicht den automatisierten Aufbau sowie die Animation von Vektorgrafiken.

### 3.4.5 Code-Generatoren und Simulatoren

Eine weitere interessante Perspektive für den Unterricht liefern Code-Generatorsysteme, die aus formalen Beschreibungen auf sehr abstrakter Ebene lauffähigen Programmcode generieren (siehe Broy et al.(1996)). Damit wird den Schülern unmittelbar die Bedeutung ihrer Modellierungsergebnisse vor Augen

geführt. An den Hochschulen werden derzeit zahlreiche derartige Systeme entwickelt, die den Schulen in absehbarer Zeit kostengünstig überlassen werden könnten. Durch visuelle Simulation des zeitlichen Ablaufs der entwickelten Modelle könnte die didaktische Wirksamkeit noch gesteigert werden. Die Schüler hätten dann die Möglichkeit, unmittelbar nach der Modellierungsphase deren Ergebnisse mit der Anforderungsbeschreibung zu vergleichen.

Ein sehr attraktives Beispiel ist der programmierbare Marienkäfer *Kara* der ETH Zürich (siehe Nievergelt (1999)), der direkt über die Eingabe eines endlichen Automaten gesteuert wird.

Speziell für die Einführung in die objektorientierte Programmierung wurde die Programmieroberfläche *BlueJ* entwickelt, die auf *Java* basiert (siehe <http://www.bluej.org/>). Das System entstand in einem Kooperationsprojekt der University of Kent und der Deakin University (Melbourne). Es bietet die Möglichkeit, aus dem Klassendiagramm heraus interaktiv Objekte anzulegen, die dann auch grafisch angezeigt werden. Mit einem Klick auf diese Objekte kann man ihre Methoden aufrufen und diesen ggf. Parameter übergeben. Auf diese Weise bleibt die ansonsten übliche „main“-Startmethode mit ihrer für Anfänger kryptischen Syntax („public static void main“) verborgen.

## 4 Die Lerninhalte

In diesem Kapitel wollen wir uns mit der Frage befassen, welche Lerninhalte zu einem zeitgemäßen Informatikunterricht gehören. Dabei soll weniger ein fester Katalog möglicher Themen als ein Verfahren zur Entwicklung eines solchen Kata-loges vorgestellt werden. Gerade der Informatikunterricht benötigt auf seinem gegenwärtigen Entwicklungsstand ein solches Verfahren, um sich einerseits im Pflichtfachbereich gegen die starke Konkurrenz anderer Fächer durchsetzen und sich andererseits gegenüber der Medienerziehung abgrenzen zu können.

### 4.1 Wozu Lerninhalte systematisieren?

Gegen die Systematisierung und Katalogisierung von Lerninhalten gibt es zahlreiche, zum Teil sehr berechtigte Vorbehalte. Baumann (1998) schreibt zu diesem Thema:

Es ist stets die Vorstellung der Konzentration auf wenige repräsentative Probleme und die Forderung der Stoffbeschränkung ... dabei. Dies ist ein immer wieder anzustrebendes, aber nie erreichtes Ideal. Durch Systematisierung und Katalogisierung wird es ad absurdum geführt.

Es hilft nichts: Um das steinige Geschäft der Entwicklung von Lernzielen der Informatik, wie es z.B. in den neuen GI-Empfehlungen (Brauer u.a., 1993) geschieht, kommt man nicht herum. Die Stadien dazu lauten wie folgt: (1) Analyse von allgemeinen Zielen und Normen, von Lebenssituationen und den für sie wichtigen Qualifikationen; (2) Analyse von Inhalten und Methoden des Fachs unter den in (1) ermittelten pädagogischen Interessen der Lernenden; (3) Analyse der Unterrichtspraxis, insbesondere der vorhandenen Curricula und deren offene und geheime Ziele; (4) Einbeziehung von Erkenntnissen der Psychologie (Entwicklungs- und Lernpsychologie). ...

Hieraus resultieren die folgenden drei didaktischen Leitlinien:

- Problemlösen mit Informatiksystemen,
- Wirkprinzipien, Struktur und Funktionsweise von Informatiksystemen,
- Grundlagen, Grenzen informatischer Wissensverarbeitung.

Grundsätzlich findet diese Aussage Baumans unsere Zustimmung. Allerdings ist bezüglich des Schulfachs *Informatik* zu bedenken, dass dieses im Gegensatz zu etablierten Fächern in Deutschland keine Tradition in dem Sinn hat, dass es ein großer Teil der Bevölkerung in seiner Schulausbildung besucht hätte. Daraus resultiert das Fehlen eines gesellschaftlichen Konsenses über die Frage, was zur Schulinformatik gehört und was nicht, ja sogar, was unter einzelnen Begriffen der Informatik zu verstehen ist. Dem ist mit der Präsentation von Leitlinien, Leitfra-

gen oder Problemkreisen alleine – so nützlich diese in anderer Hinsicht auch sein mögen – nicht beizukommen. Die Schulinformatik benötigt zusätzlich folgende Komponenten:

- eine kohärente Grundmenge potentieller Lerninhalte,
- ein plausibles *schulspezifisches* Verfahren zur Auswahl der tatsächlich zu lehrenden Inhalte,
- ein exemplarisches, einigermaßen schlüssiges Umsetzungskonzept für mindestens eine spezielle Schulart,
- möglichst viele konkrete Unterrichtsbeispiele, um die angestrebte Behandlungstiefe der im Katalog genannten Themen abzustecken.

## 4.2 Informationszentrierung

Zur Entwicklung eines Themenkataloges für die Schulinformatik benötigen wir zunächst *einen* Ausgangspunkt für unsere Überlegungen. Die in Kapitel 1 dargestellten bisherigen didaktischen Ansätze gingen von den Begriffen „Rechenanlage“, „Algorithmus“, „Anwendung“ oder „Benutzer“ aus, die den heutigen Ansprüchen eines allgemein bildenden Faches (siehe Abschnitt 2.2) nicht mehr genügen können, da die daraus resultierenden Lerninhalte nicht die nötige Breite bzw. Tiefe aufweisen. Breier (1994) fordert deshalb eine Umorientierung:

In einem zeitgemäßen Informatikunterricht steht meines Erachtens nicht der Algorithmus, sondern die Information als Erscheinungsform der realen Welt im Mittelpunkt.

Auch wir (siehe Hubwieser u. Broy (1996)) sehen den Begriff „Information“ als den zentralen Begriff des Informatikunterrichts an. Deshalb bezeichnen wir unsere Vorschläge auch als *informationszentrierten Ansatz*.

Bevor wir uns auf die Suche nach „informationszentrierten“ Lerninhalten machen können, müssen wir uns fragen, was wir eigentlich unter „Information“ zu verstehen haben.

### 4.2.1 Der Informationsbegriff

Der Begriff „Information“ wird in der deutschen Sprache auf vielschichtige Weise und in zahlreichen Bedeutungen verwendet. Es würde zu weit führen, an dieser Stelle einen Überblick geben zu wollen. Deshalb werde ich mich punktuell auf einige wenige Interpretationen beschränken, die im Zusammenhang mit der Schulinformatik eine größere Rolle spielen.

Shannon (siehe Shannon u. Weaver (1949)) stellte im Sinne einer „Entscheidungsinformation“ ein probabilistisch fundiertes Maß für den mittleren Entscheidungs- (oder Informations-) gehalt eines Zeichens innerhalb einer Nachricht auf. So bedeutend diese Arbeit auch für die Theorie der Nachrichtentechnik sein mag, so fernab liegt dieser Begriff von unserem alltäglichen Verständnis von „Information“, worauf bereits Bauer u. Goos (1982) hingewiesen haben.

Diese Alltagsbedeutung des Wortes erklärt z.B. das Duden-Fremdwörterbuch (siehe Duden Fremdwörterbuch (1997)) mit:

1. Nachricht; Auskunft; Belehrung; Aufklärung.
2. als räumliche oder zeitliche Folge physikalischer Signale ... sich zusammensetzende Mitteilung, die beim Empfänger ein bestimmtes [Denk]verhalten bewirkt.

Dies würde man aus heutiger informatischer Sicht eher als *Darstellung* von Information bezeichnen.

Wesentlich brauchbarer für unsere Zwecke sind Definitionen, die den Begriff auf die *Bedeutung* einer Nachricht anstatt auf die Nachricht selbst beziehen. So liest man in Broy (1997):

Information nennt man den abstrakten Gehalt („Bedeutungsinhalt“, „Semantik“) einer Aussage, Beschreibung, Anweisung, Nachricht oder Mitteilung. Die äußere Form der Darstellung nennt man Repräsentation (konkrete Form der Nachricht).

Im Folgenden werde ich „Information“ in diesem Sinne verwenden. Der Duden Informatik (siehe Duden Informatik (1993)) verwendet dagegen einen weiter gefassten Informationsbegriff, indem er festlegt:

Information umfasst eine Nachricht *zusammen* mit ihrer Bedeutung.

Wir wollen jedoch streng zwischen einer Information (Bedeutung) und ihrer Repräsentation (Nachricht) unterscheiden, wenn auch die eine ohne die andere nicht existieren kann.

#### **4.2.2 Das Paradigma der Informationsverarbeitung**

Nun könnte man auf die Idee kommen, alles was mit Nachrichten (im weitesten Sinne) und deren Bedeutungsgehalt zu tun hat zum Gegenstand der Schulinformatik zu machen. Damit würde man weit in die Domänen anderer Wissenschaften wie der Philosophie, Rechtswissenschaft, Psychologie oder Biologie vorstoßen. Um solche Verirrungen von vornherein auszuschließen, benötigen wir eine weitere Eingrenzung der möglichen Themenbereiche.

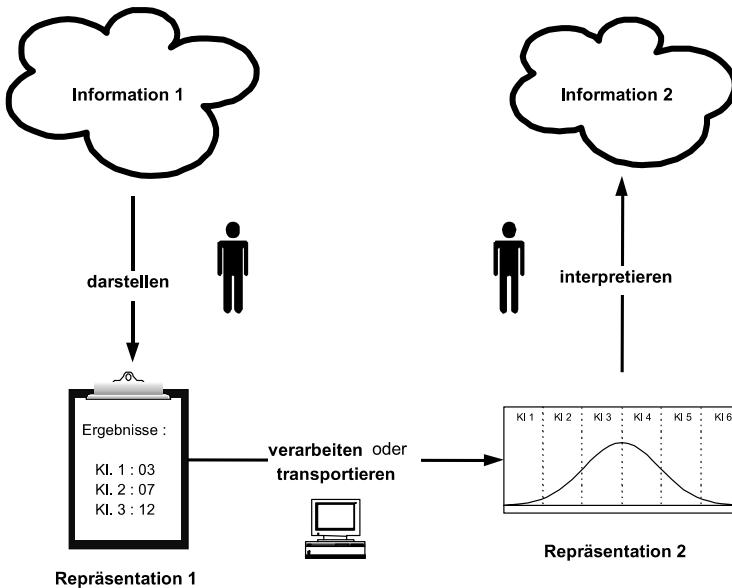
Eine sehr elegante Möglichkeit dazu bietet die Einschränkung auf den Bereich der *Informationsverarbeitung*. Dazu betrachten wir das Grundschema jeder künstlichen (d.h. nicht natürlichen) Informationsverarbeitung, das im Wesentlichen durch Erweiterung aus dem mittlerweile etwas bejahrten EVA-Prinzips (für Eingabe, Verarbeitung, Ausgabe, siehe Duden Informatik (1993)) hervorgeht:

1. Vor jeglicher Verarbeitung müssen Informationen mit Hilfe geeigneter Darstellungstechniken *repräsentiert* werden. Das kann z.B. akustisch, grafisch oder textuell geschehen.
2. Auf solchen Repräsentationen operieren dann automatische *Verarbeitungs- oder Transportprozesse*, um daraus neue Darstellungen zu erzeugen. Der Ablauf dieser Prozesse wird von anderen Informationsrepräsentationen (Pro-

grammen) gesteuert. Die Ergebnisse der Verarbeitung können wiederum als Eingabe für weitere Verarbeitungsprozesse dienen.

3. Durch *Interpretation* entstehen aus den Repräsentationen schließlich neue Informationen: Der vom Verarbeitungs- oder Transportsystem ausgegebenen Repräsentation wird von Menschen eine Bedeutung zugeordnet.

In Abb. 4.1 ist dieses Schema am Beispiel einer Testauswertung ausgeführt: Die Ergebnisse einer Testsequenz werden in Form einer Tabelle dargestellt (Repräsentation), aus der anschließend mit Hilfe geeigneter Software automatisch ein Diagramm für die gewünschte Verteilungskurve generiert wird (Verarbeitung). In einem dritten Schritt interpretiert ein menschlicher Betrachter dieses Diagramm, wobei er z.B. zu dem Schluss kommen könnte, dass die Testergebnisse annähernd einer Normalverteilung folgen.



**Abb. 4.1.** Grundschema der Informationsverarbeitung

Ein Vergleich mit unserer Definition eines Informatiksystems aus Abschnitt 1.1 zeigt, dass wir auch in diesem Schema die dort aufgeführten drei Kriterien wieder finden:

- automatische Verarbeitung,
- Vernetzung (Transport von Repräsentationen, räumliche Verteilung von Verarbeitung oder Repräsentationen) und
- Interaktion (Repräsentation und Interpretation).

Dieses Grundschema ist daher geeignet, die prinzipielle Wirkungsweise von Informatiksystemen auf einer sehr abstrakten Ebene zu beschreiben.

### 4.2.3 Die Grundmenge informatischer Lerninhalte

Auf der Suche nach einer Grundmenge möglicher Lerninhalte untersuchen wir nun die einzelnen Phasen in diesem Grundschema sowie alle Wechselwirkungen der Informatiksysteme mit ihrer unmittelbaren Umgebung. Wir beschränken uns dabei auf *Artefakte* im weitesten Sinne, also auf Vorgänge, bei denen die Informationsverarbeitung gezielt von Menschen gesteuert oder beeinflusst wird. Dies deckt sich in der heutigen Zeit weitgehend mit dem Bereich der elektronischen Informationsverarbeitung, schließt jedoch Bereiche wie die natürliche Verarbeitung von Erbinformationen aus, zumindest soweit von menschlicher Seite kein (technischer) Einfluss darauf ausgeübt wird.

Dadurch erhalten wir in etwa den in Tabelle 4.1 gezeigten inhaltlichen Rahmen, wobei unsere Intention in diesem Stadium in einer reinen *Auflistung* potentieller Lerninhalte ohne Anspruch auf Vollständigkeit oder strenge Kategorisierung liegt. Zahlreiche Stichwörter gehören offensichtlich zu mehr als einem der genannten Themenbereiche, wie etwa „Programme“ oder „Verschlüsselung“.

**Tabelle 4.1.** Inhaltlicher Rahmen für die Schulinformatik

---

Darstellung von Information	Repräsentationen, auf denen Verarbeitungsprozesse operieren (Daten): Datentypen und -strukturen, Trägermedien, Darstellungsregeln (Syntax) Repräsentation von Informationen über den Ablauf von Verarbeitungsprozessen (Verarbeitungsvorschriften): Programme und Programmiersprachen Modelle von Informatiksystemen (Systembeschreibungen): zeitliche Abläufe, Dekomposition in Subsysteme, Kommunikation mit der Außenwelt und zwischen den Subsystemen
Verarbeitung und Transport von Repräsentationen	Einsatz- und Anwendungsmöglichkeiten von Systemen zur automatischen Verarbeitung von Information zeitliche und räumliche Struktur von Informatiksystemen: Ablauf (Parallelität), Komponenten, Verteilung, Kooperation Wechselwirkungen von Informatiksystemen mit ihrer Umgebung in zeitlichem, räumlichem, menschlichem und gesellschaftlichem Kontext: Geschichte, Entwicklung, Betrieb, Bedienung, Ergonomie, Auswirkungen auf die Arbeits- und Berufswelt
Interpretation von Repräsentationen	Interpretationsregeln: Semantik und Pragmatik von Sprachen Schutz vor unerlaubten oder unerwünschten Interpretationen: Datenschutz, Zugriffsrechte, Verschlüsselung Möglichkeit von Fehlinterpretationen: Manipulationsmöglichkeiten, Darstellungsfehler

---

#### 4.2.4 Vergleich mit anderen Ansätzen

Im Vergleich zu den oben genannten früheren didaktischen Ansätzen (siehe auch Abschnitt 1.2) haben wir eine Ebene weiter abstrahiert und damit gewissermaßen alle diese Ansätze unter einem gemeinsamen Dach vereinigt.

Auch die von Friedrich (1995) aufgezählten Leitlinien

- Umgang mit Information,
- Wirkprinzipien von Informatiksystemen,
- Problemlösen mit Informatiksystemen,
- Arbeiten mit Modellen,

die inzwischen bundesweit große Anerkennung fanden (siehe Rahmenplan des Kultusministeriums von Mecklenburg-Vorpommern (1998)), werden durch unseren Ansatz abgedeckt, zumindest in dem vom Urheber intendierten Bedeutungsumfang, nämlich auf Informatiksysteme bezogen.

Baumann (1998) zählt vier seiner Ansicht nach für die Informatik zentrale Kategorien auf: „Information“, „System“, „Modell“ und „Programm“. Aus der Sicht unseres Ansatzes erscheint deren Gleichrangigkeit zweifelhaft: Die Begriffe „Information“ und „System“ können ohne Zweifel als grundlegend betrachtet werden. Wir beschränken uns dabei jedoch auf *Informatiksysteme*, die wir mit dem obigen Schema (siehe Abbildung 4.1) charakterisieren. Die Betrachtung von Systemen im Allgemeinen führt zumindest für die Schulinformatik zu weit. Ein *Modell* besteht dagegen nach unserer Auffassung aus Repräsentationen von Informationen über ein System (siehe Abschnitt 4.4.1 unten). Der Begriff kann daher aus „Information“ (damit ist auch „Repräsentation“ definiert, siehe Abschnitt 4.2.1) und „System“ abgeleitet werden. Ebenso verhält es sich mit „Programm“: Damit wird eine Repräsentation von Informationen über einen Verarbeitungsvorgang bezeichnet, also über den Ablauf eines Informatiksystems. Ein *Programm* ist damit ein Spezialfall eines *Modells*. Damit ist auch gezeigt, dass unser Ansatz inhaltlich alle vier von Baumann genannten Kategorien umfasst.

### 4.3 Didaktische Auswahlkriterien für Lerninhalte

Nun benötigen wir geeignete Selektionskriterien, die uns von der im vorigen Abschnitt umrissenen Grundmenge von Lerninhalten zu einem schülergerechten und unterrichtlich umsetzbaren Themenkatalog führen. Dazu verwenden wir die von Bruner (1960) aus allgemeindidaktischen Überlegungen heraus entwickelten vier Kriterien zur Identifizierung von so genannten *fundamentalen Ideen* eines Fachgebietes:

- *Horizontal Criterion*: breite Anwendbarkeit und Beobachtbarkeit,
- *Vertical Criterion*: Vermittelbarkeit in einem breiten Altersbereich,
- *Criterion of Time*: Beobachtbarkeit in der historischen Entwicklung und langfristiger Relevanz,

- *Criterion of Sense*: eine gewisse Bedeutung für Sprache und Denken des täglichen Lebens.

Unter Verwendung dieser Kriterien identifizierte Andreas Schwill (siehe Duden Informatik (1993), Schwill (1996)) drei fundamentale Ideen der Informatik, nämlich

- Algorithmisierung,
- Sprache und
- strukturierte Zerlegung,

die aus der Sicht unseres Ansatzes ungefähr den Bereichen *Verarbeitung*, *Darstellung* und *Verteilung* von Information entsprechen.

Für unsere Zwecke wandeln wir die genannten vier Kriterien etwas ab: Das vertikale Kriterium kann nicht für alle Lerninhalte gefordert werden, da wir uns nicht nur auf fundamentale Ideen beschränken wollen. Auf der Vermittelbarkeit wenigstens auf *einer* Jahrgangsstufe muss man natürlich bestehen. Das horizontale Kriterium wiederum kann zusammen mit dem letztgenannten „Criterion of Sense“ zur Forderung nach größtmöglicher Allgemeingültigkeit verschmolzen werden. Zusätzlich fordern wir die Beachtung des exemplarischen Prinzips. Aus unserer Sicht bleiben damit vier Forderungen, die nun eingehender untersucht werden sollen.

### 4.3.1 Allgemeine Bedeutung

Das Gelernte soll einen möglichst breiten Anwendungsbereich haben. Diese Breite lässt sich in Bezug auf Informatiksysteme in vier Klassen unterteilen (siehe Tabelle 4.2 und Hubwieser u. Broy (1997a)).

**Tabelle 4.2.** Klassifizierung von Lerninhalten nach Allgemeingültigkeit

Klasse	Beschreibung	Beispiele
1	Anwendung auch außerhalb des Bereiches der EDV möglich	Modellierungstechniken, Problemlösungsstrategien,
2	charakteristisch für alle elektronischen Informatiksysteme	Komplexitätsklassen, Grenzen der Berechenbarkeit
3	Anwendung beschränkt sich auf eine Klasse von Informatiksystemen	Programmierparadigmen, spezielle Datenstrukturen, Netzwerktopologien, Sortieralgorithmen
4	betrifft nur ein konkretes System	Syntax einer Programmiersprache, Menüstruktur eines Anwendersystems, Architektur eines Mikroprozessors

Lerninhalte aus den ersten beiden Klassen sind für die Schulinformatik uneingeschränkt geeignet, Themen aus Klasse 3 nur, wenn eine sehr umfangreiche oder sehr wichtige Klasse von Subsystemen betroffen ist. Inhalte aus Klasse 4 sind lediglich soweit zu vermitteln, als sie für die Realisierung, Implementierung oder Simulation von Konzepten höherer Klassen unbedingt nötig sind. Letztere sollten jedoch keinesfalls zur Bewertung von Schülerleistungen herangezogen werden.

### **4.3.2 Lebensdauer**

Natürlich dürfen im Unterricht nur Kenntnisse und Fertigkeiten vermittelt werden, denen auch dann noch eine ausreichende Bedeutung zukommen wird, wenn die Schüler ins Berufsleben eintreten werden. Wie wir bereits in Abschnitt 2.2 ausgeführt haben, kann das eine Zeitspanne von bis zu 13 Jahren umfassen.

Die Lebensdauer der Lerninhalte korreliert im Wesentlichen mit ihrer Allgemeinheitsklasse (siehe Tabelle 4.2). Allerdings gibt es innerhalb der einzelnen Klassen immer noch erhebliche Unterschiede. Daher stellt die Lebensdauer ein willkommenes Kriterium zur Differenzierung der Lerninhalte innerhalb der gleichen Klasse dar, welches besonders in Klasse 3 sehr nützlich sein kann: Es kann durchaus sinnvoll sein, die Datenstruktur einer zeitlich stabilen Klasse kommerzieller Software wie Textverarbeitungssysteme zu behandeln, während die Funktionsweise analoger Modems in zehn Jahren vielleicht nicht mehr interessant sein wird.

Ausgeschlossen bleiben aufgrund dieses Kriteriums auf jeden Fall Themen und Techniken, wie sie in den zahlreichen Computerzeitschriften unter der Rubrik „Neuigkeiten“ besprochen werden sowie reine Bedienerkenntnisse, die auf ein bestimmtes Produkt zugeschnitten sind.

### **4.3.3 Vermittelbarkeit**

Der interessanteste Lernstoff hat in der Schule natürlich nichts verloren, wenn eine altersgemäße Vermittlung nicht möglich ist. Demzufolge werden bestimmte Themen ganz ausgeklammert werden müssen, andere nur eingeschränkt zu behandeln sein. So wird etwa die Behandlung von Kellerautomaten die Fähigkeiten der meisten Schüler der Sekundarstufe I überfordern, während sie einfache endliche Automaten in Form von Zustandsübergangsdiagrammen bereits verstehen und anwenden können.

### **4.3.4 Exemplarische Auswahl und Einflechtung**

Sowohl die Göttinger Schule (siehe Teil A, Abschnitt 3.1) als auch die Berliner Didaktik (siehe Teil A, Abschnitt 3.2) betonen die Bedeutung *des Exemplarischen Prinzips* für die Auswahl von Lerninhalten (siehe auch Teil A, Abschnitt 4.2). Auch Baumann (1998) unterstreicht die Notwendigkeit einer Beschränkung auf einige wenige repräsentative Probleme (siehe Zitat in Abschnitt 4.1). In der Tat

muss bei der Zusammenstellung eines Lehrplans immer darauf geachtet werden, ob sich nicht für eine ganze Klasse von Konzepten ein geeigneter repräsentativer Vertreter finden lässt. So liegt es nahe, sich beispielsweise bei der Behandlung der Datenstrukturen von Standardsoftware auf ein oder zwei wichtige Typen zu beschränken oder die wesentlichen Konzepte der Rechnerkommunikation anhand eines einzigen Netzwerkprotokolls zu behandeln.

Als weitere Möglichkeit zur Reduktion der Stofffülle bietet sich die *Einflechtung* an, bei der versucht wird, gewisse Lerninhalte gleichsam nebenbei, im Zuge der Beschäftigung mit anderen Themen, zu vermitteln. Für einige Konzepte der Objektorientierung, die Funktionsweise von Rechenanlagen, Aspekte des Datenschutzes oder gesellschaftliche Auswirkungen der Informatik beispielsweise scheint es angemessen, sie immer wieder an verschiedenen Stellen anhand des gerade betrachteten Systems anzusprechen, anstatt sie allein in den Mittelpunkt des Unterrichts zu stellen.

## 4.4 Modellierung als inhaltlicher Kern

Welche Lerninhalte schließlich Eingang in die Lehrpläne finden, hängt natürlich stark von der jeweiligen Schulart ab. Dennoch gibt es einen Themenbereich der Informatik, der aufgrund seiner immensen Bedeutung für die Allgemeinbildung (siehe Hubwieser u. Broy (1996), (1997), Hubwieser, Broy, Brauer (1996)), in keinem Informatikunterricht übergegangen werden kann. Es handelt sich um den Bereich der *Modellierung*, der hier etwas eingehender erläutert werden soll (siehe auch Hubwieser (1999)).

Die Wichtigkeit des Modellierungsvorganges wird von Seiten der Informatikdidaktiker immer wieder betont, erstmals in dieser Deutlichkeit wohl von Brauer u. Brauer (1973) (siehe Zitat in Abschnitt 1.2.1). Weitere Aussagen ähnlicher Art finden sich bei Koerber u. Peters (1988) oder in den GI-Empfehlungen von 1993 (siehe Schulz-Zander et al. (1993)), wo dieses Thema nicht von ungefähr als erster Lerninhalt aufgeführt wird:

Die Schülerinnen und Schüler sollen:

A1 Exemplarische Methoden und Verfahren der Modellierung eines Ausschnitts der Wirklichkeit kennen lernen, diese anwenden und kritisch hinterfragen, z.B. im Hinblick auf die Grenzen dieser Methoden und auf das Interesse am Einsatz.

Auch die vierte Leitlinie der informatischen Bildung beschäftigt sich mit Modellierung (siehe Friedrich (1995) und Abschnitt 4.2.4).

Leider verfügte man bis vor kurzem nicht über geeignete Techniken, um diesen Modellierungsvorgang im Unterricht systematisch und in angemessener Tiefe umsetzen zu können. Aus diesem Mangel heraus gerieten die Betrachtungen zu diesem Thema im Unterrichtsgeschehen oft zu rein philosophischen, wenig schülergemäßen Exkursen. Inzwischen haben sich jedoch auf dem Gebiet der Softwareentwicklung Modellierungstechniken durchgesetzt, die aufgrund ihrer Anschaulichkeit und Beschreibungsmächtigkeit geeignet scheinen, genau diese methodische Lücke zu schließen. Die Softwaretechnik verwendet inzwischen vor allem objektorientierte

Entwurfsmethoden, die auf diesen Techniken aufsetzen. Dazu gehören die Entwicklungsmethoden von Rumbaugh et al. (1991), Booch (1994) und Jacobson et al. (1991), die inzwischen unter Beteiligung der drei Erfinder zur *Unified Modeling Technique* (UML) verschmolzen und weiterentwickelt wurden (siehe Booch, Rumbaugh, Jacobson (1997)). Einige dieser relativ neuen Techniken ermöglichen eine durchaus altersgemäße Modellierung einfacher Sachverhalte und damit eine direkte Umsetzung unseres didaktischen Ansatzes. Wie bereits in Abschnitt 3.2.2 dargelegt, kann Modellierung im Unterricht dabei sowohl *Lerninhalt* (Erlernen von Modellierungstechniken zur Beschreibung komplexer Systeme) als auch *Methode* (Erarbeitung der grundlegenden Prinzipien von Informatiksystemen durch ihre Modellierung) sein.

#### 4.4.1 Begriffsklärung

Der Begriff „Modell“ taucht in der Informatik auf vielen verschiedenen Ebenen und in zahlreichen Facetten auf. Darunter versteht man unter anderem jegliche genauere Beschreibung von Vorgängen, die Beschreibung von Problemen mit Hilfe von speziellen Modellierungsprogrammen (z.B. Stella), irgendeine graphische Darstellung aus dem betreffenden Problemkreis, eine mathematische Gleichung und vieles andere mehr.

Aus mathematischer Sicht wird unter einem Modell meist Folgendes verstanden (siehe Büttemeyer (1995)):

Eine Interpretation eines formalen Systems  $S$  heißt genau dann Modell von  $S$ , wenn die Axiome von  $S$  dabei wahren Aussagen des betreffenden Gegenstandsbereichs zuordnet sind.

Näher an einer für unsere Zwecke brauchbaren Definition liegt die Ansicht von Baumann (1996):

Modell = vereinfachte struktur- und verhaltenstreue Beschreibung eines realen Systems

Entgegen der darin enthaltenen unnötigen Beschränkung auf reale Systeme sollen Schüler allerdings u.U. auch hypothetische (geplante) Systeme modellieren, ohne deren vollständige Realisierung ernsthaft in Erwägung ziehen zu müssen. (Oft wird man sich mit der Simulation eines Teilbereichs zufrieden geben.) Wir wollen im Folgenden ein *Modell* deshalb als eine abstrahierte Beschreibung eines realen oder geplanten Systems verstehen, das die für eine bestimmte Zielsetzung wesentlichen Eigenschaften des Systems erhält. Die Erstellung einer solchen Beschreibung wollen wir *Modellbildung* oder *Modellierung* nennen.

Den Begriff *System* interpretieren wir wie Wedekind et al. (1998):

Als System im „weiteren Sinne“ ... gilt dabei

- (a) eine Menge von Elementen (Systembestandteilen), die
- (b) durch bestimmte Ordnungsbeziehungen miteinander verbunden und
- (c) durch klar definierte Grenzen von ihrer Umwelt geschieden sind.

Die Identifikation oder Festlegung der im letzten Punkt genannten Grenzen kann bereits eine der wesentlichen (und schwierigsten) Aufgaben des Modellierungsvorgangs darstellen.

Von diesem Standpunkt aus gesehen stellen *Datenstrukturen*, *Verarbeitungsvorschriften* und *Systembeschreibungen* Modelle oder Teile von Modellen dar, allerdings auf unterschiedlichen Abstraktionsstufen. Während Datenstrukturen und Verarbeitungsvorschriften auf die Implementierungstechnik bezogen sind, beschäftigen sich Systembeschreibungen mit den Eigenschaften des zu modellierenden Systems. Konsequenterweise eignen sich letztere auch eher zur Beschreibung von Systemen außerhalb der elektronischen Datenverarbeitung. Deshalb kann man sie oft der Allgemeinheitsklasse 1 (siehe Tabelle 4.2) zuordnen, während Datenstrukturen und Verarbeitungsvorschriften meist eher der Klasse 2 oder 3 angehören.

#### 4.4.2 Programmierung und Modellierung

Wenn also auch ein Programm im Sinne einer Darstellung von Verarbeitungsvorschriften als Modell betrachtet werden kann, dann müssen wir uns an dieser Stelle der Frage stellen, inwiefern die Entwicklung konkreter, lauffähiger Programme zur Allgemeinbildung beitragen kann. Zu diesem Thema können wir auf eine lange und bei oberflächlicher Betrachtung oft sehr kontroverse Diskussion zurückblicken. Rechenberg (1994) argumentiert etwa gegen Programmierung:

Dass alle Schüler das Programmieren erlernen müssten, um Computer später auszunutzen zu können, stimmt jedenfalls nicht. Schon heute sind die weitaus meisten Computeranwendungen im Privatbereich schlüsselfertig gelieferte Programmpakete, und das wird in Zukunft verstärkt so sein. Programmieren ist und bleibt deshalb Spezialistenarbeit.

Ähnlich behauptet Burkert (1995):

Die Konstruktion von Software – im Großen wie im Kleinen – gehört zum ingenieurwissenschaftlichen Teil der Informatik und ist nicht allgemeinbildend.

Auch Meier (1990) spricht der Programmierung zumindest in Bezug auf die Informatik einen tieferen Bildungsgehalt ab:

Programmierunterricht ist auf ein absolutes Minimum zu reduzieren. Programmieren hat im Fach Mathematik eine Berechtigung, wenn es darum geht, Verständnis für das Denken in Algorithmen bewusst zu fördern. Programmierunterricht fördert aber nur in einem sehr unbedeutenden Maße das Verständnis für Probleme der Informationsverarbeitung.

Im Gegensatz dazu halten Hoppe u. Luther (1996) Programmierung für essentiell wichtig:

Ebenso wie das elementare Rechnen die „Primärerfahrung“ der Mathematik ist, gilt dies entsprechend für das Programmieren als Primärerfahrung der Informatik. Die zentralen Begriffe der Informatik ... erwachsen aus den Erfordernissen des Program-

mierens. ... Programmieren ist Ausgangspunkt und Endpunkt gedanklicher Abstraktionsprozesse der Informatik. ... Zusammenfassend stellen wir fest, dass die Erfahrung des Programmierens eine Schlüsselrolle für das Verständnis informatischer Grundbegriffe spielt.

J. Nievergelt (1993) ist derselben Meinung:

Das „Programmieren im Kleinen“ eignet sich als Einstieg in die Informatik ganz besonders, wenn man Perfektion bis ins Detail anstrebt. Hier stehen algorithmische Aspekte im Vordergrund: Entwurf eines Algorithmus, Korrektheitsbeweis, zu verwendende Datenstrukturen, Analyse des Zeit- und Speicherbedarfs. Die Programme sind kurz – oft weniger als eine Seite lang –, können aber nur über einen mathematischen Gedankengang verstanden werden. Dadurch knüpft das Programmieren im Kleinen auf natürliche Weise an die Mathematik an, was im Mittelschulunterricht eine gegenseitige Befruchtung ermöglicht.

Bei näherer Betrachtung haben solche Differenzen oft ihre Ursache in einer unterschiedlichen Interpretation des Begriffs „Programmierung“, wie Sigrid Schubert (1991) treffend feststellt:

Informatikunterricht soll kein Programmierkurs sein. Warum eigentlich nicht? ... Problemlösen (Modellieren und Strukturieren) unter Anwendung von Informatikprinzipien und -methoden gilt als erstrebenswert. Die Programmiersprache soll im Hintergrund (Mittel zum Zweck) bleiben. Das aber ist Programmierung (nicht zu verwechseln mit Codierung).

Im selben Artikel warnt sie allerdings ebenfalls vor einem „Programmierkurs“:

Tatsächlich kann mit dem Schwerpunkt auf der Programmiersprache ein Kurs entstehen, der bestenfalls vorgezogene Berufsbildung darstellt und die allgemein bildenden Ziele verfehlt.

Um derartige Missverständnisse zu vermeiden, müssen wir klarstellen, was wir unter „Programmierung“ verstehen wollen. In Duden Informatik (1993) findet man dazu:

Unter Programmieren versteht man zum einen den Vorgang der Programmerstellung und zum anderen das Teilgebiet der Informatik, das die Methoden und Denkweisen beim Entwickeln von Programmen umfasst.

Im letzteren, weiter gefassten Sinne halte ich mit Schubert (1991) Programmierung für einen wesentlichen, wenn auch nicht für den wichtigsten Bestandteil einer allgemein bildenden Schulinformatik, allerdings nur, solange zwei Bedingungen erfüllt sind:

- Es muss sich (im Sinne von Duden Informatik (1993)) wirklich um die *Implementierung* eines vorher entwickelten Modells handeln, also keinesfalls um Stegreifprogrammierung nach dem Prinzip „Versuch und Irrtum“.
- Die *Syntax* der verwendeten Sprache darf nicht in den Vordergrund treten. Die Schüler dürfen also keinesfalls durch die Eigenheiten der jeweiligen Sprache von der Problematik der Modellierung abgelenkt werden. Ein „Pro-

grammierkurs“, der von Sprach- anstatt von Problemstrukturen ausgeht, ist im Pflichtfachbereich fehl am Platze.

Unter Einhaltung dieser Regeln kann die Programmierung als *Implementierung von abstrakten Modellen* dafür sorgen, dass diese durch Simulation veranschaulicht und überprüft werden können, was vielen Schülerinnen und Schülern erst den eigentlichen Zugang zur Modellierung ermöglichen wird.

Darüber hinaus gibt es aber auch noch weitere informatische Lerninhalte, die über die Programmierung vermittelt werden können. Ich stimme Rechenberg (1997) zu, wenn er, trotz seiner oben erwähnten Ablehnung von Programmierung im Unterricht, als „zentrales Gedankengut der Informatik“ u.a. aufzählt:

Algorithmisches Denken. Das algorithmische Denken scheint mir das Wichtigste zu sein, was die Informatik vermitteln kann. ... Dazu braucht man jedoch keine Kenntnis von Programmiersprachen oder Datentypen oder gar Rekursion. Auf die Denkmethode kommt es an, nicht auf ihre technische Verwirklichung in der gegenwärtigen Informatik.

...

Das Prinzip des von-Neumann-Computers. Eine elementare Darstellung der Bestandteile des von-Neumann-Computers und die Erläuterung des zweischrittigen Befehlszyklus (Befehl holen und Befehl ausführen) sollte nicht fehlen. Sie ist nicht zu schwierig, und sie erklärt, was denn eigentlich in einem Computer vor sich geht. Zumindest aber vermittelt sie die Einsicht, dass es keine Hexerei ist, und entmystifiziert damit den Computer. Hier kann der Lehrer den Schülern klar machen, dass der Computer nur eine Maschine ist, wie andere Maschinen auch; zwar kompliziert, aber doch nichts als eine Maschine.

...

Der Variablenbegriff imperativer Programmiersprachen. Natürlich kommt man ohne eine Programmiersprache nicht aus. Aber mehrmals habe ich feststellen müssen, dass Schüler angeblich eine ganze Programmiersprache gelernt hatten und die einfache Aufgabe, die Werte zweier Variablen miteinander zu vertauschen, nicht lösen konnten. Es fehlte ihnen die begriffliche Klarheit darüber, dass Variablen Behälter sind und wie man diese Behälter füllen und umfüllen kann. Das ist erschütternd: Man lehrt Rekursion, aber der Schüler kann die Werte zweier Variablen nicht miteinander vertauschen! Zu diesem Kapitel gehören noch andere Dinge, zum Beispiel die Unterscheidung zwischen der Gleichheitsrelation und der Zuweisungsoperation.

Ganz ohne Programmierung geht es also offensichtlich nicht. Zumindest an einem Einblick in die imperative Programmierung kommt man (unabhängig von sonstigen nützlichen Seiteneffekten der Programmierung wie Veranschaulichung, Motivierung, Handlungsorientierung etc.) schon wegen der Notwendigkeit der Vermittlung der drei hier von Rechenberg genannten Konzepte nicht vorbei. Dann erhebt sich aber die Frage, welche Lerninhalte man nun in einen realen Lehrplan aufnehmen könnte, um

- sicherzustellen, dass im Unterricht wirklich genügend Gewicht auf die *Modellierung* und *Strukturierung* der behandelten Probleme (in Abgrenzung zur *Kodierung*) gelegt wird,

- zu verhindern, dass sich die Behandlung der Modellierung lediglich in kurzen, abstrakten Exkursen „über die Köpfe der Schüler hinweg“ erschöpft,
- zu vermeiden, dass am Ende doch wieder spezielle Implementierungs-techniken (wie z.B. Zeiger) oder spezifische Eigenheiten der verwende-ten Programmiersprache (wie die Formatierung der Ausgabe) in den Mit-telpunkt des Unterrichts rücken?

Als Lösung bietet sich an, in Lehrplänen die Verwendung von *Modellierungstechniken* vorzuschreiben, wie sie im letzten Absatz besprochen wurden. Damit erzeugt man eine *Hilfsebene* zwischen der Problem- und der Implementierungsebene (siehe auch Körber u. Peters (1988)), die es erlaubt, im Lehrplan alle wesentlichen Anforderungen an den Unterricht zu formulieren, ohne auf programmiertechnische Details wie spezielle Datenstrukturen oder bestimmte Algorithmen eingehen zu müssen. Durch die Vorgabe gewisser Modellierungstechniken erzwingt man die Behandlung bestimmter Problemklassen und Problemlösungsstrategien, ohne die Implementierungsplattform festlegen zu müssen. Als prüfungsrelevant werden damit nur Fertigkeiten im *Modellieren* und nicht im *Programmieren* angesehen. Diese Vorgehensweise soll nun anhand einer beispielhaften Kurssequenz erläutert werden (siehe auch Hubwieser (1999)).

#### **4.4.3 Unterricht auf der Grundlage von Modellierungstechniken**

Wie könnte nun ein solcher, im Wesentlichen von Modellierungstechniken be-stimmter Unterricht in der Praxis aussehen? Während man im Rahmen der profes-sionellen Softwareentwicklung in der Analyse- und Entwurfsphase meist eine *Kombination* aus mehreren verschiedenen Modellierungstechniken verwendet, bevor man sich für eine oder mehrere Implementierungstechniken entscheidet, ist es im Schulunterricht aus nahe liegenden didaktischen Gründen geboten,

- die Modellierungstechniken zunächst *einzeln* einzuführen,
- *einzeln* auf geeignete Probleme anzuwenden und
- die erzeugten Modelle möglichst sofort zu implementieren.

Wir wollen nun eine beispielhafte Folge solcher Modellierungstechniken für die Mittelstufe des Gymnasiums als Kern einer allgemein bildenden Schulinformatik näher untersuchen. Bei den vorgeschlagenen Techniken handelt es sich von ihrem Wesen her um abstrakte Visualisierungen (Diagramme) im Sinne von Reichen-berger u. Steinmetz (1999), die zum Teil von Rumbaugh et al. (1991) beschrieben und von uns an die Anforderungen der Schule adaptiert wurden (siehe auch Hub-wieser u. Broy (1996), Hubwieser (1999)).

**Funktionale Modellierung.** Die Verteilung von Informationssystemen gewinnt zunehmend an Bedeutung. Bei der funktionalen Modellierung mit Hilfe von Da-tenflussdiagrammen (ebenfalls nach Rumbaugh (1991)) werden zunächst komplexe Systeme in leichter überschaubare Bausteine (Teilsysteme) zerlegt (siehe Abb. 4.2) und der Informationsfluss zwischen diesen Modulen untersucht. Da man mit

Hilfe von Datenflussdiagrammen natürlich auch viele Informatiksysteme strukturieren und übersichtlich darstellen kann, scheint es sinnvoll, die Folge der Modellierungstechniken damit zu beginnen.

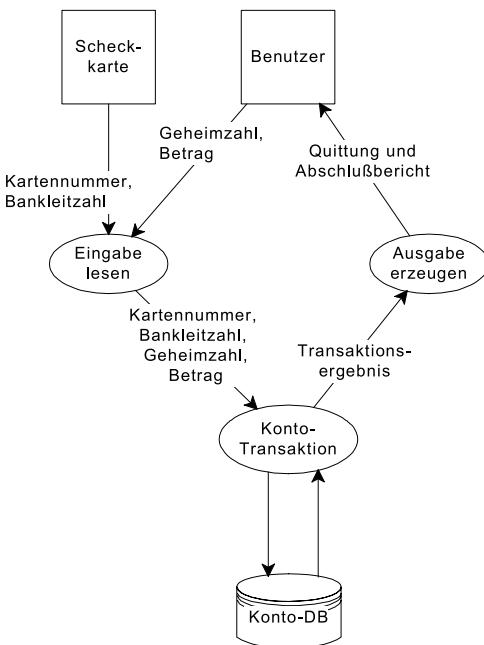


Abb. 4.2. Funktionales Modell eines Geldautomaten

Soweit sich Datenflussdiagramme auf datenverarbeitende Prozesse beschränken, die durch (nichtrekursive) Funktionen beschreibbar sind und auf Datenspeicherung verzichten, können sie wie in Abb. 4.3 gezeigt in den Rechenblättern eines Tabellenkalkulationssystems implementiert werden.

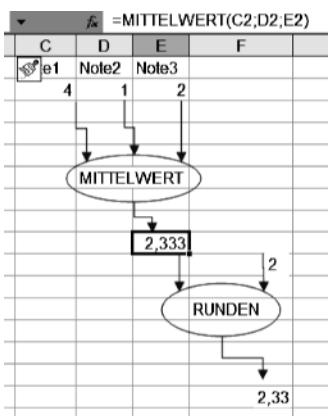


Abb. 4.3. Implementierung eines funktionalen Modells in einem Rechenblatt

Dabei beschränken wir uns auf die Verwendung der Funktionen, die bereits vom Tabellenkalkulationssystem zur Verfügung gestellt werden und betrachten diese zunächst als „black boxes“ (siehe Hubwieser (2004) und Hubwieser (2005)).

Später (d.h. nach einer Einführung in die Grundkonzepte der Programmierung) können die Teilsysteme in Form von Funktionen, Prozeduren oder Methoden ausprogrammiert werden, wobei die Schüler sehr gut in Gruppen mit unterschiedlicher Aufgabenstellung arbeiten können.

**Datenmodellierung** ist eine relativ einfach zu erlernende Technik, die anhand statischer Beispiele näher betrachtet und eingeübt wird. Bei der Umwandlung von Entity-Relationship-Modellen (nach Chen (1976), siehe Abb. 4.4) in Systeme von Tabellen lernen die Schülerinnen und Schüler die Grundstrukturen relationaler Datenbanksysteme kennen. Bei der Arbeit mit den Tabellen erkennen sie die Notwendigkeit der Sicherung von Datenkonsistenz und die Nachteile von Datenredundanzen. Die Betrachtung von Transaktionen leitet schließlich zu einer ablauforientierten Sicht über, welche die anschließende Einführung von Zustands-Übergangsdiagrammen motiviert.

Da in letzter Zeit bei Datenmodellen auch im Datenbankbereich zunehmend UML-Klassendiagramme eingesetzt werden, empfehlen wir, diese auch im Unterricht anstatt der klassischen ER-Modellierung zu verwenden. So kann den Schülerinnen und Schülern die Gewöhnung an eine der ohnehin schon zahlreichen Modellierungstechniken erspart werden, da sie später ohnehin mit Klassendiagrammen arbeiten müssen.

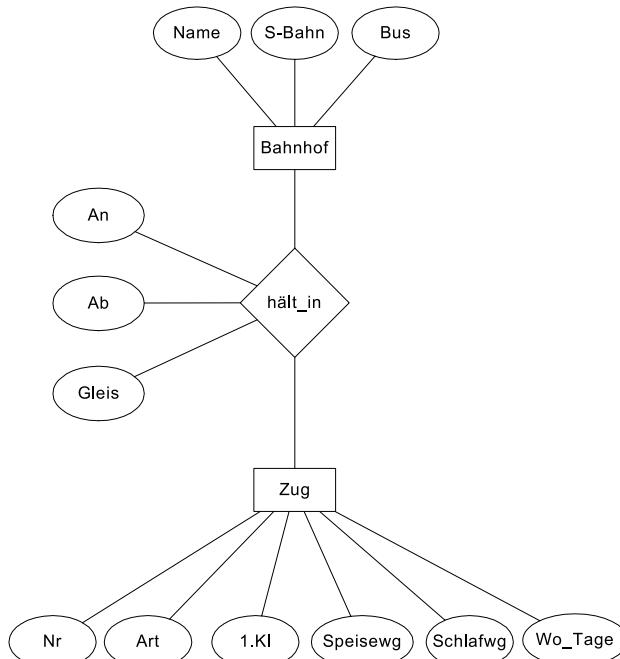


Abb. 4.4. ER-Modell eines Fahrplanausschnittes

**Zustandsorientierte Modellierung.** Im Zusammenhang mit Datenbanktransaktionen sind die Schüler bereits einem einfachen Zustandskonzept (Datensatz gesperrt/freigegeben) begegnet. Dieses wird nun vertieft, indem wir Abläufe aus dem Alltagsleben mit Hilfe von Zustands- Übergangsdiagrammen modellieren.

Diese Beschreibungstechnik abstrahiert reale, kontinuierliche zeitliche Abläufe, indem sie einzelne Abschnitte ihres zeitlichen Verlaufs jeweils einer der beiden folgenden Kategorien zuordnet:

- relativ lang andauernde *Zustände* (symbolisiert durch abgerundete Rechtecke), welche einen andauernden Vorgang und/oder einen konstanten Wert eines Attributs repräsentieren und
- in verschwindend kurzen Zeitspannen ablaufende *Übergänge* zwischen diesen Zuständen (symbolisiert durch Pfeile). Jeder Pfeil wird zumindest mit der Aktion, welche den jeweiligen Übergang auslöst, markiert.

Die Notation entnehmen wir Rumbaugh (1991). Aus dem täglichen Leben kennen die Schülerinnen und Schüler technische Automaten verschiedenster Art, die hier gut als Einstieg dienen können (siehe Abb. 4.5).

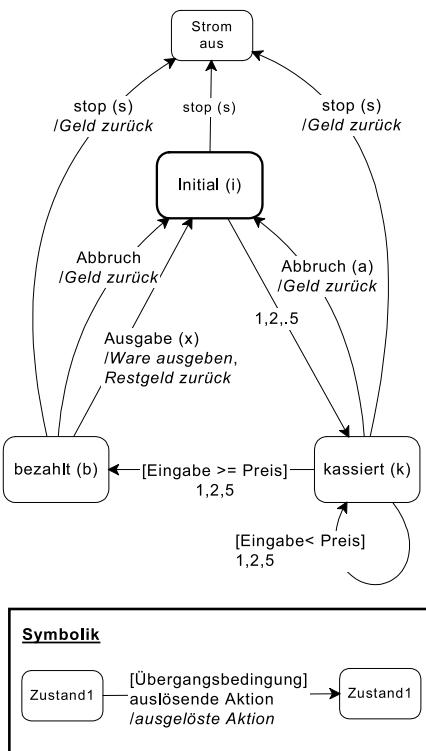


Abb. 4.5. Zustands- Übergangsdiagramm eines Getränkeautomaten

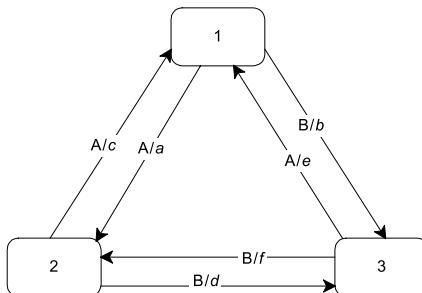
Die Diagramme werden anschließend in imperative Programme umgesetzt, wobei sich den Schülern auf natürliche Weise das Variablenkonzept und der Ablauf eines imperativen Programms als Spur im Zustandsraum erschließt. Um die Programmierung im Zaum zu halten, bestehen wir darauf, dass nur Abläufe implementiert werden, die vorher durch ein Zustands- Übergangsdiagramm beschrieben worden sind. Dies geht nach einem festen Schema vor sich, das für den Automaten aus Abb. 4.5 das folgende Programmfragment liefert:

```

Wiederhole bis Endbedingung erreicht
Falls Zustand =
1:  Falls Eingabe =
      A: Aktion a; Zustand:=2
      B: Aktion b; Zustand:=3
2:  Falls Eingabe =
      A: Aktion c; Zustand:=1
      B: Aktion d; Zustand:=3
3:  Falls Eingabe =
      A: Aktion e; Zustand:=1
      B: Aktion f; Zustand:=2

```

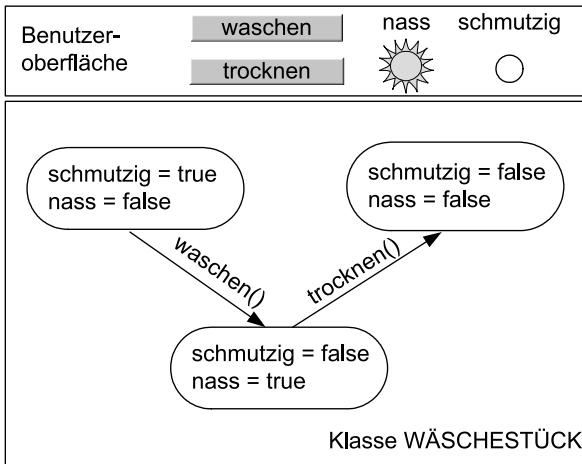
Mit der verwendeten Notation für Zustands- Übergangsdiagramme lassen sich bei Bedarf mit Hilfe von speziellen Datenstrukturen (Keller, Band) auch Kellerautomaten oder Turingmaschinen darstellen. Allerdings ist dies wohl erst in einem Oberstufenkurs machbar.



**Abb. 4.6.** Schematisches Zustands-Übergangsdiagramm

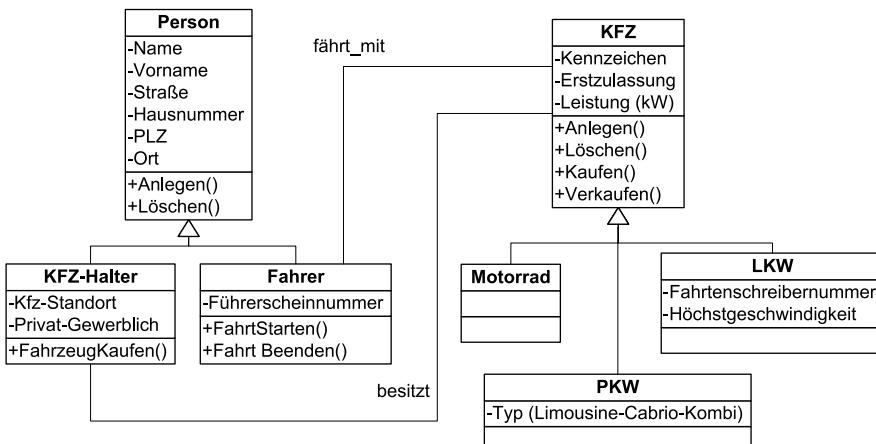
**Objektorientierte Modellierung.** Nun wird die im ersten Abschnitt vorgenommene Unterteilung komplexer Systeme in Subsysteme mit Hilfe der objektorientierten Modellierung bis auf Objektebene verfeinert. Es werden Objekte identifiziert und analysiert, ihre Attribute bestimmt und ihre Methoden erarbeitet. Diese Sichtweise kommt der natürlichen Art des Menschen, seine Umwelt wahrzunehmen, besonders nahe (siehe Anderson (1985) und Schwill (1995)). Die vorliegenden Erfahrungen der Schüler mit der ER-Modellierung stellen eine gute Grundlage dar. Der Objektbegriff wird zunächst mit Hilfe des Automatenkonzeptes sorgfältig fundiert: Objekte erscheinen als gekapselte Automaten, wobei der Zugriff auf interne Zustandsvariable durch öffentlich zugängliche Methoden realisiert wird (siehe Abb. 4.7). Bei Modellierung und Implementierung ist sehr sorgfältig ist auf eine saubere begriffliche Unterscheidung von Klassen als abstrakten

Beschreibungen (analog zu Tabellschemata) und Instanzen als konkreten Ausprägungen von Klassen (analog zu Datensätzen) zu achten.



**Abb. 4.7.** Ein Objekt der Klasse WÄSCHESTÜCK als gekapselter Automat

Das Verhalten des Gesamtsystems entsteht aus dem Zusammenspiel seiner Objekte, die durch Beziehungen verbunden sind und gegenseitig ihre Methoden durch Botschaften aktivieren. Später können dann auch noch weiterführende objektorientierte Konzepte wie z.B. Klassenhierarchien besprochen werden (siehe Abb. 4.8).

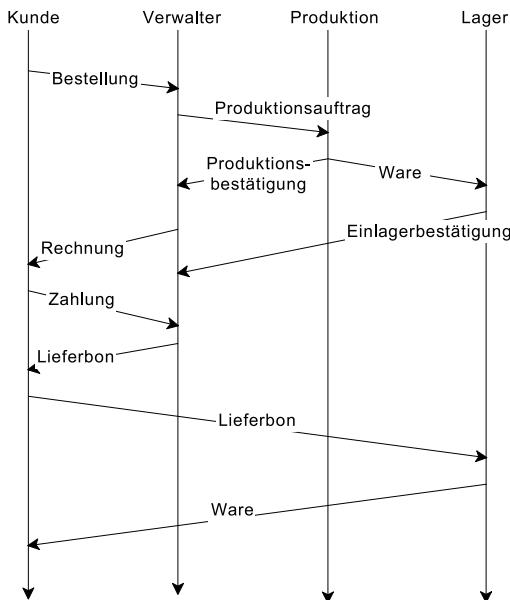


**Abb. 4.8.** Ein Klassendiagramm mit Klassenhierarchien

Gerade im Bereich der Objektorientierung vermischen sich oft die Begriffswelten von *Modellierung* und *Programmierung*. Im Hinblick auf Allgemeinbildung kommt es uns dabei hauptsächlich auf die Modellierung an. Die objektorientierte

Programmierung bietet lediglich eine einfache und effiziente Möglichkeit, die erarbeiteten Modelle zu implementieren, liefert aber für sich genommen kaum Beiträge zur Allgemeinbildung.

**Verteilte Systeme.** Unter Verwendung von Interaktionsdiagramm (siehe Abb. 4.9) modellieren wir jetzt den Ablauf von Kommunikationsprozessen zwischen einzelnen Objekten. Diese Art der Darstellung orientiert sich an *Time Sequence Diagrams* (siehe Facci (1995)) und Message Sequence Charts (siehe Rudolph et al. (1996)). An dieser Stelle kommen auf natürliche Weise Aspekte der Parallelität von Abläufen ins Spiel, wenn Objekte während der Ausführung fremder Methoden nicht mehr warten, sondern selbst Aktionen ausführen. Insbesondere die Regelung der Zugriffe auf gemeinsame Ressourcen bedarf einer genaueren Betrachtung. Über Synchronisationskonzepte für parallele Prozesse gelangen wir schließlich zur Untersuchung lose gekoppelter verteilter Systeme als Modelle für Rechnernetze. Dabei soll speziell auf die wichtigsten Eigenheiten der Internettechnik eingegangen werden, um deren große Bedeutung für Berufswelt und Privatsphäre zu würdigen. Die Implementierung von parallelen Prozessen auf Schulniveau kann z.B. durch quasiparallelen Ablauf von mehreren Programmen in verschiedenen Shell-Fenstern oder mit Hilfe von nebenläufigen Prozeduren bzw. Methoden simuliert werden, wie wir es bereits in Hubwieser u. Broy (1996) vorgeschlagen haben.



**Abb. 4.9.** Interaktionsdiagramm einer Auftragsbearbeitung

**Abschlussprojekt.** In einem großen Abschlussprojekt entwerfen und implementieren die Schüler schliesslich unter simultaner Verwendung mehrerer der bisher gelernten Modellierungstechniken ein größeres Softwaresystem. Die Teamfähigkeit, Zeitplanungsstrategien und Kommunikationstechniken werden dabei entwickelt und eingeübt.

Zu den hier dargestellten Lehrplansequenz folgen in Teil C exemplarische Unterrichtssequenzen, in denen u.a. die unterrichtliche Umsetzung von datenorientierter und zustandsorientierter Modellierung detailliert beschrieben wird.

## 5 Ein Gesamtkonzept

Nachdem wir im letzten Kapitel mit Hilfe des Grundschemas der Informationsverarbeitung eine Grundmenge möglicher Lerninhalte der Schulinformatik abgesteckt und den Schwerpunkt *Modellierung* ausführlich behandelt haben, wollen wir uns nun einer exemplarischen Umsetzung an einem Schultyp zuwenden. Es liegt nahe, dafür das Gymnasium auszuwählen, da es aufgrund seines (relativ) hohen intellektuellen Anspruchs und seiner langen Ausbildungsdauer den umfassendsten Katalog von Lerninhalten aufnehmen kann. Wir gehen dabei von einer achtjährigen Ausbildungsdauer aus, die inzwischen in der Mehrheit der deutschen üblich ist. Als Bezugsmodell dient das neue achtjährige bayerische Gymnasium (G8).

Die anderen Schularten können naturgemäß nur Teile des gymnasialen Lernstoffs anbieten, die zusätzlich in Darbietung und Abstraktionsniveau an die veränderten Zielsetzungen angepasst werden müssen. Soweit es länderspezifische Vorgaben zu beachten gilt, orientieren wir uns an bayerischen Verhältnissen.

### 5.1 Die Rahmenbedingungen

Wie bereits in der Einleitung zu Kapitel 2 ausgeführt wurde, ist es außerordentlich schwierig, im Kampf um wertvolle Unterrichtszeit mit den etablierten Fächern zu konkurrieren. Infolgedessen wird die Informatik Abstriche von der aus ihrer Sicht optimalen Stundenzahl hinnehmen müssen. Unter anderem sind die folgenden organisatorischen Randbedingungen zu beachten:

1. In der ersten Jahrgangsstufe des Gymnasiums muss sehr behutsam vorgegangen werden. Stoffüberfrachtung wäre hier ein besonders schwerer Fehler. Deshalb hat ein neues Fach keine Aussichten, in dieser Jahrgangsstufe aufgenommen zu werden. Dennoch sollte die Informatik möglichst früh beginnen.
2. Um Anspruch auf Zulassung als Leistungskursfach zu erlangen, sollten drei Jahrgangsstufen unter Beteiligung von Stufe 10 im Pflichtfachbereich vertreten sein.
3. Einstündige Fächer sind pädagogisch generell nicht sinnvoll, für die Informatik speziell wegen der häufigen praktischen Arbeit an Rechenanlagen keinesfalls akzeptabel.
4. In den Jahrgangsstufen 6, 7 und 8 existiert ein starker Stoffdruck zahlreicher anderer Fächer, insbesondere der Fremdsprachen.

Aus diesen obigen Randbedingungen ergibt sich zusammen mit gewissen inhaltlichen Minimalanforderungen der folgende Vorschlag: Der Informatik-Pflichtunter-

richt beginnt in der 6. Jahrgangsstufe, worauf ein einjähriges Wahl- oder Wahlpflichtangebot folgt. Ab der 8. Jahrgangsstufe bis zur 10. Jahrgangsstufe wird die Informatik als Pflichtfach fortgesetzt. Das Fach ist in allen genannten Jahrgangsstufen mindestens zweistündig.

## 5.2 Die Unterrichtsmodule

Gemäß der unterschiedlichen Entwicklungsphasen der Schüler haben die entsprechenden Unterrichtsmodule verschiedene Zielsetzungen und jeweils andere dominierende Lerninhalte. Für ein jeweils zweistündiges Unterrichtsfach am Gymnasium schlagen wir die in Tabelle 5.1 dargestellte Aufteilung und thematische Gewichtung vor.

**Tabelle 5.1.** Die Module informatischer Bildung am Gymnasium

JGSt.	Ausrichtung	dominante Lerninhalte	vorherrschende Arbeitstechnik
6	Grundlagen, Terminologie, Vorbereitung für Medieneinsatz	Repräsentation von Information: Datenstrukturen und Verarbeitungsvorschriften	Arbeiten mit Dokumenten
7	Verbreiterung und Anwendung der Kenntnisse	s. oben	Arbeiten mit Dokumenten
8–10	Allgemeinbildung	Systembeschreibungen	Modellierung und Simulation
11, 12	Allgemeine Studien- und Berufsvorbereitung	Konstruktions- und Funktionsprinzipien von Informatiksystemen	Analyse und Konstruktion von Informatiksystemen

Im Folgenden soll dieses Gerüst mit Lerninhalten ausgefüllt werden. Die genannten Themen sind dabei als Angebot zur Auswahl je nach den vorliegenden Umständen gedacht, keinesfalls zur vollständigen Behandlung.

### 5.2.1 Das Fundamentum

Am Anfang jeder weiterführenden Bildung muss eine grundlegende Einführung in die wichtigsten allgemein bildenden Begriffe und Denkweisen der Informatik stehen. Dies sollte frühestens in der 6. Jahrgangsstufe geschehen, da die Schüler erst in diesem Alter die nötige Abstraktionsfähigkeit erlangen, um über das kon-

kret verwendete System hinaus allgemeinere, grundlegende Konzepte zu verstehen (siehe Teil A, Abschnitt 1.3). Andererseits darf der erste Kontakt mit der Informatik nicht länger hinausgeschoben werden, da sich sonst durch die unvermeidliche häusliche Auseinandersetzung mit Informatiksystemen „selbst gestrickte“, eventuell unzutreffende mentale Modelle (siehe Abschnitt 2.2.2) bilden, deren Korrektur später sehr aufwendig sein kann. Zudem wird es mit zunehmendem Alter immer schwieriger, den Informatik-Anfangsunterricht gleichermaßen auf die Bedürfnisse von Computerbesitzern und -nichtbesitzern, Mädchen und Jungen abzustimmen.

Das Vorgehen ist altersgemäß spielerisch und handlungsorientiert, jedoch keinesfalls ungenau oder unsystematisch. Hier wird der Grundstein für den Aufbau angemessener mentaler Modelle und die Verwendung einer sauberen, ausdrucksstarken Terminologie in späteren Jahren gelegt. Davon profitiert auch der Unterricht in anderen Fächern, wenn Informatiksysteme als Medien oder Lernhilfen eingesetzt werden sollen.

**Tabelle 5.2.** Lerninhalte des Fundamentums

---

Erstellung und Präsentation von Dokumenten	Textverarbeitung: Texte, Absätze, Zeichen als Objekte mit Attributen. Markieren, Ausschneiden, Einfügen als grundlegende Operationen auf Texten, Einbau von Grafiken Hypertext: Struktur von Hypertextdokumenten, Einbau von Verweisen und multimedialen Objekten Tabellenkalkulation: Zeilen, Spalten und Zellen, Typen von Zellinhalten, Verweise auf andere Zellen, einfache Rechenfunktionen Grafikprogramme: typische Datenstrukturen von Vektor- und Pixelgrafik, mögliche Operationen darauf, Abschätzung des Platzverbrauchs, Manipulationsmöglichkeiten
Verwaltung und Transport von Dokumenten	Ordner zur hierarchischen Verwaltung von Dateien, beispielhafte Ordnersysteme, Verschieben und Kopieren von Dateien und Unterordnern, Hierarchische Dateisysteme: Dateien als Container für die Daten von Standardsoftwareprogrammen, Löschen und Kopieren von Dateien Versand von Dokumenten mit Hilfe von elektronischer Post Publikation von Dokumenten im Internet, Client-Server Prinzip Systematische Suche nach Dokumenten in Verzeichnisstrukturen und im Internet
Verarbeitung von Dokumenten, Automatisierung der Verarbeitung	Umgangssprachliche Formulierung von Verarbeitungsvorschriften, Bausteine von Algorithmen: Sequenz, Wiederholung und Auswahl Funktionsweise von Rechenanlagen in Blockschaltbildern

---

Wir orientieren uns in dieser Stufe am Begriff *Dokument* und den damit verbundenen Operationen. Dokumente und ihre Komponenten werden als Objekte

mit charakteristischen Attributen und zugeordneten Methoden aufgefasst. Im Zentrum der unterrichtlichen Betrachtungen steht vor allem die Analyse von typischen Dokumentenklassen wie Texten, Tabellen, Grafiken. Als Lerninhalte schlagen wir eine *Auswahl* der in Tabelle 5.2. aufgezählten Themen vor.

Zur Abschätzung der Schwierigkeit der Lerninhalte haben wir in Tabelle 5.3 die Konzepte, Klassen, und Operationen aufgeführt, die den Schülerinnen und Schülern auf dem Weg durch das Fundamentum der Informatik begegnen. Zahlreiche Anregungen zu dieser Objektsicht können Baues et al. (1996) entnommen werden. Auch Knapp u. Fischer (1998) beschäftigen sich intensiv mit diesem objektorientierten Einstieg in den Informatikunterricht.

**Tabelle 5.3.** Konzepte, Klassen und Operationen im Modul Fundamentum

Thema	Neue Konzepte	Typische Klassen	Operationen
Datenstrukturen von Vektorgrafik	Klassen und Instanzen, Attribute	Linie, Rechteck, Textbereich,	Markieren, Löschen, Kopieren, Einfügen, Skalieren, Verschieben
Datenstrukturen von Textdokumenten	Beziehungen zwischen Klassen	Textdokument, Absatz, Zeichen, Seite	
Verwaltung von Dokumenten	Hierarchische Strukturen	Datei, Ordner	Erzeugen, Verschieben, Kopieren, Verknüpfen, Öffnen
Versand von Dokumenten	Adressierung, Abruf und Zustellung, Client-Server-Konzept	E-Mail-Dokument, Anhang, Adresse, Sender, Empfänger, Vermittler	Abschicken, Vermitteln, Abrufen, Empfangen, Öffnen
Vernetzen und Publizieren von Dokumenten	Vernetzung, Verweis	Dokument, Server, Client	Anbieten, Abrufen
Automatische Verarbeitung von Information	Bausteine von Algorithmen	Anweisungen, Programm(-text)	Ausführen, Wiederholen, Auswählen

Um Missverständnissen vorzubeugen, wollen wir betonen, dass wir in diesem Modul zwar objektorientiert analysieren und modellieren, aber keinesfalls objektorientiert programmieren wollen. Die Lernziele aus dem Bereich „Automatisierung von Verarbeitungsvorgängen“ können problemlos durch sorgfältige umgangssprachliche Formulierung von Verarbeitungsvorgängen (z.B. zur Erstellung von Serienbriefen) erreicht werden. Falls man programmieren will, empfehlen wir einfache, intuitiv zu verstehende Programmiersysteme wie etwa den Roboter *Karel* (siehe Abschnitt 3.4.4).

Zusätzlich zum eigentlichen Informatik-Anfangsunterricht sollte ein *freiwilliger Schreibmaschinenkurs* angeboten werden, um den Kindern von Anfang an die richtige Bedienung der Tastatur nahe zu bringen.

### 5.2.2 Die Wahlmodule

In optionalen, jeweils 2-stündigen Kursen aus dem Wahlpflichtbereich werden im Verlauf der Jahrgangsstufen 7 und 8 die im Fundamentum bereits systematisch vermittelten Kenntnisse und Fähigkeiten genutzt, um auf anschauliche Weise schülergemäße Problemstellungen mit Hilfe von Informatikanwendungen zu lösen.

**Tabelle 5.4.** Themenschwerpunkte der Wahlpflichtmodule

---

Textverarbeitung	Einbindung von Grafiken, Tabellen und anderen Objekten in Texte, Layoutprinzipien, Layoutvorlagen, Serienbriefe, Grundprinzipien von Texterkennungssystemen
Hypertextstrukturen	Abbildung logischer Beziehungsgefüge in Dokumentenstrukturen, Anreicherung mit multimedialen oder aktiven Elementen, Präsentation durch Webserver
Kommunikation in Rechnernetzen	Dienste in Rechnernetzen (Vertiefung), Informationssuche im Internet, Versand und Herunterladen von Dokumenten, Produktion und Veröffentlichung eigener Dokumente
Tabellenkalkulation	Berechnungen und Präsentation mit Hilfe von Tabellenkalkulation, ausführliche Gestaltung von Präsentationsgrafiken, Einsatz von Tabellenkalkulationen im Bereich der Wirtschaft
Computergrafik	Pixel- und Vektorgrafiken, Abschätzung des Platzbedarfs, Digitalisierung von Grafiken, mathematische Formulierung von ebenen oder räumlichen Transformationen, Grundstrukturen von CAD-Systemen
Konzeption von Rechenanlagen	Aufbau und Funktionsweise von Computern und Computernetzen, Konzeption und Planung von lokalen Netzwerken, Rechte- und Verzeichnisstrukturen, Sicherungsstrategien

---

Das spielerische Moment steht immer noch im Vordergrund. Es ist allerdings darauf zu achten, dass keine wesentlichen Inhalte aus den folgenden Pflichtmodulen vorweggenommen werden. Tiefergehende Betrachtungen über Modellierungstechniken sind daher zu vermeiden.

Diese Stufe sichert für besonders interessierte Schüler die Kontinuität ihrer Beschäftigung mit der Informatik. Diese Kontinuität ist auch unabdingbare Voraussetzung für eine aktive Beteiligung ausgewählter Schüler am Betrieb der schulischen Rechneranlage, ohne die heute kaum eine größere Schule mehr auskommen kann. Thematisch steht weiterhin der Begriff *Dokument* unter Betonung typischer

Datenstrukturen im Vordergrund. Mögliche Themenschwerpunkte der einzelnen Module zeigt Tabelle 5.4. Selbstverständlich ist, je nach den Interessenlagen der Schüler, auch eine Kombination aus diesen Modulen denkbar.

### 5.2.3 Informatische Allgemeinbildung

Während bei den bisher besprochenen Modulen der Aufbau eines soliden begrifflichen Fundamentes im Vordergrund stand, rückt nun der eigentliche allgemein bildende Kern der Schulinformatik in den Vordergrund. Das Ausbildungsziel dieser Stufe ist die Vermittlung einer *vertieften informatischen Allgemeinbildung* „zur Vorbereitung auf Studium und Beruf“ entsprechend den gesetzlichen Vorgaben (siehe Abschnitt 2.1.1). Zentrales Unterrichtsthema ist hier die *Modellierung* von realen oder geplanten Systemen mit Hilfe spezieller, altersgemäßer Beschreibungstechniken (siehe Abschnitt 4.3). Die Schüler befinden sich in der 9. bis 11. Jahrgangsstufe, das entspricht einem Altersbereich von 14–17 Jahren. Die Kenntnisse aus den Wahlmodulen können als willkommene Hilfen zur Veranschaulichung und Bereicherung des Unterrichtsgeschehens dienen, sie dürfen aber nicht vorausgesetzt werden. Tabelle 5.5 zeigt mögliche Lerninhalte dieser Module. Eine eingehendere Besprechung der kausalen Zusammenhänge des hier aufgeführten Unterrichtsstoffes findet sich in Abschnitt 4.4.3.

Auch hier zeigt die Informatik, wie bereits im Fundamentum, einen nützlichen Nebeneffekt auf den Unterricht anderer Fächer: Besonders in den Naturwissenschaften werden Ausschnitte der realen Welt oft auf abstrakte, quantifizierbare Modelle abgebildet. Aus Unkenntnis der zugrunde liegenden Denkweisen entsteht oft Unverständnis bei den Schülern, vor allem indem Aussagen über Modelle in naiver Weise auf die Wirklichkeit bezogen werden. Eine wichtige Aufgabe des Fachs Informatik in der Mittel- und Oberstufe ist es daher, solche Modellbildungsverfahren vor dem Erfahrungshintergrund der Schüler aufzuzeigen und diese damit nachvollziehbar und selbstständig anwendbar zu machen.

**Tabelle 5.5.** Lerninhalte der allgemein bildenden Module

---

Repräsentation von Information	Wiederholung und Systematisierung der Vorkenntnisse: Datenstrukturen: Objekte und Attribute von Standardsoftware Verarbeitungsvorschriften: Algorithmen, Kontrollstrukturen Systembeschreibungen: Datenflussdiagramme von Rechenanlagen
Datenmodellierung	Modellierung mit Hilfe von Entity-Relationship-Diagrammen: Objekte (Entitäten); Klassen (Entitätsmengen); Attribute und Wertebereiche; Beziehungen zwischen Klassen Umsetzung von ER-Modellen in relationale Datenbanken: Schema, Tabellen, Spalten, Datentypen, Datensätze, Schlüssel, Redundanz, Konsistenz, Normalformen 1-3 Operationen auf Tabellen: Datenmanipulation, Abfragen, Berichte, Datenschutz, Client-Server Prinzip

Zustandsorientierte Modellierung	Modellierung von Abläufen mit Hilfe von Zustandsübergangsdiagrammen: Zustände und Übergänge; andauernde, auslösende und ausgelöste Aktionen; endliche deterministische Automaten mit und ohne Ausgabe Implementierung von Automaten durch imperative Programme: Bausteine von Algorithmen (Zuweisung, Kontrollstrukturen: Sequenz, Auswahl, Wiederholung), Darstellung durch Struktogramme, Zustände von Variablen, Steuerung der Übergänge durch Kontrollstrukturen, Syntaxdiagramm Automatenmodell des Von-Neumann-Rechners: Realisierung von Variablen und Verarbeitungsvorschriften durch Zustände von Speicherzellen
Funktionale Modellierung	Aufteilung komplexer Systeme in Teilsysteme (Komponenten), Datenflussdiagramme, schrittweise Verfeinerung Modularisierung von imperativen Programmen: Funktionen und Prozeduren; Schnittstellen; Rekursion als Hilfsmittel zur Problemlösung: Rekursive Beschreibung von Algorithmen aus funktionaler Sicht
Objektorientierte Modellierung	Objektmodell: Identifikation von Objekten, Klassenbildung Objekte als Automaten: Objekte vereinen Datenstrukturen (Attribute) und Verarbeitungsvorschriften (Methoden); Attributname und -wert; Objekt als Instanz seiner Klasse; Datenkapselung; Klassenhierarchien Kommunikation zwischen Objekten: Gegenseitiger Aufruf von Methoden durch Botschaften
Softwareprojekte	Phasenmodell der Softwareentwicklung: Planung, Analyse, Entwurf, Implementierung, Test, Einführung, Wartung, Pflege Anwendung des Phasenmodells bei der Konstruktion und Simulation eines komplexen Systems Beachtung von Datenschutz und Datensicherheit Simultaner Einsatz mehrerer Modellierungstechniken Erstellung einer ausführlichen Dokumentation mit kritischer Wertung des gesamten Projektverlaufs

---

Wie in den Abschnitten 3.3 und 3.4 bereits ausführlich dargelegt, sollten die von den Schülern konstruierten Modelle in der Regel auch implementiert werden. Welches System man dafür verwendet, hängt einerseits natürlich von der jeweiligen Implementierungstechnik, andererseits aber auch von der unterrichtlichen Vorgesichte, dem schulischen Umfeld, der Vorbildung der Lehrkraft und der installierten bzw. verfügbaren Software ab. Tabelle 5.6 unterbreitet einige Vorschläge für passende Implementierungswerkzeuge (siehe Hubwieser (1999)).

**Tabelle 5.6.** Implementierungsvorschläge zu den Modellierungstechniken

Schwerpunkt	Modellierungstechnik	Implementierung
Wiederholung, Einführung, Überblick	Repräsentationsformen: Datenstrukturen, Verar- beitungsvorschriften, Systembeschreibungen	Standardsoftware
Datenstrukturen	ER-Modellierung	Relationale DB-Systeme
Abläufe	Zustands-Übergangs- diagramme	Variable, Zuweisung und Kontrollstrukturen in einer imperativen Programmier- sprache
Dekomposition in Subsysteme	Datenflussdiagramme	Funktionen (bzw. Prozedu- ren) in einer Tabellenkalkula- tion und einer imperativen Programmiersprache
Kommunikation zwischen Objekten	Objektdiagramme, Mes- sage-Sequence Charts	Objektorientierte Program- miersprache

## 5.2.4 Oberstufe

Auf lange Sicht wird sich in allen Bundesländern eine spezielle Oberstufeninformatik durchsetzen. In Grund- und Leistungskursen werden besonders an der Informatik interessierte Schülerinnen und Schüler ihre Kenntnisse mit zum Teil bereits *wissenschaftspropädeutischer Ausrichtung* vertiefen. Hier würden spezielle Konzepte und Methoden der Informatik im Vordergrund stehen, insbesondere soweit sie zu einer Beurteilung der Möglichkeiten und Grenzen von Informatiksystemen beitragen. Im Fall einer Abiturprüfung in Informatik sind die Rahmenbedingungen der Beschlüsse der Kultusministerkonferenz über die „Einheitlichen Prüfungsanforderungen in der Abiturprüfung Informatik“ in der jeweils gültigen Fassung einzuhalten. Derzeit gilt der Beschluss von 1989 (KMK 1991), in dem sich unter „fachliche Inhalte“ folgenden Themenbereiche finden:

### 1.1.1. Algorithmische Grundlagen, Gegenstände und Methoden der Informatik

- Algorithmen und Datenstrukturen
- Programme
- Methoden der Software-Entwicklung

### 1.1.2 Funktionsprinzipien von Hard- und Software-Systemen

- Anwendersoftware
- Programmiersprachen und -umgebungen
- Betriebssoftware
- Rechnermodelle und Rechnerkonfiguration
- Theoretische Grundlagen

### 1.1.3 Anwendungen von Hard- und Software-Systemen und deren gesellschaftliche Auswirkungen

- Anwendungsbereiche
- Mensch-Maschine-Schnittstelle
- Grenzen und Möglichkeiten/Chancen und Risiken des Einsatzes der Informations- und Kommunikationstechniken

Inhaltlich wird die Modellbildung nach wie vor eine tragende Rolle spielen, allerdings werden auch mehr und mehr spezielle Aspekte der theoretischen Fundierung sowie der Implementierung besprochen (siehe Tabelle 5.7).

**Tabelle 5.7.** Mögliche Lerninhalte der Oberstufe

Themenbereich	Lerninhalte Grundkurs	Vertiefung im Leistungskurs
Graphen, rekursive Datenstrukturen und Algorithmen	Listen, Bäume, allg. Graphen, spezielle Algorithmen darauf	Funktionale Programmierung, Effizienzüberlegungen (mehrfache Verkettung etc.)
Automaten und formale Sprachen	Reguläre Sprachen und endliche Automaten, kontextfreie Sprachen, lexikalische und syntaktische Analyse	Kellerautomaten, Registermaschine und einfache Assemblersprache, semantische Synthese
Parallele Prozesse	Petrinetze, Nebenläufigkeit, Vergabe von Betriebsmitteln, Synchronisation, Kommunikation, Protokolle, einfache Schichtenmodelle, Dienste globaler Netze	vollständiges Schichtenmodell, Prozess- und Speicherverwaltung von Betriebssystemen
Korrektheit, Berechenbarkeit und Komplexität	Korrektheitsüberlegungen, und Turingmaschine, Band- und Speicherkomplexität	Hoare-Kalkül, einfache Korrektheitsbeweise, Komplexitätsklassen P und NP

## 5.3 Vorschläge für andere Schularten

Trotz ihres anders gearteten Bildungsauftrages sind große Teile unserer obigen Vorschläge für das Gymnasium auch in den anderen allgemein bildenden Schularten einsetzbar. Wir gehen dabei grundsätzlich von einem zweistündigen Fach aus. Für berufliche Schulen können allerdings aufgrund ihres breiten berufsspezifischen Anforderungsspektrums vom Gärtner bis zum Informationselektroniker keine einheitlichen Aussagen bezüglich der Lerninhalte gemacht werden.

### 5.3.1 Realschule

Im Bereich der Realschulen gilt es vor allem zu berücksichtigen, dass die Ausbildungszeit (der sechsjährigen Form) um ein Jahr kürzer ist, als die Ausbildungszeit des Gymnasiums bis zur Kollegstufe. Zweitens dürfte die Abstraktionsfähigkeit der Schüler im Durchschnitt unter der altersgleicher Gymnasiasten liegen. Drittens absolvieren die Abgänger der Realschule in der Regel nach ihrem Schulabschluss eine Lehrzeit und stehen damit wesentlich früher im Berufsleben als die Gymnasiasten. Daraus folgt ein geringerer Anspruch an die Langlebigkeit der Unterrichtsinhalte in Verbindung einer verstärkten Nachfrage nach praxisrelevanten Informatikkenntnissen. Dazu könnte etwa die in Tabelle 5.8 gezeigte Konstruktion passen.

**Tabelle 5.8.** Lerninhalte an der Realschule

JGSt	Schwerpunkt	Inhalte
6	Erstellung, Präsentation und Verwaltung von Dokumenten	Objektstruktur von Grafik- und Textverarbeitungsprogrammen, Anreicherung von Texten mit Grafiken, Ordner und Verzeichnisse
7	Transportieren, Vernetzen und Publizieren von Dokumenten	Objektstruktur und Funktionalität von Hypertext- und E-Mailsystemen, Erstellung und Publikation von Hypertextstrukturen, Anreicherung von Hypertexten mit multimedialen Elementen
8	Verarbeitung von Information	Serienbriefe mit Textverarbeitungen, Objektstruktur und Funktionalität von Tabellenkalkulationsprogrammen
9, 10	Datenorientierte Modellierung, Zustandsorientierte und Funktionale Modellierung	ER-Modellierung und Datenbanksysteme, Zusandübergangsdiagramme und imperative Programmierung, Datenflussdiagramme und Funktionen

### 5.3.2 Hauptschule

Die Anpassung gymnasialer Inhalte an die Hauptschule folgt demselben Muster wie oben für die Realschule, allerdings in verstärktem Maß: noch weniger Zeit, noch mehr Praxisorientierung, noch weniger Abstraktionsmöglichkeiten. Dies führt zu einem weiter reduzierten Vorschlag für die Lerninhalte (siehe Tabelle 5.9).

**Tabelle 5.9.** Lerninhalte an der Hauptschule

JGSt.	Schwerpunkte	Inhalte
6	Erstellung, Präsentation und Verwaltung von Dokumenten	Objektstruktur von Grafik- und Textverarbeitungsprogrammen, Anreicherung von Texten mit Grafiken, Ordner und Verzeichnisse
7	Transportieren, Vernetzen und Publizieren von Dokumenten	Objektstruktur und Funktionalität von Hypertext- und E-Mailsystemen, Erstellung und Publikation von Hypertextstrukturen, Anreicherung von Hypertexten mit multimedialen Elementen
8	Verarbeitung von Information	Objektstruktur und Funktionalität von Tabellenkalkulationsprogrammen
9	Datenorientierte Modellierung	ER-Modellierung und Datenbanksysteme

---

## **Teil C**

# **Unterrichtsbeispiele**

Unsere in den vorausgegangenen Kapiteln in knapper Form vorgestellten Konzepte zur Schulinformatik sollen nun im letzten Teil des Buches anhand einer Reihe von Unterrichtsskizzen näher erläutert werden. Diese sollen als exemplarische Projekte schlaglichtartig die wichtigsten Konzepte der Schulinformatik beleuchten. Wir erheben dabei keinen Anspruch auf Vollständigkeit oder auf die Darstellung einer konsequenten Einführung in diese Konzepte.

Wir beginnen mit der objektorientierten Modellierung gängiger Informatiksysteme (u.a. Grafikprogramme, Textverarbeitungssysteme und Hypertextsysteme) im Fundamentum, anhand derer die wichtigsten Begriffe der Schulinformatik eingeführt und erläutert werden. Darauf folgt in der Mittelstufe der Übergang zur Modellierung von Systemen aus der Alltagsumgebung der Schüler mit Hilfe diverser Modellierungstechniken (siehe auch Teil B, Abschnitt 4.4.3). Zur Simulation der Modelle werden hierbei wiederum Informatiksysteme herangezogen.

Für den folgenden Oberstufenunterricht machen wir vier exemplarische Vorschläge, die vor allem aufzeigen sollen, dass auch hier trotz der höheren Ansprüche an Komplexität und Abstraktion eine handlungsorientierte Ausrichtung des Unterrichts möglich ist.

Den Abschluss bildet eine durchgehende Projektsequenz zu einem Anwendungsfeld (Bankwesen), in der nochmals alle wesentlichen Modellierungstechniken aufgegriffen und an jeweils einem Beispiel illustriert werden.

Die folgenden Unterrichtssequenzen sind aus Gründen der Übersichtlichkeit nicht durchgängig im Sinne einer optimalen Unterrichtsvorbereitung ausgearbeitet. Detailliertere Hinweise wie Vorschläge für Hefteinträge (in Kleindruck) beschränken sich beispielsweise auf didaktisch besonders wichtige Stellen. Aufgaben und Lernziele werden nur dort angegeben, wo es zur Hervorhebung der Kerninhalte unbedingt notwendig ist.

Während im realen Unterricht Kontinuität und Sparsamkeit bezüglich der verwendeten Modellierungs- und Implementierungstechniken das Vorgehen bestimmen sollten, werden wir im folgenden eher Wert auf Vielfalt legen, um dem Leser eine möglichst große Auswahl solcher Techniken anbieten zu können.

# 1 Anfangsunterricht in Informatik

Es liegt nahe, unsere Folge von Unterrichtsvorschlägen mit dem gymnasialen Anfangsunterricht (oder Fundamentum, siehe Teil B, Abschnitt 5.2.1) zu beginnen, der allerdings (mehr oder weniger angepasst) auch in allen anderen Schularbeiten vermittelt werden kann. Dabei geht es vor allem um die objektorientierte Modellierung gängiger Informatiksysteme mit besonderer Betonung der Datenstrukturen. Aus abstrakterer Sicht bearbeiten wir verschiedene Formen der Repräsentation von Information mit Hilfe von Grafiken, Texten, Dateien, elektronischer Post und Hypertext. Außerdem vermitteln wir einen ersten Einblick in die Struktur informationsverarbeitender Prozesse anhand der Bausteine von Algorithmen. All das geht handlungsorientiert und spielerisch vor sich: Die Schüler erstellen und bearbeiten Dokumente, Dateibäume oder Hypertextstrukturen zu altersgemäßen Aufgabenstellungen und analysieren dann das Ergebnis ihrer Arbeit mit Hilfe informatischer Konzepte.

## 1.1 Datenstrukturen

Datenstrukturen von Standardsoftware wie Grafikprogrammen oder Textverarbeitungen stellen spezielle Formen der Repräsentation von Information dar, die bereits in der 6. Jahrgangsstufe bearbeitet werden können. Wir beginnen dabei mit Vektorgrafiken, da die beteiligten Objekte (Kreise, Rechtecke, Textbereiche etc.) hier leicht identifiziert und gegeneinander abgegrenzt werden können. Das Erkennen eines Textabsatzes als Objekt setzt dagegen einen weiteren gedanklichen Schritt voraus, der besser erst im Anschluss an eine erste Begegnung mit Objektstrukturen erfolgen sollte.

Der Altersstufe entsprechend (siehe Teil A, Abschnitt 1.3) muss der Informatikunterricht der 6. Jahrgangsstufe sehr anschaulich und schülerorientiert sein. Daraus ergibt sich zwangsläufig, dass die Schüler ausgiebig mit dem Rechner und seiner Software arbeiten. Hierbei stellt sich dem Lehrer immer wieder die Frage, wo sein Unterricht die Grenze zwischen informatischer Bildung und reiner Benutzer- oder Produktschulung (siehe Teil B, Abschnitt 1.1) überschreitet. Dies ist im Einzelfall oft nicht eindeutig zu beantworten. Wenn die in Teil B, Abschnitt 4.3 beschriebenen Prinzipien *Allgemeingültigkeit* und *Lebensdauer* ausreichende Beachtung finden, kann dennoch nicht allzu viel schief gehen. Eine detaillierte Darstellung des Lehrweges sowie zahlreiche Aufgaben kann man unserem Unterrichtswerk Frey et al. (2004) entnehmen.

### 1.1.1 Lernziele

Diese Sequenz hat sich u.a. die folgenden Ziele gesetzt: Die Schüler sollen in der Lage sein:

- die Objekte von Vektorgrafik- und Textverarbeitungsprogrammen zu identifizieren, typische Attribute zu erkennen und deren Werte festzustellen,
- den Unterschied zwischen Klassen und Instanzen zu beschreiben und Beispiele für beide Kategorien anzugeben,
- die Struktur typischer Klassen zu beschreiben,
- aus einer Liste von Objekten und Attributwerten ein Dokument zu erstellen,
- die Wirkung und Bezeichnung typischer Methoden der behandelten Objekte zu beschreiben und verschiedene Arten des Aufrufs zu nennen,
- Dokumente aus einer Sequenz von Methodenaufrufen und Wertzuweisungen zu konstruieren,
- wichtige Beziehungen zwischen Klassen wie „enthält“ oder „ist Teil von“ aufzustellen und zu interpretieren.

Diese Kenntnisse und Fertigkeiten sollen soweit irgend möglich anhand altersgemäßer Problemstellungen *während der praktischen Arbeit* am Rechner erworben werden. Für das Erreichen dieser Lernziele setzen wir etwa 16 Unterrichtsstunden zu je 45 Minuten an.

### 1.1.2 Notation

Zur Darstellung der Objektstrukturen verwenden wir einerseits Objekt- und Klassendiagramme, deren Syntax an die Altersgruppe angepasst wurde. Zusätzlich hat sich die Verwendung einer speziellen kompakten Beschreibungssprache bewährt, die den Kindern vor allem Schreibarbeit sparen soll. Der Zugriff auf die Attribute eines Objektes erfolgt dabei über die bekannte Punktnotation:

Tisch. Eckenrundung = 30.

Entgegen der in der objektorientierten Programmierung üblichen Identifikation von individuellen Instanzen über Referenzen (Zeiger) greifen wir auf die Objekte über *eindeutige Namen* zu. Jede Klasse erhält also ein Attribut „*Name*“, das oft nicht explizit aufgeführt wird. Wir setzen stillschweigend voraus, dass dieses Attribut bei zwei verschiedenen Instanzen einer Klasse auch mit verschiedenen Werten belegt wird (analog zu Schlüsselfeldern bei relationalen Datenbanken).

Falls es mehrere Möglichkeiten für die Benennung eines Begriffs gibt und sich keine durchschlagenden didaktischen Argumente zu Gunsten einer davon finden lassen, verwenden wir den mittlerweile sehr breit akzeptierten Standard der *Unified Modeling Language* (siehe Booch, Rumbaugh, Jacobson (1997)).

### 1.1.3 Software

Das folgende Unterrichtsbeispiel wurde mit dem Produkt *CorelDraw®* ausgeführt, das wie jede Software sowohl Vor- als auch Nachteile hat. So kann man damit einerseits mit Hilfe der rechten Maustaste (Eigenschaften im Kontextmenü) sehr schön die Attribute eines Objektes inspizieren, andererseits sind diese oft optisch auf *Menükarten* zu *Attributgruppen* zusammengefasst, wodurch die begriffliche Komplexität für die Schüler zunimmt. Nachdem diese Attributgruppen selbst wiederum als Objekte betrachtet werden können, verwenden wir auch hier eine Punktnotation, also

Tisch.Umriss.Breite

zur Bezeichnung des Attributes „Breite“ in der Gruppe „Umriss“. Alternativ kann natürlich jedes andere vektororientierte Zeichenprogramm verwendet werden.

Für die in 1.1.10 folgende Analyse der Datenstruktur von Textdokumenten benutzen wir *MS-Word®*, wobei die Bedeutung der verwendeten Software hier noch geringer ist als bei den oben genannten Grafikprogrammen.

### 1.1.4 Aufgabenstellung

Die Schüler sollen mit Hilfe eines Vektorgrafikprogrammes einen Plan der Einrichtung ihres Wohnzimmers (alternativ des Klassenzimmers o.ä.) erstellen.

Für einige Schüler könnte dies der erste Kontakt mit dem Rechner und einschlägiger Grafiksoftware sein, während sich andere vielleicht schon für versierte „Computerzeichner“ halten. Zur weiteren Arbeit benötigen jedoch alle Schüler (je nach Hard- und Softwareumgebung) die folgenden Kenntnisse bzw. Fertigkeiten:

- Einschalten und Hochfahren des Rechners,
- Anmelden mit Kennung und Passwort,
- Starten des Vektorgrafikprogrammes,
- Grundlegender Umgang mit der Maus (Mauspfeil positionieren, Auswählen durch Klicken, Markieren und „Ziehen“),
- Bedienung der Tastatur mit Auffinden der Buchstaben.

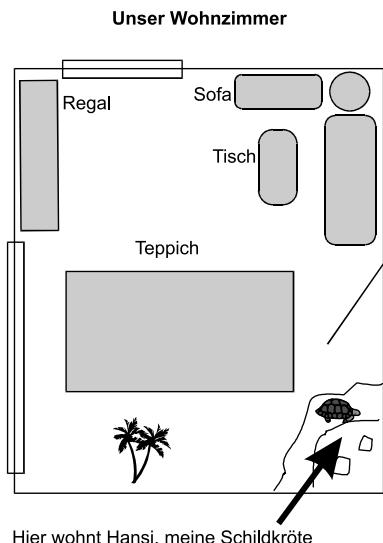
Eine Vermittlung dieser Kenntnisse ohne innere Differenzierung (siehe Teil A, Abschnitt 2.8) würde einen Teil der Schüler überfordern, während sich ein anderer Teil vielleicht langweilen würde. Wir setzen daher auf problembezogenen Einzelunterricht, indem wir einen *schriftlichen* Arbeitsauftrag ausgeben und damit alle Schüler zur Arbeit am Rechner anregen:

- Schalte den Rechner ein (Schalter hinten links).
- Warte, bis du zur Anmeldung aufgefordert wirst.
- Gib als Kennung „Schüler“ und als Passwort „schnecke1“ ein.
- Starte das Programm Corel Draw, indem du mit der linken Maustaste auf das Programmsymbol klickst.
- Wähle das Werkzeug „Rechteck“ und ziehe ein großes Rechteck auf.
- Zeige mit der Maus auf das Rechteck, drücke die rechte Maustaste.

- Zeichne und gestalte auf diese Weise alle Einrichtungsgegenstände des Wohnzimmers.

Den „Computerneulingen“ bringen wir während dieser allgemeinen Arbeitsphase mit Unterstützung bereits erfahrenerer Schüler die nötigen Grundfertigkeiten in Einzelberatung bei.

Um der Kreativität der Schülerinnen und Schüler freien Lauf zu lassen, sollte man ihnen auf keinen Fall eine Vorgabe für das Ergebnis machen. Die Lehrkraft muss sich dagegen im Klaren darüber sein, wie das Produkt am Ende der Sequenz aussehen könnte. Einen möglichen Erwartungshorizont dafür zeigt Abb. 1.1.



**Abb. 1.1.** Ein mögliches Ergebnis der Zeichenphase

### 1.1.5 Objekte, Klassen und Instanzen

Nach der Fertigstellung der ersten Zeichnung führen wir die neuen Begriffe ein:

Die Einzelteile einer Zeichnung (Tisch, Stuhl, Schrank) nennen wir *Objekte*. Wir stellen fest, dass wir mehrere verschiedene Arten von Objekten erschaffen können: Rechtecke, Ellipsen, Textbereiche etc. Dabei können mehrere Objekte eines Typs in der gleichen Zeichnung vorkommen: 2 Rechtecke, 3 Ellipsen, 4 Textbereiche etc. Diese Objektarten heißen *Klassen*. Jedes Objekt der Zeichnung ist eine *Instanz* genau einer Klasse. Für die Aussage, dass der Tisch eine Instanz der Klasse Rechteck ist, schreiben wir:

Tisch: RECHTECK.

„Rechteck“ bezeichnet hier übrigens eine Klasse von Objekten in Vektorgrafikdokumenten und nicht etwa Rechtecke im mathematischen Sinn, was man mit den Kindern auch ansprechen sollte. Wichtig ist dabei auch eine exakte Sprechweise im Zusammenhang mit dem Klassenbegriff: Eine Klasse ist keinesfalls eine *Menge* von Objekten sondern eher eine *Beschreibung* der Eigenschaften der Objekte dieser Menge. Darauf weist u.a. die Tatsache hin, dass „zwei“ leere Mengen nicht zu unterscheiden sind, zwei verschiedene Klassen jedoch auch dann, wenn es keine Objekte dieser Klassen gibt.

Die Tabelle 1.1. zeigt einige Beispiele für Klassen und Instanzen aus der Sicht der Schülerinnen und Schüler.

**Tabelle 1.1.** Klassen und Instanzen im Plan unseres Wohnzimmers

Klasse	(Name der) Instanz
Ellipse	runder Tisch
Rechteck	Couch
Linie	Tür
Textbereich	Titel
Handzeichnung	Schildkrötengehege
Grafikobjekt	Schildkröte

### 1.1.6 Attribute und Attributwerte

Um ein Problembewusstsein für den Begriff „Attribut“ zu erzeugen, regen wir die Schüler an, die Erscheinungsform bereits gezeichneter Objekte durch Manipulation der Werte ihrer Attribute zu verändern. Wir stellen uns zunächst die Frage, wodurch sich zwei verschiedene Instanzen einer Klasse unterscheiden. Wir vergleichen: Beim Objekt Couch findet sich etwa für das Attribut „Eckenrundung“ der Wert 74, während sich beim Objekt Regal an gleicher Stelle der Wert 0 findet. Ein näherer Vergleich der Attribute beider Objekte ergibt die Erkenntnis, dass deren Struktur völlig identisch ist. Lediglich die eingetragenen Werte unterscheiden sich. Daher legen wir fest:

Verschiedene Instanzen derselben Klasse haben dieselben *Attribute*. Allerdings können sich diese Attribute in ihren *Werten* unterscheiden. Diese Werte können u.a. Zahlen, Texte oder Bildsymbole sein. Wir schreiben kurz:

Couch.Eckenrundung = 74

für die Aussage: „Das Attribut Eckenrundung der Instanz Couch (der Klasse Rechteck) hat den Wert 74“.

Einige Beispiele für Attribute und Attributwerte des Objektes Couch finden sich in Tabelle 1.2.

**Tabelle 1.2.** Attribute und Attributwerte

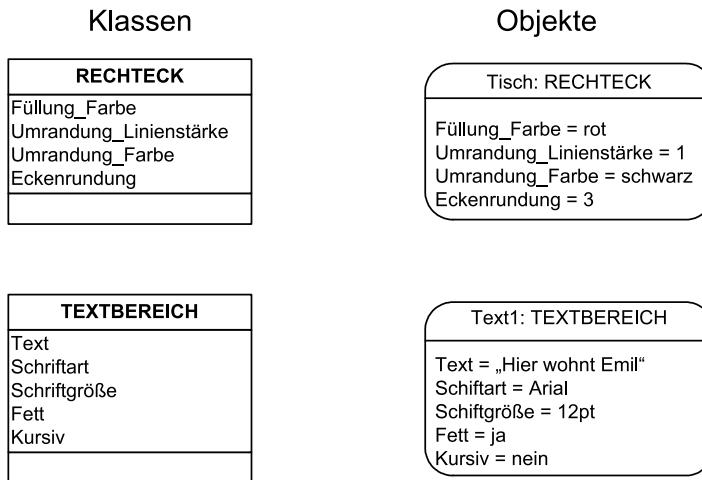
Attribut	Wert z.B.
Breite (Gruppe Umriss)	0,076mm
Füllung (Gruppe Füllung)	Stile/Recyclingpapier
Farbe (Gruppe Umriss)	Schwarz

### 1.1.7 Klassen und Attributstrukturen

Wodurch unterscheiden sich eigentlich die Instanzen verschiedener Klassen? Wir betrachten ein Rechteck und ein Textobjekt und stellen fest, dass die beiden Klassen einige gemeinsame, aber auch verschiedene Attribute aufweisen (siehe auch Abb. 1.2).

Die *Attributstruktur* (Anzahl, Gruppen und Namen) eines Objektes wird durch seine *Klasse* festgelegt. Instanzen gleicher Klassen haben dieselbe Attributstruktur, Instanzen verschiedener Klassen unterscheiden sich in ihrer Attributstruktur.

Abbildung 1.2 zeigt einen Vorschlag für eine grafische Darstellung aller bisher gelernten Begriffe am Beispiel der Klassen „Textbereich“ und „Rechteck“.

**Abb. 1.2.** Klassen und Objekte, Attribute und Attributwerte

### 1.1.8 Methoden und Botschaften

Bei der Arbeit an unserem Wohnzimmerplan haben wir festgestellt, dass sich die notwendigen Operationen mit Objekten nicht auf das Setzen von Attributwerten

beschränken lassen. Wir mussten Objekte *ausschneiden*, *kopieren*, *einfügen*, *verschieben* oder *drehen*. Interessant waren dabei die folgenden Beobachtungen:

1. Man kann Objekte *ausschneiden* (dann sind sie weg), aber danach auch wieder irgendwo einfügen (aus dem „Nichts“). Das funktioniert aber nur für das *zuletzt ausgeschnittene* Objekt. *Kopieren* funktioniert ähnlich, aber das Zielobjekt verbleibt dabei in der Zeichnung.
2. Ähnliche Objekte können zunächst durch *Kopieren* und anschließendes *Verändern* einzelner Attribute erzeugt werden. Die Klasse (und damit die Attributstruktur) ändert sich dabei nicht. Außerdem werden nach dem Kopieren nur die Attribute des markierten Objektes verändert, nicht etwa die der „Vorlagen“. Die Kopien erhalten also eine eigene Identität.

Wir betrachten Objekte ab jetzt als „handlungsfähige“ Gebilde und definieren:

Jede Klasse legt für ihre Instanzen typische *Operationen* fest, wie „sich markieren“, „sich ausschneiden“, „sich kopieren“, „sich einfügen“, „sich drehen“, „sich verschieben“. Diese Operationen (der Instanzen) heißen *Methoden*. Wir schreiben dafür z.B.:

Couch.aus schneiden()

und meinen die Methode „Ausschneiden“ des Objektes „Couch“ der Klasse „Rechteck“. Die Klammern dienen zur Unterscheidung von eventuell gleichnamigen Attributen.

Die Methoden (Operationen) der Objekte werden durch *Botschaften* aufgerufen. Diese Botschaften können Mausklicks, Menübefehle oder das Drücken eines Bildsymbols sein.

Einige Methoden erfordern weitere Hilfsmittel. So benötigt man für ausgeschnittene oder kopierte Objekte einen *Behälter* für die Aufbewahrung bis zur Wiederverwendung durch „Einfügen“ (*Zwischenablage*). Sein Inhalt kann unter manchen Betriebssystemen (z.B. *MS-Windows 98*) direkt betrachtet werden.

Außerdem gibt es Methoden, die eine oder mehrere Angaben benötigen, um ausgeführt werden zu können. Beim Drehen muss beispielsweise der Winkel angegeben werden. Diese Angaben heißen *Parameter* und werden zwischen den Klammern der Methode geschrieben. Wir schreiben z.B. für eine Drehung um 30° gegen den Uhrzeigersinn:

Couch.drehen(30).

Glücklicherweise müssen wir uns beim Zeichnen meist nicht um diese Zahlenangaben kümmern, da der Drehwinkel automatisch berechnet und weitergegeben wird.

Eine besondere Rolle spielen die Methoden, mit denen man die Werte von Attributen verändert kann. Nach dem folgenden Aufruf:

Couch.SetzeFüllfarbe(rot)

hat das Attribut „Füllfarbe“ des Objektes „Couch“ den Wert „rot“.

Bei der Verwendung von Methoden stellen wir außerdem fest, dass Programme oft für den Aufruf derselben Methode verschiedene Möglichkeiten zur Verfügung stellen (siehe Tabelle 1.3).

**Tabelle 1.3.** Beispiele für Methoden und Aufrufmöglichkeiten unter *MS-Word<sup>®</sup>*

Klasse	Methode	Aufrufe
Alle	Ausschneiden()	Strg-X, Symbol „Schere“, Menüfolge: <i>Bearbeiten-Ausschneiden</i>
Alle	Drehen(Winkel)	nochmaliges Klicken, Menüfolge: <i>Anordnen-Verändern-Drehen</i>
Textobjekt	Rechtschreibprüfen()	Menüfolge: <i>Text-Schreibhilfsmittel-Rechtschreibprüfung</i>

---

### 1.1.9 Übungsaufgaben und Lernzielkontrollen

Die folgende kleine Kollektion von Aufgaben soll lediglich Orientierungshilfen zu Ausrichtung und Tiefe des möglichen Prüfungsstoffes bieten.

1. Zeichne einen Lastwagen, der mindestens die folgenden Objekte enthält: Rad1, Rad2: ELLIPSE; Aufbau, Fahrerkabine: RECHTECK; Fahrer: FREIHANDLIE. Erzeuge die Räder dabei durch Kopieren.
2. Welche Objekte kommen in der vorliegenden Zeichnung vor (Klassen, wesentliche Attribute und -werte)?
3. Fertige mit einem Vektorgrafikprogramm eine Zeichnung von einem Haus an und nenne die Methoden, die du dabei ausführst.

### 1.1.10 Objektstruktur von Textverarbeitungssystemen

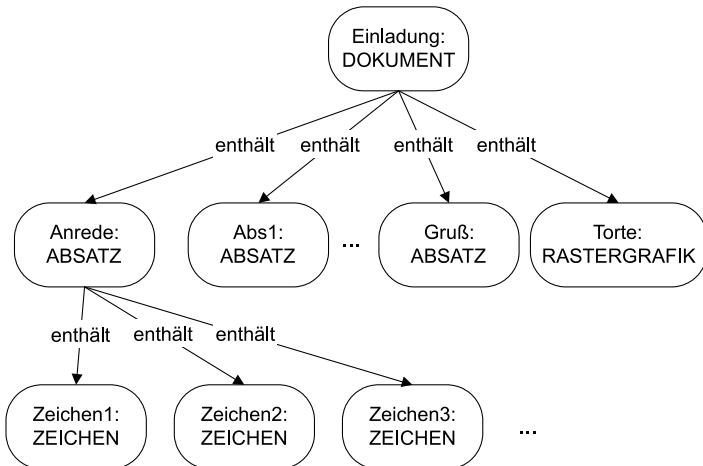
Nun übertragen wir die soeben erworbene (objektorientierte) Begriffswelt auf ein Textverarbeitungssystem. Dabei identifizieren wir z.B. die Klassen „Textdokument“, „Seite“, „Absatz“, „Zeichen“ mit den in Tabelle 1.4 aufgezählten Attributen.

**Tabelle 1.4.** Klassen und Attribute einer Textverarbeitung (*MS-Word<sup>®</sup>*)

Klasse	Attribute z.B.
Textdokument	Name, Änderungsdatum, Größe
Seite	Ränder, Papierformat, Kopf- Fußzeile
Absatz	Stil, Einzug, Zeilenabstand, Abstand vorher /nachher
Zeichen	Schriftart, Schriftschnitt, Schriftgrad, Farbe, Effekte

### 1.1.11 Beziehungen zwischen Objekten

Anhand der Datenstruktur von Texten können nun auch *Beziehungen* zwischen Objekten eingeführt werden, z.B.: „ist Teil von“, „enthält“. Wir verwenden dafür zunächst Objektdiagramme:



**Abb. 1.3.** Beziehungen zwischen Objekten

## 1.2 Dateien und Ordner

Im Rahmen der bisherigen Arbeit mit Vektorgrafik- und Textverarbeitungsprogrammen wurden bereits öfter Dokumente auf der Festplatte gespeichert bzw. von dieser geladen. Nun systematisieren wir diese Erkenntnisse und betrachten Dateien als *Container für ein Dokument*. Für *Container von Dateien* geben wir dem Begriff „Ordner“ gegenüber „Verzeichnis“ den Vorzug, und zwar aus folgenden Gründen:

- „Ordner“ suggeriert, dass die Dateien wie lose Blätter *aufgenommen* werden, im Gegensatz zum *Inhaltsverzeichnis*, das lediglich aus *Verweisen* besteht,
- ein „Ordner“ besteht nicht nur aus den eingehefpten losen Blättern, er besitzt darüber hinaus noch (beschriftete) Deckel und Heftvorrichtungen, so wie ein Dateiverzeichnis einen Namen und andere Attribute besitzt,
- physikalische Ordner können selbst wieder eingeordnet werden, bei Verzeichnissen erwartet man das weniger,
- im Bereich von Bürossoftware ist „Ordner“ gebräuchlicher.

### 1.2.1 Lernziele

Die Schüler sollen nach der Unterrichtssequenz in der Lage sein:

- Dateien als Container für Dokumente und
- Ordner als Container (für weitere Ordner und/oder Dateien) zu verwenden,
- die typischen Attribute dieser Container zu beschreiben,
- Ordnerstrukturen nach vorgegebenen logischen Gesichtspunkten aufzubauen,
- Dateien mit bestimmten Attributwerten in Verzeichnisstrukturen aufzufinden,
- eine Menge von Dateien in einem Ordner mit Hilfe von Unterordnern zu strukturieren.

### 1.2.2 Dateien und Dokumente

Eine Datei *enthält ein Dokument*, deshalb sind die beiden Begriffe nicht gleichwertig (auch wenn eine Datei oft denselben Namen wie das enthaltene Dokument trägt). Anhand einer schreibgeschützten Datei kann dieser Unterschied zwischen Dokument und Datei gut veranschaulicht werden: Lädt man das in der Datei enthaltene Dokument in ein Anwendungsprogramm, so kann man es problemlos ändern und unter einem anderen Namen (in einer neuen Datei) speichern, aber nicht die ursprüngliche Datei überschreiben. Ebenso deutlich wird der Unterschied beim Anlegen eines neuen Dokumentes (meist automatisch mit dem Namen „Dokument1“ versehen): Bevor es erstmals auf der Festplatte gespeichert wird, existiert das Dokument bereits und kann editiert werden, obwohl es noch keine Datei dazu gibt. Ein Systemabsturz zu diesem Zeitpunkt resultiert daher in einem Totalverlust der bisherigen Arbeit an diesem Dokument.

Auch Dateien haben Attribute, die (auf MS-Windows® bezogen) in Tabelle 1.5. beschrieben werden.

**Tabelle 1.5.** Attribute und Attributwerte von Dateien

Attribut	Bedeutung	Werte z.B.
Name	Name der Datei	Mein Brief.doc
Größe	Umfang in Bytes	3kB, 30MB
Änderungsdatum	Datum des letzten Schreibzugriffs	30.11.99
Schreibgeschützt	Kann verändert werden?	ja/nein
Versteckt	Anzuzeigen?	ja/nein
System	Teil des Betriebssystems?	ja/nein
Archiv	Seit der letzten Datensicherung geändert?	ja/nein

---

### 1.2.3 Ordnerstrukturen

Nach der Produktion einer Reihe verschiedener Dateien, die bisher alle in demselben Ordner abgelegt wurden, stellen die Schüler ein zunehmendes Chaos in diesem Ordner fest. Es wird immer schwieriger, eine bestimmte Datei aufzufinden. Als Lösung bietet sich an, im benutzten Ordner neue *Unterordner* anzulegen und die Dateien darin einzurichten. Dazu könnte man die folgende Lernaufgabe stellen:

Wir wollen Informationen über den Kontinent Amerika strukturieren. Dazu legen wir einen Ordnerbaum an, der dessen geographische Struktur (Nord- Süd- Mittelamerika) und darunter die zugehörigen Länder widerspiegelt (Brasilien, Chile, Argentinien unter Südamerika). Dann sammeln wir in Gruppenarbeit Informationen über die einzelnen Länder aus Büchern, CDROMS oder dem Internet und legen sie in Form geeigneter Dokumente in dem jeweiligen Ordner ab.

Zusätzlich wollen wir versuchen, für jedes Land ein Textdokument mit Informationen über Staatsoberhaupt, Staatsform, Hauptstadt und Einwohnerzahl, ein Bilddokument mit der Flagge, ein Bilddokument mit der Landkarte, ein Tondokument mit der Nationalhymne einbauen. Außerdem werden in den Ordner der mittleren Ebene Landkarten und allgemeine Informationen abgelegt.

Nach Abschluss des Projektes (siehe auch Abb. 1.4) fassen wir unsere Erkenntnisse zusammen:

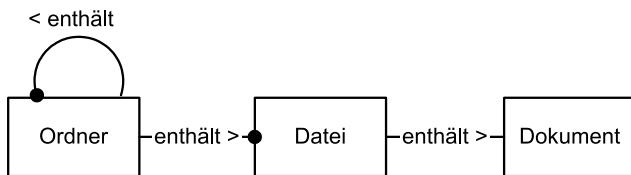
Dokumente können mit Hilfe von *Ordner* verwaltet werden. Solche Ordner können

1. *Dokumente* und/oder
2. *andere Ordner* (Unterordner) enthalten.

Ordner bilden *Ordnerbäume*:

1. Es gibt einen Ordner, in dem alle anderen enthalten sind (*Wurzel*).
2. Jeder Ordner (bis auf die Wurzel) hat genau einen Ordner, in dem er enthalten ist („Überordner“).

Solche Beziehungen zwischen den Objekten bestimmter Klassen können sehr kompakt in *Klassendiagrammen* dargestellt werden (siehe Abb. 1.4). Wir verwenden dafür eine der UML-Notation (siehe Booch , Rumbaugh, Jacobson (1997)) ähnliche Syntax, die wir jedoch etwas vereinfacht haben. Insbesondere beschreiben wir Kardinalitäten größer als eins durch einen fetten Punkt am Ende der Beziehungslinie, etwa für: „Ein Absatz kann mehr als ein Zeichen enthalten“. Man beachte auch, dass die Beziehungen grundsätzlich ungerichtet sind, wogegen ihre Bezeichner meist nur in einer Richtung Sinn ergeben, die deshalb auch dort einge tragen wird.



**Abb. 1.4.** Klassenmodell von Ordnern, Dateien und Dokumenten

Bei der Interpretation solcher Klassendiagramme sollte besonders darauf geachtet werden, dass die Schülerinnen und Schüler die Beziehungen *nicht* fälschlicherweise als direkte Beziehungen zwischen *Klassen* („die Klasse Objekt enthält die Klasse Datei“), sondern als Beziehung zwischen *Objekten* der jeweiligen Klassen („Objekte der Klasse Ordner enthalten Objekte der Klasse Datei“) auffassen.

### 1.2.4 Methoden

Für Dateien und Ordner verwenden wir ähnliche Methoden wie für Texte: Markieren, Löschen, Kopieren, Einfügen, Verschieben. Zusätzlich kann man Ordner und Dateien öffnen, allerdings mit einem auf den ersten Blick unterschiedlichen Resultat: Beim Öffnen einer Datei wird diese u.U. in ein (festgelegtes) Anwendungsprogramm geladen, beim Öffnen eines Ordners wird sein Inhalt in Form einer Liste angezeigt. Bei näherer Betrachtung stellt sich heraus, dass auch das Anzeigen des Ordnerinhaltes über ein spezielles Anwendungsprogramm (z.B. *MS-Explorer*) vor sich geht.

## 1.3 Versand von Dokumenten

Technische Details, Anglizismen und Abkürzungen sollten (nicht nur hier) soweit möglich vermieden werden. Es geht keineswegs darum, den Schülern die Funktionsweise des Internet zu erklären, sondern sie behutsam in einige grundlegende Prinzipien von Rechnernetzen einzuführen. Dem didaktischen Prinzip der *Reduktion* kommt in diesem Themenbereich besondere Bedeutung zu.

Der Begriff *E-Mail* (Schreibweise laut „Duden – Die neue Rechtschreibung“) ist leider zweideutig: Einerseits wird damit das gesamte *Übermittlungskonzept* mitsamt seiner technischen Realisierung (Protokolle, Datenformat, etc.) verstanden, andererseits kann auch nur eine einzelne *Nachricht* gemeint sein. Wir achten deshalb darauf, diesen Begriff nur in eindeutiger Weise zu verwenden.

### 1.3.1 Lernziele und Zeitrahmen

Diese Lektion hat sich u.a. die folgenden Ziele gesetzt: Die Schüler sollen in die Lage versetzt werden:

- elektronische Nachrichten mit Anhängen zu verfassen, zu versenden und zu empfangen,
- den Weg einer Nachricht (im Wesentlichen) zu beschreiben,
- den Unterschied zwischen automatischer Zustellung und Abruf zu kennzeichnen,
- die wesentlichen Attribute einer Nachricht zu charakterisieren,
- die Struktur einer E-Mail-Adresse und die Bedeutung ihrer Einzelteile wiederzugeben.

Für die gesamte Lernsequenz sind etwa 7 Unterrichtsstunden zu je 45 Minuten anzusetzen.

### **1.3.2 Systemanforderungen**

Die Unterrichtssequenz ist mit jeder Netzwerkstruktur und Software durchführbar, solange sich eine Möglichkeit zum Versand und Empfang elektronischer Nachrichten bietet. Allerdings wäre es sehr günstig für die Übungsphasen, wenn die Schüler die Gelegenheit hätten, sich gegenseitig Nachrichten zukommen zu lassen. Sehr förderlich wäre auch der Zugriff auf eine externe Zieladresse, über die Lehrkräfte auf die Nachrichten der Schüler antworten könnte.

### **1.3.3 Aufgabenstellung**

Wir wollen eine Blitzumfrage veranstalten, in der geklärt werden soll, wer aus der Klasse zu Hause Zugang zu einem Computer hat. Das Ergebnis soll dann per E-Mail an alle Schüler verschickt werden, erst als Text, später mit grafischen Darstellungen (Tortendiagramme). Alternativ wäre ein Wörterratespiel oder eine Fortsetzungsgeschichte denkbar.

### **1.3.4 Erste Schritte mit dem System**

Die Schüler können inzwischen soweit mit dem Rechner umgehen, dass sie sich anmelden, Programme starten und einfache Grafiken und Texte verfassen können. Deshalb dürfte der Start und die Bedienung des E-Mail-Systems keine großen Probleme verursachen. Das Spiel beginnt mit dem Versand eines *Umfrageformulars* an alle Teilnehmer.

*Betreff:* Formular für die Umfrage  
*Datum:* Tue, 28 Sep 1999 13:43:00 +0200  
*Von:* "Peter Hubwieser" <hubwiese@hypercons.bnro.de>  
*Rückantwort:* "Peter Hubwieser" <Peter.Hubwieser@in.tum.de>  
*Firma:* Bürgernetz Rosenheim  
*An:* <pc1@gymaib.bnro.de>

Liebe(r) Schüler(in),

hier ist das Formular für unsere Umfrage. Falls eine der Vorgaben zutrifft, mache bitte davor ein X. Falls keine Vorgaben vorhanden sind, trage bitte die verlangten Angaben ein.

- Wir haben zu Hause Computer:                    ja            nein,  
und zwar:    \_\_\_\_\_ Stück.
- Ich habe Zugang zu einem dieser Computer:  ja            nein.
- Der Computer gehört:    mir            meinen Eltern            einem meiner Geschwister.
- Es handelt sich um einen:  PC            Mac            anderen Typ.
- Ich mache darauf vor allem:                         \_\_\_\_\_,  
aber gelegentlich auch:                                 \_\_\_\_\_.

So das war's für diesmal. Lege bitte mit der Methode „Neu“ eine neue Nachricht an, kopiere den Absender dieser Nachricht in das Feld „An“, den Empfänger dieser Nachricht in das Feld „Von“, und gib als „Betreff“ an: „Antwort von“ gefolgt von deinem Namen. Vielen Dank. Bis bald, dein Informatiklehrer.

Falls sich mehrere Schüler einen Arbeitsplatz teilen, müssen entsprechend viele Formulare an die Arbeitsplätze verschickt werden. Die Schüler werden dann aufgefordert, die Methode „Abrufen“ anzuwenden, deren Bedeutung erst später genauer erklärt wird.

Anhand dieser Nachricht werden die Attribute und möglichen Werte eines Objektes der Klasse „Nachricht“ erklärt (siehe Tabelle 1.6).

**Tabelle 1.6.** Attribute von elektronischen Nachrichten

Attribut	Bedeutung	Wert z.B.
Von	Absenderadresse	Peter.Hubwieser@in.tum.de
An	Empfängeradresse	pc1@gymaib.bnro.de
CC (carbon copy)	Empfängeradresse für Kopien	pc2@gymaib.bnro.de, pc4@gymaib.bnro.de ...
Betreff	Thema der Nachricht	Formular für die Umfrage
Nachrichtentext	Eigentlicher Inhalt der Nachricht	Liebe Schüler...

Jetzt ist es an der Zeit für die Schüler, selbst aktiv zu werden. Sie füllen das verlangte Formular aus und schicken es an den Lehrer zurück. Aus didaktischen Gründen empfiehlt es sich, das zuerst durch *Erstellen* einer neuen Nachricht und Kopieren von Absenderadresse und Textvorgabe zu erledigen. Beim nächsten Mal kann man dann die Vereinfachung über die Methode „Antworten“ nutzen.

Die Lehrkraft sammelt jetzt alle Antworten, wertet die Ergebnisse aus und schickt diese als Textdatei an die Schüler zurück.

### 1.3.5 Der Weg einer elektronischen Nachricht

Natürlich ergeben sich beim Arbeiten mit dem E-Mail-System zahlreiche Fragen nach der Funktionsweise des Transports, z.B.:

- Was soll die ominöse Operation „Abrufen“ genau bedeuten?
- Wieso muss man Nachrichten abrufen, wenn der Absender sie doch abschickt? Bei Postbriefen und Telefonanrufen muss der Empfänger ja auch nicht aktiv werden.
- Oder doch? Man denke an das Ausleeren des Briefkastens oder das Abheben des Telefonhörers beim Klingeln des Telefons.

Auf jeden Fall liegt Klärungsbedarf vor. Ein Blick auf den Quelltext einer Nachricht zeigt, dass der Transport einer E-Mail nicht so einfach ist, wie er auf den ersten Blick aussieht (siehe Abb.1.5).

```
Received: from spooler by hypercons.bnro.de (Mercury/32 v2.01); 28 Sep 99  
13:47:56 +0100  
Return-path: <hubwiese@hypercons.bnro.de>  
Received: from pchub1 (194.95.220.16) by hypercons.bnro.de (Mercury/32 v2.01);  
28 Sep 99 13:47:47 +0100  
Message-ID: <004401bf09a6$a3715d40$0101a8c0@kolbermoor.bnro.de>  
Reply-To: "Peter Hubwieser" <Peter.Hubwieser@in.tum.de>  
From: "Peter Hubwieser" <hubwiese@hypercons.bnro.de>  
To: <pc1@gymaib.bnro.de>  
Subject: =?Windows-1252?Q?Formular_f=FCr_die_Umfrage?=br/>  
Date: Tue, 28 Sep 1999 13:43:00 +0200  
Organization: =?Windows-1252?Q?B=FCrgernetz_Rosenheim?=br/>  
MIME-Version: 1.0  
Content-Type: text/plain; charset="Windows-1252"  
Content-Transfer-Encoding: 8bit  
X-Priority: 3  
X-MSMail-Priority: Normal  
X-Mailer: Microsoft Outlook Express 5.00.2014.211  
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2014.211  
X-Mozilla-Status: 8001  
Liebe(r) Schüler(in) ...
```

**Abb. 1.5.** Quelltext einer elektronischen Nachricht

Natürlich erheben wir nicht den Anspruch, alle Komponenten dieser Nachricht erklären zu wollen. Es soll nur kurz der Weg einer Nachricht vom Sender zum Empfänger durch das Internet skizziert werden (siehe Abb. 1.6).

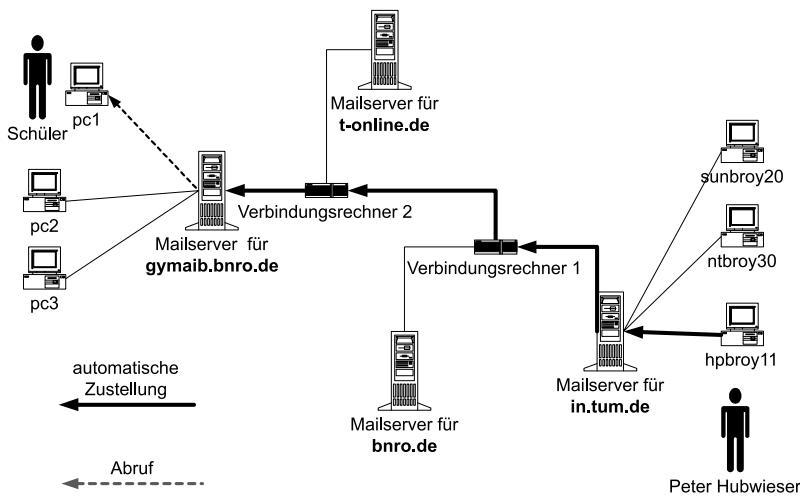


Abb. 1.6. Der Weg einer elektronischen Nachricht durch ein Rechnernetz

Dabei spielen zwei Arten von Rechnern eine besondere Rolle:

So genannte *Mailserver* sind in der Lage, Nachrichten ins Internet weiterzuleiten oder Nachrichten für mehrere Benutzer bis zur Abholung zwischenzuspeichern. Spezielle *Verbindungsrechner* (engl. „Router“) übernehmen die Rolle von Weichen bei der Eisenbahn: Sie entscheiden an Kreuzungspunkten, welchen Weg eine Nachricht auf ihrem Weg durch das Internet einschlagen soll.

Der Transport einer Nachricht findet dabei auf zwei verschiedene Arten statt:

1. Vom Rechner des Absenders über dessen zuständigen Mailserver bis zu dem Mailserver, der für den Empfänger zuständig ist, wird die Nachricht *automatisch weitergeleitet*. Falls dabei etwas schief geht, erhält der Absender eine Fehlermeldung.
2. Auf dem Mailserver des Empfängers wird die Nachricht solange zwischengespeichert, bis sie der Empfänger von einem dazu geeigneten Rechner aus *abruft*. Dabei hat er die Wahl, ob eine Kopie der Nachricht auf dem Server verbleibt oder nicht.

### 1.3.6 Das Format der Adressen

Nachdem wir nun den Weg einer Nachricht kennen, können wir die Struktur der E-Mail-Adressen untersuchen. Wie bei der Briefpost muss auch bei einer elektronischen Nachricht eine *Empfängeradresse* angegeben werden. Die *Absenderadresse* wird von der Software (meist) automatisch eingefügt.

Eine E-Mail-Adresse besteht aus zwei Teilen, die durch das Zeichen @ für das englische „at“ (deutsch „bei“) getrennt werden:

Peter.Hubwieser@in.tum.de.

Der erste Teil (links von @) bezeichnet einen *Benutzer*. Damit kann aber auch ein „Standardbenutzer“ eines bestimmten Rechners gemeint sein („Schüler“ oder „pc1“). Der zweite Teil (rechts von @) steht für den *Mailserver*, zu dem die Nachricht transportiert werden soll. Eine wichtige Rolle spielen dabei die so genannten *Domänen*. Das sind Bezeichnungen für Teilnetze des Internet, etwa die Rechner der Fakultät für Informatik der Technischen Universität München (in.tum.de), das Netzwerk des Gymnasiums Bad Aibling (gymaib.bnro.de) oder die Rechner der *Deutschen Telekom*® (t-online.de). Genau genommen müsste der Eintrag für den Mailserver des Gymnasiums Bad Aibling allerdings so aussehen:

mailserv.gymaib.bnro.de.

Da es jedoch für jede Domäne in der Regel genau einen zuständigen Mailserver gibt, genügt es in den meisten Fällen, nur den Namen der Domäne hinter dem @ anzugeben, also

pc1@gymaib.bnro.de oder Peter.Hubwieser@in.tum.de.

### 1.3.7 Anhängen von Anlagen

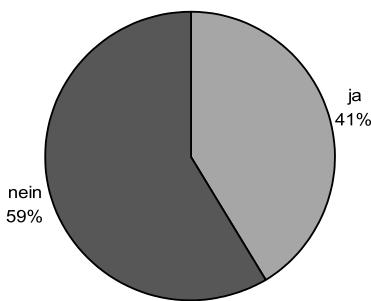
Nun wollen wir unser Projekt weiterführen, indem wir das tabellarische Ergebnis der Umfrage in eine *Präsentationsgrafik* umwandeln und die Ergebnisse als *Anhang* (Methode „Anfügen“) verschicken. Ein nützlicher Nebeneffekt ist dabei die (u.U.) erste Begegnung mit einer Tabellenkalkulation. Falls genügend Zeit vorhanden ist, könnte der Lehrer die Erstellung der Diagramme gruppenweise in Auftrag geben. Die Ergebnisse der Gruppen werden dann wiederum per E-Mail an den Lehrer zurückgeschickt (siehe Abb. 1.6), der sie zu einem gemeinsamen Abschlussdokument vereinigt. So könnte aus der folgenden einfachen Tabelle 1.7 mit Hilfe der Präsentationsfunktionen eines Tabellenkalkulationsprogramms das in Abb. 1.7 dargestellte Diagramm werden.

**Tabelle 1.7.** Teilergebnis der Umfrage

Wir haben zu Hause einen Computer:	
ja	nein
12	17

Abschließend stellen wir fest:

Mit Hilfe von *Anhängen* („Attachments“) kann man beliebige Dateien per E-Mail transportieren. Allerdings muss darauf geachtet werden, dass die Anhänge *nicht zu groß* sind, um die Wartezeiten (und damit die Kosten) für die Übertragung der Nachrichten in Grenzen zu halten.



**Abb. 1.7.** Tortendiagramm als Ergebnis der Umfrage

### 1.3.8 Aufgaben

1. Schreibe und verschicke eine E-Mail an ... mit dem Thema ..., dem Text ... und dem Anhang ...
2. Beschreibe den Weg einer Nachricht von müller@dom.de zu meier@haus.de.
3. Frau Hugenbauer hat ein E-Mail-Konto bei der Internet-Firma mailbox.de. Sie will ihre Nachrichten sowohl von ihrem Büro wie auch von zu Hause aus abrufen können. Dabei ist ihr besonders wichtig, dass alle Nachrichten auf dem Rechner in ihrem Büro vorliegen. Wie kann Frau Hugenbauer dieses Problem lösen?
4. Zähle alle wichtigen Attribute eines Objektes der Klasse „Nachricht“ auf, beschreibe ihr Bedeutung und nenne jeweils einige mögliche Werte.
5. Beschreibe die Struktur einer E-Mail-Adresse und die Bedeutung der einzelnen Elemente.

## 1.4 Hypertext

Nachdem wir bisher die Erstellung und Verwaltung von Dokumenten sowie deren Versand mit Hilfe elektronischer Post behandelt haben, wollen wir uns nun der Vernetzung von Dokumenten mit Hilfe des *Hypertextkonzeptes* (am Beispiel der *Hypertext Markup Language HTML*) zuwenden. Dabei sollen die Schüler vor allem eine Möglichkeit zur Strukturierung von Informationen kennen lernen, weniger die technischen Eigenheiten des verwendeten Systems.

### 1.4.1 Lernziele

Die Schüler sollen nach dieser Lektion in der Lage sein:

- die wesentlichen Operationen und den zeitlichen Verlauf des Abrufs einer Webseite zu beschreiben,
- die Aufgaben eines Webbrowsers und eines Webservers anzugeben,

- Hypertextdokumente mit Bildern und Verweisen zu erstellen,
- die Komponenten eines URL anzugeben und zu beschreiben,
- Informationsstrukturen in Hypertextstrukturen abzubilden.

### 1.4.2 Die Aufgabenstellung

Auch hier stellen wir uns wieder eine Aufgabe, die den Einsatz der zu erlernenden Techniken nahelegt:

Es soll ein *Klassenalbum* entwickelt werden, wobei jeder Schüler (nach dem Vorbild der in jedem Schreibwarengeschäft erhältlichen Vorlagen) zum Thema „das ist meine Schulkasse“ je ein Dokument mit seinen persönlichen Daten und Vorlieben erstellen soll. Dabei ergeben sich zahlreiche Anlässe zum Einbinden von Fotos oder Verweisen auf andere Dokumente.

Zunächst erstellt jeder Schüler einen ersten Entwurf für sein persönliches Dokument und speichert diesen in einer Datei mit seinem Namen ab. Anfangs handelt es sich dabei nur um ein Textdokument mit dem kleinen Unterschied, dass wir einen speziellen Editor für Webseiten verwenden. Dann fügen wir ein Passfoto des Schülers ein, das sich zunächst nur als Instanz einer neuen Klasse „Bild“ darstellt. Eigentlich interessant wird es allerdings erst, wenn der erste Verweis eingebaut wird.

### 1.4.3 Verweise auf andere Dokumente

Einleitend betrachten wir dazu einige Webseiten, die den Erfahrungsbereich der Schüler berühren, beispielsweise die Webdarstellung der eigenen Schule. Beim Erforschen dieser Struktur wird den Schülern schnell das Funktionsprinzip der Verweise klar. Anstatt lange Erklärungen abzugeben, bauen wir gleich selbst Verweise in unsere Dokumente ein. Zuerst erstellen wir als Ausgangspunkt für die ersten Verweise eine gemeinsame Liste aller beteiligten Schülerinnen und Schüler, in der jeder Name mit einem Verweis auf das persönliche Dokument des jeweiligen Schülers hinterlegt wird. Anhand dieses Beispiels definieren wir eine neue Klasse „Verweis“ mit dem einzigen Attribut „Zieladresse“.

Nun sind die Schüler in der Lage, auch in ihre persönlichen Dokumente Verweise aufzunehmen. Dafür bieten sich zahlreiche Anlässe, beispielsweise unter „mein(e) beste(r) Freund(in) in der Klasse“ oder „mein Lieblingstier“ oder „meine liebste Sportart“, wobei die Zieladressen sowohl im lokalen Netzwerk als auch außerhalb der Schule im Internet liegen können (siehe Abb. 1.8).

Das bin ich:		
<i>Passfoto</i>	Name:	Meier
	Vorname:	Anna-Katharina
	Geburtsdatum:	12.7.88
	Augenfarbe:	braun
Mein(e) beste(r) Freund(in) in der Klasse:	<u>Dorothea Huber</u>	
Mein Lieblingstier:	<u>Pferd</u>	
Mein Lieblingsessen:	<u>Spaghetti</u>	
Meine Lieblingsmusik:	<u>Mambo Nr. 5</u>	

**Abb. 1.8.** Muster für das persönliche Dokument einer Schülerin mit eingebauten Verweisen

#### 1.4.4 Datenwege

Schließlich fassen wir unsere Erkenntnisse über die technische Funktionsweise des *World Wide Web* (WWW) zusammen:

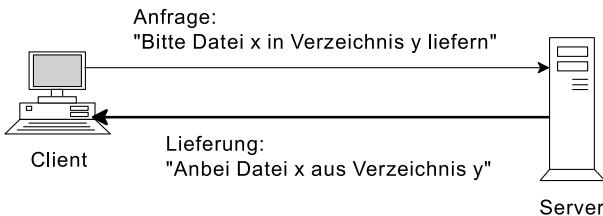
Im WWW gibt es (ebenso wie bei der elektronischen Post) spezielle Rechner (*Web-server*), deren Aufgabe es ist, Webseiten anzubieten. Wir fragen von unserem Rechner mit Hilfe eines speziellen Programms (*Webbrowser*) unter Angabe einer Adresse in einem besonderen Format (URL für *Uniform Resource Locator*, siehe unten) beim entsprechenden Webserver an, ob er die gewünschte Seite auf Lager hat. Diese Adresse kann dabei

- mit der Hand in ein dafür vorgesehenes Feld eingegeben,
- aus Adressenlisten (z.B. Favoriten) entnommen,
- oder einfach nur durch Anklicken eines Verweisobjektes übergeben werden.

Falls die Seite lieferbar ist, schickt der Webserver die entsprechenden Daten an unseren Rechner. Hier handelt es sich also um ein Beispiel für das Client-Server Prinzip (siehe Abb. 1.9). Das Format der Adressen (URL-Format) folgt dem Schema:

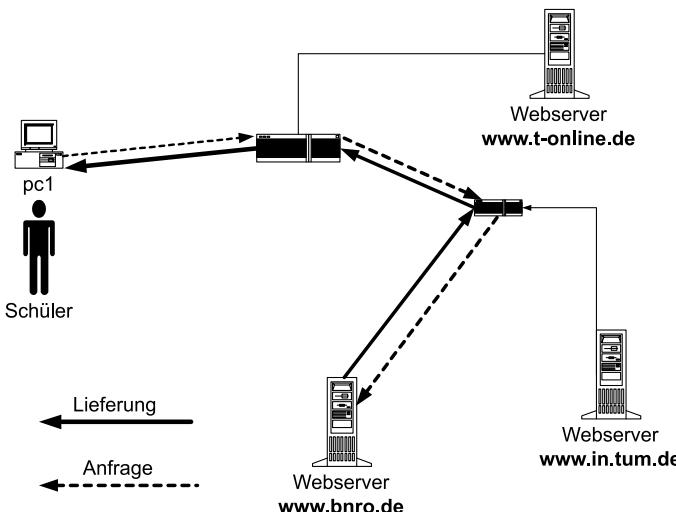
Protokoll://Webserver/Verzeichnis/Datei,

z.B. <http://www.bnro.de/schulen/gymaib01/index.html>. Falls die URL rechts ohne Angabe einer Datei endet, setzt der Webserver einen Standardnamen ein (meist index.html oder homepage.html).



**Abb. 1.9.** Das Client-Server-Prinzip

Allerdings ist der Weg von Anfrage und Lieferung meist nicht ganz so einfach. In Wirklichkeit sind bei der Abwicklung der Transaktion noch zahlreiche weitere Verbindungsrechner beteiligt (siehe Abb. 1.10).



**Abb. 1.10.** Datenwege beim Abrufen von Webseiten

Natürlich gibt es hier zahlreiche Fehlerquellen: Wir erkunden, was geschieht, wenn man die Adresse eines nicht vorhandenen Webserver oder einer nicht vorhandenen Seite eingibt.

#### 1.4.5 Datenschutzaspekte

Besonders wichtig bei diesem Projekt ist die Beachtung des Datenschutzes, was mit den Schülern ausführlich anhand folgender Fragen besprochen wird:

Was geschieht, wenn wir unsere Seiten öffentlich ausstellen würden? Welche kommerzielle Nutzungsmöglichkeiten bieten sich durch diese Informationen?

## 1.5 Verarbeitung von Information

Zum Abschluss des Kurses wollen wir einen Blick auf grundlegende Techniken zur Verarbeitung von Information werfen. Genau genommen geht es um Repräsentationen von Informationen über Verarbeitungsvorgänge. Die Schüler sollen dabei anhand der Beschreibung von einfachen Verarbeitungsvorgängen die Strukturelemente von Algorithmen (Sequenz, Auswahl und Wiederholung) kennen lernen. Dabei geben wir uns mit einer beliebigen Form der Wiederholung zufrieden (je nach verwendetem System). Ob wir mit einer festen Anzahl von Wiederholungen arbeiten, oder volle Berechenbarkeit (über Wiederholung mit Anfangs- oder Endbedingung) erreichen, ist auf dieser Stufe nicht relevant.

### 1.5.1 Software

Für unser Vorhaben eignet sich auf *keinen Fall* eine herkömmliche Programmiersprache (wie *Pascal*, *C* oder *Java*). Eine dem Alter der Schüler angemessene Programmierschnittstelle bietet dagegen z.B. der Roboter Karel, den man mit einfachen, intuitiv verständlichen Befehlsfolgen programmieren kann (siehe auch Pattis (1981), Nievergelt (1999)). Die folgende Befehlssequenz veranlasst Karel beispielsweise, ein Quadrat aus Ziegelsteinen zu legen (vorausgesetzt, dass er dafür genug Platz zur Verfügung hat):

```
Befehl QUADRAT
Beginn
    wiederhole 4 mal
        HINLEGEN
        SCHRITT
        LINKSUM
        *wiederhole
Ende
```

Wenn Karel an einer Wand angelangt ist, kann er mit Hilfe von bedingten Anweisungen entsprechend reagieren, z.B.:

```
Wenn IST WAND
    dann LINKSUM
    sonst SCHRITT
```

Wir ziehen es hier jedoch vor, die Erstellung von Serienbriefen mit einem geeigneten Textverarbeitungsprogramm (z.B. *MS-Word 97®*) zum Anlass unserer Be trachtungen über Informationsverarbeitung zu nehmen, was einige Vorteile bietet:

- Die Schüler sind mit der Bedienung des Systems bereits vertraut.
- Wir bereiten die Begriffe „Datensatz“ und „Datenfeld“ für die spätere Datenmodellierung vor.
- Die erlernten Fertigkeiten lassen sich im Alltag einsetzen.

## 1.5.2 Lernziele

Nach Abschluss der Sequenz sollen die Schüler in der Lage sein:

- einfache Abläufe algorithmisch zu beschreiben,
- einfache Algorithmen aus elementaren Befehlen und den Strukturelementen (Sequenz, bedingte Anweisung, Wiederholung) aufzubauen und
- auf dem gewählten Beispielsystem umsetzen zu können.

## 1.5.3 Aufgabenstellung

Wir wollen eine Geburtstageinladung verfassen und für alle Klassenkameraden ausdrucken. Dabei soll die Anrede „Liebe(r)“ davon abhängen, ob es sich um ein Mädchen oder einen Jungen handelt.

## 1.5.4 Umsetzung

Als Einstieg entwerfen wird unsere Einladung als Text mit Hilfe des gewohnten Textverarbeitungssystems. Dann erstellen wir ein Steuerdokument mit den Datensätzen für die Empfänger und Adressen (siehe Tabelle 1.8). Dabei verwenden wir die Begriffe „Datensatz“ und „Datenfeld“ im Sinne von „Spalte“ und „Zeile“ einer Tabelle.

**Tabelle 1.8.** Steuerdatensätze für Serienbriefe

Anrede	Vorname	Name	Adresse1	PLZ	Ort
Herr	Hans	Müller	Reitweg 18	83555	Schnuckelstadt
Frau	Hanna	Schmid	Sumpfweg 3	98555	Tauchburg

«Anrede» «Vorname» «Name»  
«Adresse1»  
«Postleitzahl»  
«Ort»

Lieber «Vorname»,

Hiermit möchte ich dich ganz herzlich zu meinem Geburtstagsfest am 12.7.2000 in die Truthahnstr. 17 einladen.

Das Fest beginnt um 14:30 und endet um 17:30.

Bis dann, viele Grüße, Anna.

**Abb. 1.11.** Die fertige Einladung als Serienbrief

Anschließend fügen wir die Felder für die Namen und Adressen der Klassenkameraden in unsere Geburtstagseinladung ein. Beim Einbau der Anrede stoßen wir auf die Frage, wie wir feststellen können, ob es sich um ein Mädchen oder einen Jungen handelt. Die Lösung liegt in Verwendung eines bedingten Terms, wie er beispielsweise in *Microsoft Word*® angeboten wird:

Wenn Anrede = „Frau“ dann „Liebe“ sonst „Lieber“.

Unser Serienbrief könnte dann so aussehen, wie in Abb. 1.11. dargestellt.  
Nun versuchen wir zu beschreiben, was bei der Ausgabe der Serienbriefe an alle Klassenkameraden vor sich geht. Das Ergebnis wird in Umgangssprache formuliert:

Wiederhole für alle Datensätze

Schreibe Inhalt des Feldes Anrede  
Schreibe Inhalt des Feldes Vorname  
Schreibe Inhalt des Feldes Name  
Schreibe Inhalt des Feldes Adresse1  
Schreibe Inhalt des Feldes Postleitzahl  
Schreibe Inhalt des Feldes Ort  
wenn Anrede = „Frau“  
    dann Schreibe „Liebe“  
    sonst Schreibe „Lieber“  
Schreibe Text des Einladungsdokumentes.

Anhand dieser Beschreibung legen wir fest, was wir unter einem Algorithmus verstehen wollen (aus Broy (1998)):

Ein Algorithmus ist ein Verfahren mit einer präzisen (d.h. in einer genau festgelegten Sprache abgefassten) endlichen Beschreibung unter Verwendung tatsächlich ausführbarer elementarer Verarbeitungsschritte.

Schließlich identifizieren wir in unserem Beispiel die Strukturelemente von Algorithmen: Wiederholung, Auswahl, Sequenz von elementaren Verarbeitungsschritten.

### 1.5.5 Aufgaben

Beschreibe algorithmisch:

- wie du Sahne mit einen Schneebesen steif schlagen kannst,
- wie du aus LEGO-Steinen einen Turm baust,
- wie du einen Pudding kochst.

## 2 Repräsentation von Information

Nach dem im vorigen Kapitel besprochenen Fundamentum begegnen die Schülerinnen und Schüler in der Mittelstufe wieder einem Pflichtfach Informatik. Für den ersten Themenkreis „Repräsentation von Information“ aus diesem Unterrichtsmodul soll nun eine Unterrichtssequenz vorgestellt werden.

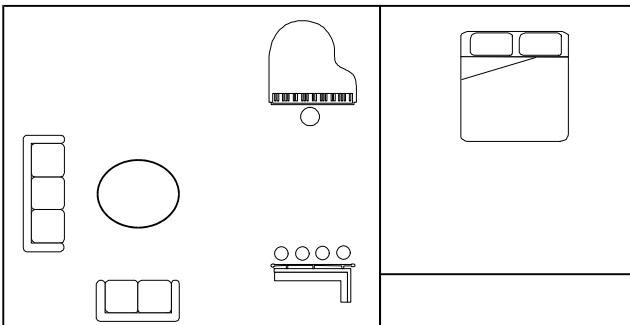
### 2.1 Formen der Repräsentation von Information

Wir betrachten zwei mögliche Repräsentationsformen von Information, nämlich Datenstrukturen und Verarbeitungsvorschriften. Die folgende Unterrichtssequenz widmet sich vor allem der ersten Kategorie, wobei am Beispiel der Datenstrukturen von Raster- bzw. Vektorgrafiken (siehe z.B. Schrack (1978)) zwei grundsätzlich verschiedene Repräsentationsformen derselben (grafischen) Informationen erklärt und im Hinblick auf Kosten und Nutzen verglichen werden. Als Beispiel für Verarbeitungsvorschriften besprechen wir (umgangssprachlich formulierte) Algorithmen zur Umwandlung der Datenformate bzw. zur Datenkompression. Im folgenden Kapitel 3 werden dann Verarbeitungsvorschriften am Beispiel der Tabellenkalkulation wiederholt bzw. eingeführt (siehe auch Hubwieser u. Broy (1996)).

Diese erste Unterrichtssequenz zur 9. Jahrgangsstufe beschränkt sich (wie bereits das Fundamentum) noch auf die Beschreibung vorgegebener Informatiksysteme. Der Übergang zur Beschreibung von Strukturen allgemeiner Systeme der „realen Welt“ erfolgt in der nächsten Sequenz über *Datenmodellierung* (siehe Kapitel 3).

### 2.2 Aufgabenstellung

Wir zeichnen mit einem *Rastergrafikprogramm* (wie *MS-Paint*<sup>®</sup>, *Corel-Paint*<sup>®</sup> o.ä.) eine beliebige Strichzeichnung, bei der sich die Notwendigkeit einer Verschiebung von Figuren ergibt. Eine mögliche Aufgabe wäre etwa (wie bereits in Kapitel 1 vorgeschlagen) die Planung einer Wohnungseinrichtung.



**Abb. 2.1.** Vektorgrafik einer Wohnungseinrichtung

Aus der Aufgabenstellung ergibt sich öfter die Notwendigkeit, beim Einpassen von Möbelstücken bereits gezeichnete Elemente der Grafik zu verschieben. Die Schüler stellen dabei fest, dass die Verschiebung von Objekten hier nur unter Mitnahme einer rechteckigen Umgebung möglich ist. Das erweist sich oft als sehr hinderlich. Deshalb wechseln wir auf ein *vektororientiertes* Zeichenprogramm (*MS-Draw®*, *Corel-Draw®* o.ä.) und bearbeiten damit dieselbe Aufgabenstellung (siehe Abb. 2.1), wobei wir erkennen, dass sich einzelne Figuren nun problemlos ohne Beeinflussung ihrer Umgebung verschieben lassen.

## 2.3 Problemanalyse

Nun wollen wir den Unterschieden zwischen beiden verwendeten Grafikprogrammen auf den Grund gehen. Dazu gehen wir aus einer völlig anderen Richtung an die Problematik heran und simulieren (einfache) Grafiken mit Hilfe einer Tabellenkalkulation. Dies erlaubt einerseits eine Simulation der Situation unter Umgehung der üblichen Grafiksoftware und bietet andererseits die Gelegenheit, eine Tabellenkalkulation aus einer neuen Perspektive zu betrachten. Beides erzeugt zusätzliche kognitive Anknüpfungspunkte im Sinne des konstruktivistischen Prinzips der *kognitiven Flexibilität* (siehe Teil A, Abschnitt 1.4). Außerdem kann durch eine Tabelle die Struktur der Rastergrafik sehr schön veranschaulicht werden.

### 2.3.1 Eine Tabelle als Rastergrafik

Zur Vorbereitung stellen wir in einem Bereich der Tabelle Spaltenbreite und Zeilenhöhe so ein, dass die einzelnen Zellen am Bildschirm sehr klein und nahezu quadratisch erscheinen. Zusätzlich wählen wir eine Symbolschriftart, mit der man schwarze Quadrate darstellen kann (etwa *Wingdings* unter *MS-Windows®*). Dann erstellen wir in der ersten Zeile bzw. Spalte jeweils eine Zahlenfolge als Koordinatensystem.

Nun sollen die Schüler zwei Kreise  $k1(A;10)$  und  $k2(B;5)$  mit  $A(15|15)$  und  $B(25|15)$  aus einzelnen Bildelementen (z.B. aus schwarzen Quadraten) zeichnen. In einem ersten Versuch werden die Pixel manuell eingetragen. Das Ergebnis ist nicht sehr zufrieden stellend. Insbesondere stören die Treppeneffekte, deren Unvermeidbarkeit bei gekrümmten Linien in Rastergrafiken sogleich besprochen werden kann.

Nun stellen wir den Schülern eine Aufgabe, die wiederum die oben bereits festgestellten Probleme der Rastergrafik aufwirft: „Verschiebe  $k2$  so, dass er konzentrisch zu  $k1$  liegt.“

Die nahe liegende Lösung mittels Ausschneiden und Kopieren von  $k2$  hat den Schönheitsfehler, dass wiederum eine Rechtecksfläche mitsamt Teilen des anderen Kreises mitgenommen wird. Ein erneutes Zeichnen beider Kreise scheint unmöglich zu sein.

### 2.3.2 Mathematische Objekte

Wir versuchen einen zweiten Ansatz, indem wir die Funktionalität der Tabellenkalkulation ausnutzen und die Kreispunkte mit Hilfe einer Formel je nach Position der Zelle automatisch eintragen lassen. Dazu benötigen wir zunächst die beiden Kreisgleichungen (bei Bedarf erfolgt eine Herleitung mit Hilfe des Satzes von Pythagoras). Allgemein lauten die Gleichungen

$$k: (x - m_x)^2 + (y - m_y)^2 = r^2.$$

In unserem Fall soll daher in einem Punkt  $P$  der Tabelle ein Quadrat gezeichnet werden, falls  $P(x | y) \in k1$  oder  $P(x | y) \in k2$  gilt, also falls

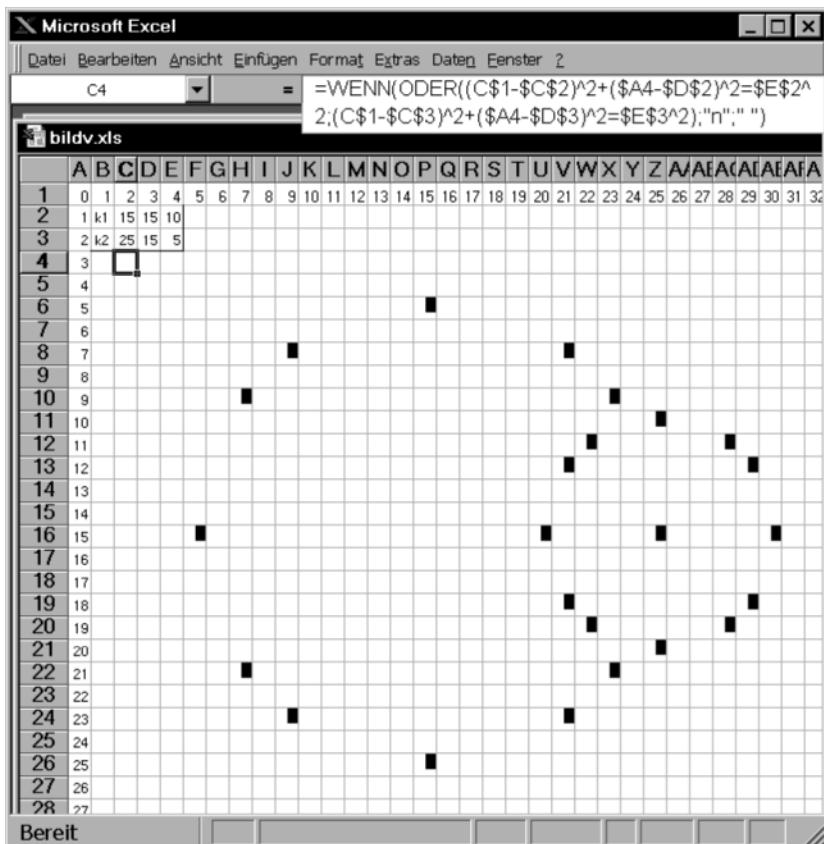
$$(x - 15)^2 + (y - 15)^2 = 100 \vee (x - 25)^2 + (y - 15)^2 = 25.$$

Dies übersetzen wir nun gemäß der Syntax der Tabellenkalkulation mit Hilfe der Tatsache, dass sich die  $x$ -Koordinate eines Punktes über diesem in Zeile 1 bzw. die  $y$ -Koordinate links davon in Spalte A befindet, in einen bedingten Ausdruck (in MS-Works® erzeugt der Wert „n“ im Zeichensatz *Wingdings* das gewünschte schwarze Quadrat):

`WENN(ODER((B$1-30)^2+($A2-20)^2=25;(B$1-20)^2+($A2-20)^2=100);"n";" ")`.

Diesen Ausdruck tragen wir zunächst in Zelle B2 ein, um ihn anschließend in alle noch freien Zellen zu kopieren. Daraufhin werden die Kreise zwar automatisch gezeichnet, das Verschieben eines Kreises ist allerdings immer noch mühsam, weil die Formel geändert und dann wieder in alle Zellen kopiert werden muss. Dieser Nachteil lässt sich beseitigen, indem ein Bereich der Tabelle für den Eintrag der Radien und Mittelpunkte reserviert wird, etwa der Bereich B2–E3, wie in Abb. 2.2. Die zweite Version der Formel nimmt dann auf diese Einträge Bezug:

`WENN(ODER((F$1-$C$2)^2+($A2-$D$2)^2 = $E$2^2;  
(F$1-$C$3)^2 + ($A2-$D$3)^2 = $E$3^2);"n";" ")`.



**Abb. 2.2.** Eine Tabellenkalkulation als Grafikprogramm

Auch diese Formel kopieren wir wieder in alle noch freien Zellen, woraufhin wir auf Knopfdruck Kreise mit beliebigen Mittelpunkten und Radien erzeugen können (siehe Abb.2.2).

Nun ist es auch an der Zeit, anhand der eingetragenen Formeln festzuhalten, dass es sich dabei nicht um die Darstellung statischer Informationen handelt, sondern um Informationen über Verarbeitungsvorgänge in Form von Termen, wie sie den Schülern aus der Algebra seit langem bekannt sind. Neu ist allerdings das Konzept *bedingter Terme*, die in Abhängigkeit vom Wert eines anderen Terms zwei verschiedene Werte liefern können, in unserem Fall das Zeichen „n“ oder ein Leerzeichen.

Zusammenfassend stellen wir fest, dass es offensichtlich zwei Strategien für die Darstellung von grafischen Informationen gibt:

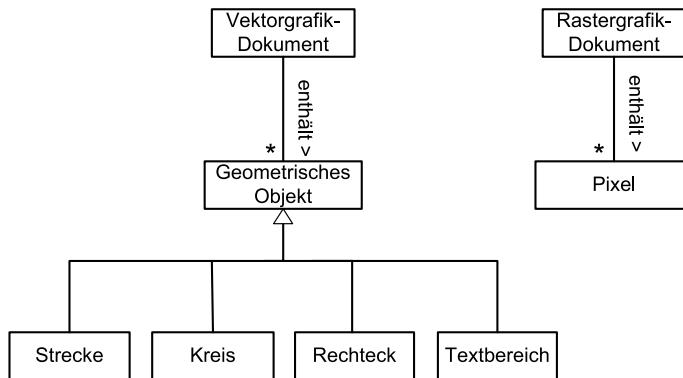
1. Die Eigenschaften eines jeden Punktes im Raster werden einzeln beschrieben (Rastergrafik).
2. Die geometrischen Eigenschaften der darzustellenden Objekte werden mathematisch definiert (Vektorgrafik).

## 2.4 Datenstrukturen

Nun sollen mit Hilfe informatischer Beschreibungstechniken die soeben gewonnenen Erkenntnisse präzisiert und ihre Konsequenzen auf die grafische Datenverarbeitung untersucht werden.

### 2.4.1 Das Datenmodell

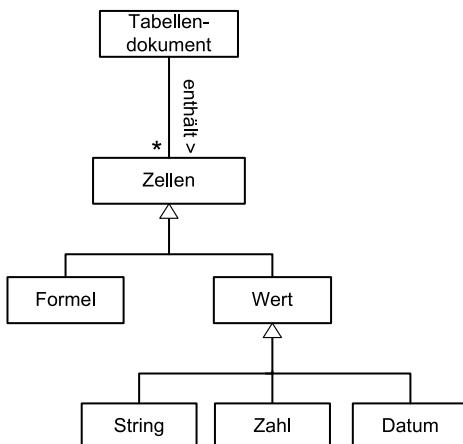
Wir erarbeiten für beide Darstellungstypen je ein Datenmodell, z.B. im Form eines Klassendiagramms (siehe Abb. 2.3).



**Abb. 2.3.** Datenmodelle für Grafiken

Wegen des Auftretens von Varianten (Strecke, Kreis, etc.) muss man dabei ein Generalisierungskonzept für Klassen (zu Oberklassen) verwenden.

Zum Vergleich beschreiben wir mit derselben Technik auch gleich die Datenstruktur einer Tabellenkalkulation (vgl. Abb. 2.4):



**Abb. 2.4.** Datenstruktur einer Tabellenkalkulation (vereinfacht)

Nun wenden wir uns den Besonderheiten der beiden Grafiktypen zu, soweit sie für die Allgemeinbildung relevant sind.

### 2.4.2 Rastergrafik

Bei der Rastergrafik hat man es meist mit einer festen Anzahl von Zeilen und Spalten zu tun (z.B. Bildschirm, Rasterdrucker). Deshalb kann man die geometrische Position aus der Lage des jeweiligen Bildelementes (meist *Pixel* für *picture element* genannt) innerhalb der gesamten Sequenz bestimmen. Falls das Bild die Spalten  $x \in 0 \dots x_{anz}-1$  und die Zeilen  $y \in 0 \dots y_{anz}-1$  hat, so nimmt (im Falle einer Zeilen-Spalten-Nummerierung) das Pixel mit den Koordinaten  $x, y$  innerhalb der gesamten Sequenz die Stelle  $s$  ein, wobei

$$s = y^*(x_{anz}) + x.$$

Umgekehrt ergeben sich die Koordinaten  $x, y$  aus  $s$  über die Formeln (mit div als Operator der Ganzzahldivision):

$$\begin{aligned} y &= s \text{ div } x_{anz}, \\ x &= s \text{ mod } x_{anz}. \end{aligned}$$

Der Farbwert der Bildelemente kann unterschiedlich fein abgestuft sein (Farbtiefe), wobei eine feinere Farbauflösung natürlich auf Kosten des Speicherbedarfs geht. Der Speicherplatz, den ein Bild mit 1014 × 768 Pixeln (unabhängig von seinem Informationsgehalt) beansprucht, ist in Tabelle 2.1 in Abhängigkeit von der Farbtiefe aufgeführt.

**Tabelle 2.1.** Speicherplatzbedarf einer Rastergrafik mit 1024 × 768 Pixeln

Farbtiefe	Anzahl der Farbstufen	Speicherplatz ca.
1 Byte	256	500 kB
2 Byte	65536	1 MB
3 Byte	17 Mio	1,5 MB

Diese Zahlen können durch Betrachten des Plattenplatzverbrauchs unserer bisher produzierten Rastergrafiken (siehe Abschnitt 2.2) unschwer bestätigt werden.

### 2.4.3 Vektorgrafik

Im Gegensatz zur Rasterdarstellung beanspruchen Vektorgrafiken dagegen kaum Speicherplatz. In unserem Datenmodell könnte man zum Beispiel jeder Komponente 4 Byte zugestehen und erhielte so einen Platzbedarf von  $8 \times 4 = 32$  Byte je Objekt. Zusätzlich zu den Objekten müssen bei den Vektorgrafiken allerdings noch Informationen über Bildgröße, Hintergrund usw. mitgeführt werden, die

natürlich auch Platz beanspruchen. Wir können auch das mit Hilfe der Dateigrößen unserer Vektorgrafiken verifizieren.

## 2.5 Verarbeitungsprozesse

Wenn zwei verschiedene Formate für grafische Informationen existieren, dann stellt sich damit auch die Frage, ob (und ggf. wie) man ein Bild von einer Rasterin eine Vektordarstellung transformieren kann. Dabei haben wir, wie bereits oben bei den Berechnungen der Tabellenkalkulation, Verarbeitungsvorgänge vor uns, für deren Beschreibung wir diesmal umgangssprachlich formulierte Algorithmen verwenden.

### 2.5.1 Transformation einer Vektorgrafik in eine Rastergrafik

In dieser Richtung findet sich ein einfacher Algorithmus für die Umwandlung:

*wiederhole* für alle Spalten:

*wiederhole* für alle Zeilen:

*wenn* das Bildelement zu einem Objekt gehört:

*dann* wende dessen Attribute entsprechend an,

*sonst* zeichne in Hintergrundfarbe.

Alle Vektorgrafiken, die auf Rastergeräten wie Bildschirmen oder Laserdruckern ausgegeben werden sollen, müssen auf ähnliche Weise ungerechnet werden.

### 2.5.2 Transformation einer Rastergrafik in eine Vektorgrafik

Diese Transformation erfordert weitaus aufwendigere Algorithmen, die prinzipiell Folgendes leisten müssen:

1. das Pixelmuster ist auf die Existenz bekannter mathematisch beschreibbarer Muster abzusuchen, die
2. abgegrenzt, identifiziert und durch mathematische Funktionen beschrieben werden müssen.
3. Außerdem müssen noch die Attribute dieser Objekte mit passenden Werten belegt werden.

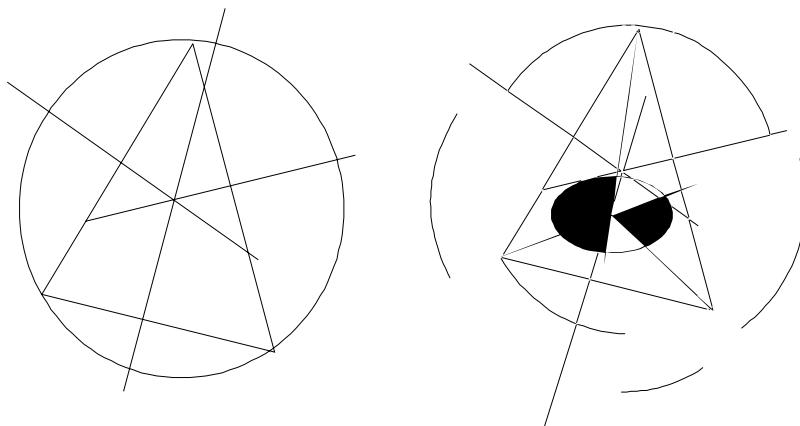
## 2.6 Arbeit mit den Modellen

Zur Festigung und Vertiefung des Gelernten stellen wir uns nun eine konkrete Aufgabe je Datenmodell.

### 2.6.1 Transformationszyklus einer Vektorgrafik

Als Erstes wollen wir ein Dreieck samt Umkreis als Vektorgrafik produzieren und in eine Rastergrafik umwandeln. Das Ergebnis soll dann mit einem Rastergrafikprogramm bearbeitet werden, um nochmals auf die Unterschiede hinzuweisen. Schließlich wollen wir die Rücktransformation in eine Vektorgrafik versuchen und das Resultat mit dem Ausgangsprodukt vergleichen.

Wir zeichnen also zunächst ein Dreieck aus drei Strecken und einen Kreis darum. Dann versuchen wir durch Verschieben, Drehen und Skalieren der Objekte, die gewünschte Situation annähernd zu realisieren. Hier werden die Möglichkeiten der Vektorgrafik verdeutlicht (siehe linke Figur der Abb. 2.5).



**Abb.2.5.** Vektorgrafik vor (links) und nach (rechts) dem Umwandlungszyklus

Nach der Erstellung und Speicherung wandeln wir das Bild (z.B. mit Hilfe der Exportfunktion von *Corel-Draw*<sup>®</sup>) in eine Rastergrafik (z.B. im *BMP*-Format) um. Ein Vergleich des Umfangs der beiden Dateien zeigt uns, dass die Rastergrafik im Vergleich zur Vektorgrafik ungefähr den 50fachen Platz belegt.

Nun laden wir unsere Rastergrafik in ein rasterorientiertes Zeichenprogramm (z.B. *Corel-Paint*<sup>®</sup>) und versuchen, mittels der Winkelhalbierenden einen Inkreis einzuziehen. Wegen der mangelnden Verschiebungsmöglichkeiten gelingt uns dies nur schwer, was uns wiederum die Nachteile der Rastergrafik vor Augen führt.

Im nächsten Schritt vektorisieren wir dann unsere Rastergrafik mittels eines geeigneten Programms (etwa *Corel-Trace*<sup>®</sup>). Wenn wir das Resultat wiederum in unser Vektorzeichenprogramm laden, stellen wir fest, dass wir nun erneut Zugriff auf die einzelnen Objekte haben. Allerdings sind diese jetzt anders gruppiert, so werden Kreise dabei meist in einzelne Bögen zerlegt (siehe rechte Figur in Abb.2.5).

## 2.6.2 Rastergrafik und Fotoretusche

Da die Vektorgrafik offensichtlich so leicht zu handhaben ist, müssen wir uns fragen, wozu man Rastergrafiken überhaupt einsetzen kann. Als typische Anwendung dafür wollen wir deshalb in einer Foto-Retusche ein Werbephoto für einen Urlaubsort verschönern (oder eine geplante Umgehungsstraße in eine vorher unbeführte Landschaft einpassen o.ä.).

Mittels eines Scanners digitalisieren wir ein Foto eines griechischen Urlaubsortes aus einem Reisekatalog und retuschieren den darauf abgebildeten monströsen Hotelkomplex weg. Für Fotobearbeitung wäre das Konzept der Vektorgrafik ungeeignet, da sich kaum identifizierbare geometrische Objekte in den Bildern finden lassen (siehe Abb. 2.6).

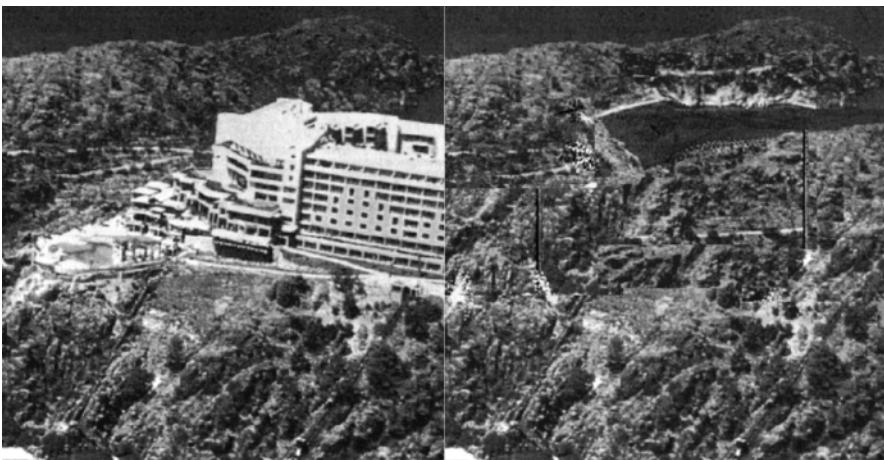


Abb. 2.6. Original (links) und Retusche (rechts)

## 2.7 Diskussion und Ausblick

Abschließend fassen wir Vor- und Nachteile der beiden Darstellungsarten zusammen und besprechen spezielle Einsatzgebiete (siehe Tabelle 2.2) .

**Tabelle 2.2.** Vor- und Nachteile von Vektor- bzw. Rastergrafik

	Rastergrafik	Vektorgrafik
Charakter	Sequenz von Pixeln	Arrangement von mathematischen Objekten
Vorteile	gut retuschierbar, keine analytische Beschreibung der Formen nötig, einfache Verarbeitungsalgorithmen	geometrische Operationen mit den Objekten sind leicht realisierbar, geringer Speicherplatzverbrauch

---

Nachteile	hoher Speicherplatzverbrauch	komplexe Erkennungsalgorithmen
Anwendungen	fotorealistische und künstlerische Bilder	Bilder mit geometrischen Objekten

---

Als Ausblick kann man noch mögliche Komprimierungsstrategien für Rasterbilder behandeln.

### 2.7.1 Graphics Interchange Format (GIF)

Das *GIF*-Kompressionsformat der Firma *Compuserve*® benützt ein Kompressionsverfahren ohne Informationsverlust. Häufig vorkommende Folgen von Bytes werden dabei in Tabellen eingetragen und durch einen Verweis auf diesen Eintrag ersetzt. Ein Beispiel dazu (aus Savola (1995)): Der Text

„The rain in Spain falls mainly on the plain, while the rain in the Amazon just falls“

enthält 85 Zeichen. Mit den Abkürzungen:

$W := \text{„the“}$ ,  $X := \text{„ain“}$ ,  $Y := \text{„on“}$ ,  $Z := \text{„falls“}$

erhält man daraus die folgende auf 57 Zeichen (das entspricht 67% der ursprünglichen Anzahl) komprimierte Darstellung:

„W rX in SpX Z mXly Y W plX, while W rX in W AmazY just Z“.

Dabei geht keinerlei Information verloren. Zusätzliche Kosten entstehen lediglich durch den Zeitverbrauch des Kompressionsalgorithmus Darstellung und den Platzverbrauch der Zuordnungstabelle.

### 2.7.2 Joint Photographic Experts Group (JPG)

Hier werden die Farben des Bildes analysiert und dann diejenigen Informationen weggelassen, die für das menschliche Auge nicht besonders wichtig sind. Bei dieser Kompressionstechnik geht im Gegensatz zum *GIF*-Verfahren tatsächlich Information verloren, was in den meisten Fällen bei einem Vergleich mit dem unkomprimierten Bild deutlich sichtbar wird.

Unser Hotelfoto aus Abb. 2.6. beispielsweise belegt im *BMP*-Format 246 kB, als *GIF*-Datei 153 kB und nach Kompression zum *JPG*-Bild nur noch 25 kB.

# **3 Funktionale Modellierung, Teil 1**

Der Themenkreis funktionale Modellierung soll in zwei Stufen dargestellt werden. In diesem Kapitel soll mit einer „Black-Box“-Sicht der informationsverarbeitenden Prozesse gearbeitet werden, bei der wir uns auf die Zerlegung größerer Systeme in Subsysteme und die Beschreibung des Informationsaustausches zwischen diesen Subsystemen beschränken. Das Innenleben (d.h. den Algorithmus) dieser Subsysteme (die als informationsverarbeitende Prozesse angesehen werden), soll außer Acht gelassen werden. Im zweiten Teil des Kapitels wird die funktionale „Black-Box“-Sichtweise zur Strukturierung von Computerhardware angewandt.

In Kapitel 6 folgt dann ein Projekt zur funktionalen Modellierung, das auch die innere Verarbeitungsstruktur der Prozesse modelliert und mit Hilfe einer Skriptsprache programmiert.

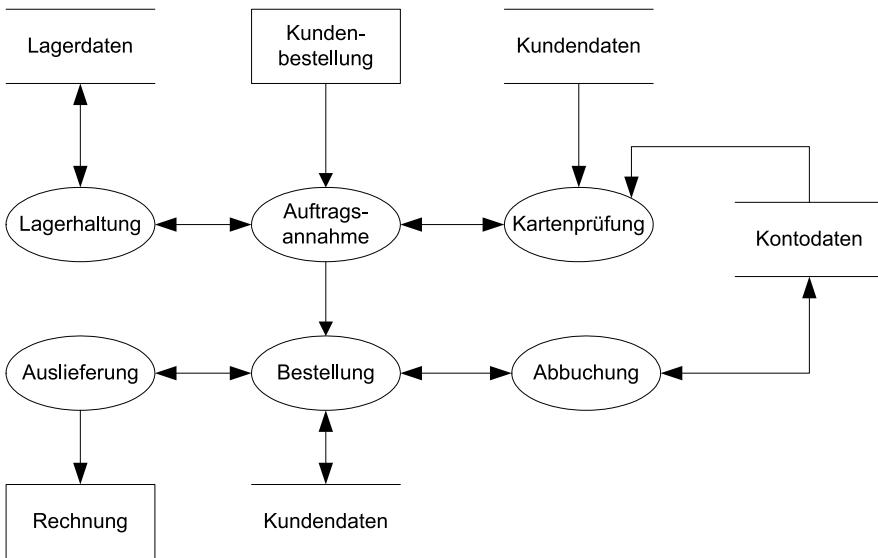
## **3.1 Modellierung mit Hilfe von Funktionen**

In diesem Abschnitt geht es vor allem um die Beschreibung von Systemen durch Datenflussdiagramme. Als Implementierungswerkzeug wählen wir hier Tabellenkalkulationssysteme, die sich wegen der funktionalen Struktur ihrer Formeln sehr gut eignen (siehe auch Hubwieser (2004)). Dies bringt allerdings die Einschränkung mit sich, dass nur informationsverarbeitende Prozesse verwendet werden können, die als Funktionen dargestellt werden können (siehe Hubwieser (2005)). Nur dann können sie auch durch die vordefinierten Funktionen eines Tabellenkalkulationssystems (z.B. SUMME, MITTELWERT, etc.) simuliert werden. Eine ausführliche Darstellung des Lehrweges sowie zahlreiche Aufgaben und Beispiele finden sich in unserem Unterrichtswerk Hubwieser et. al. (2007).

### **3.1.1 Datenflussdiagramme**

Zunächst sollen die Schülerinnen und Schüler die grundlegenden Prinzipien der funktionalen Modellierung kennen lernen. Hierbei geht es grundsätzlich um eine Lösung des Problems, dass man komplexe Systeme oft nicht im Ganzen bis in jedes Detail beschreiben kann. Zur Abhilfe gliedert das betrachtete System (z.B. einen Betrieb) in mehrere Teilsysteme (z.B. Einkauf, Vertrieb, Marketing, ...), beschreibt die Interaktion bzw. Kommunikation *zwischen* diesen Teilsystemen (Komponenten) durch ein Datenflussdiagramm (siehe Abb. 3.1), lässt dabei aber die innere Struktur der Komponenten (zumindest vorläufig) außer Acht. Daher spricht man auch von „Black-Box“-Sicht. Die Teilsysteme (z.B. Auftragsannah-

me, Lagerhaltung in Fig. 3.1) werden als informationsverarbeitende Prozesse betrachtet, die bestimmte Eingabedaten entgegennehmen, zu Ausgabedaten verarbeiten und diese zur Verfügung stellen.



**Abb. 3.1.** Datenflussdiagramm eines Versandhauses

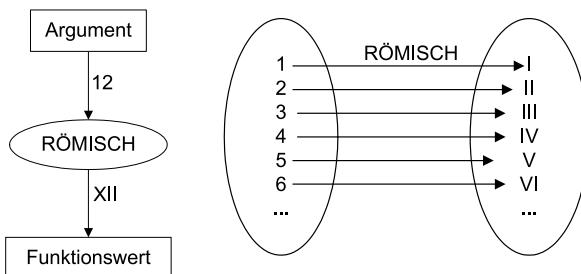
Datenflussdiagramme wurden bereits 1966 in DIN 66001 genormt. In der neuesten Fassung von Dezember 1983 findet sich folgende Definition: „Ein Datenflussplan stellt Verarbeitungen und Daten sowie die Verbindungen zwischen beiden dar.“ In dieser Norm werden 19 verschiedene grafische Symbole zur Darstellung aufgeführt, wovon heute allerdings nur noch ein Teil verwendet wird. Wir beschränken uns im Unterricht zunächst auf vier Arten von Elementen (Beispiele siehe Abb. 3.1):

- informationsverarbeitende Prozesse (Ellipsen), z.B. Bestellung, Abbuchung,
- Ein- und Ausgabedaten (Rechtecke), z.B. Kundenbestellung, Rechnung
- Datenspeicher (begrenzt von zwei horizontalen Linien), z.B. Kontodaten, Kundendaten
- Datenflüsse zwischen den o.g. drei Arten von Elementen, wobei direkte Datenflüsse zwischen Ein- und Ausgabestellen sowie Datenspeichern nicht zugelassen sind.

### 3.1.2 Funktionen

Nun schränken wir die Verwendung informationsverarbeitender Prozesse auf Funktionen ein, die als eindeutige Zuordnungen zwischen Ein- und Ausgabewerten definiert werden. Hier müssen sich die Schüler auch an nichtnumerische Funktionen gewöhnen, denen sie in der Schulmathematik meist nicht begegnen. Im

Gegensatz zur Mathematik hat die Informatik für den etwas nebulösen Begriff „Zuordnung“ eine handfeste Bedeutung anzubieten: die Zuordnung wird durch tatsächlich funktionsfähige Automaten hergestellt, die Eingabedaten entgegennehmen und ihnen die jeweils daraus berechneten Ausgabedaten zuordnen. Im einfachsten Fall ist ein solcher Automat eine vordefinierte Funktion des Tabellenkalkulationssystems (z.B. RÖMISCH in Abb. 3.2), in komplizierteren Fällen eine ganzes Informatiksystem, das aus vielen Hard- und Softwarekomponenten besteht (z.B. das Internet).



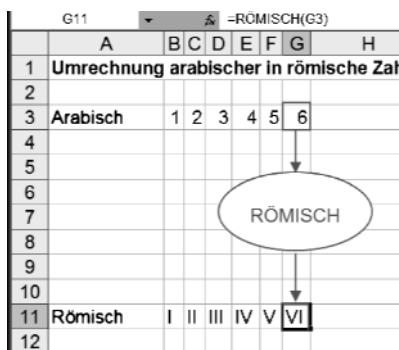
**Abb. 3.2.** Zuordnung durch die Funktion RÖMISCH

### 3.1.3 Tabellenkalkulationssysteme

Spätestens an dieser Stellen erfolgt der Einsatz eines Tabellenkalkulationssystems zur Veranschaulichung der funktionalen Strukturen. Nach einer kurzen Erklärung der Struktur der Rechenblätter (als Dokumente dieser Systeme), die aus Zellen bestehen, die wiederum in Spalten und Reihen angeordnet sind und darüber auch eindeutig identifiziert werden (z.B. Zelle B7). Die Zellen können (konstante Werte (Daten) oder Formeln enthalten. Formeln beginnen mit einem Gleichheitszeichen, auf das wiederum ein Term folgt. Jede Zelle hat neben ihrem Bezeichner (z.B. B7) einen Typ (z.B. Text oder Zahl) und einen Wert. Falls die Zelle einen konstanten Wert (Datum als Singular von Daten) enthält, ist dieser auch ihr Wert. Falls sie eine Formel enthält, wird ihr Wert durch Auswertung des Terms der Formel berechnet.

Die Strukturen von Datenflussdiagrammen werden dann folgendermaßen in Rechenblättern abgebildet (siehe Abb. 3.3) :

- Funktionen (als Spezialfall informationsverarbeitender Prozesse) werden durch Formeln dargestellt,
- Datenflüsse werden über Zellbezüge in diese Formeln eingebaut,
- Ein- und Ausgabestellen sind Zellen, deren Inhalt über Zellbezüge in Formeln transferiert wird.

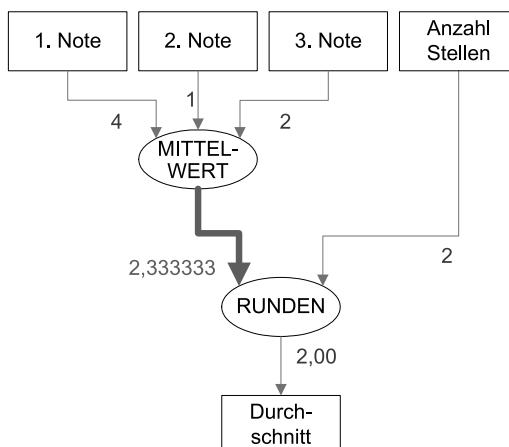


**Abb. 3.3.** Datenflussdiagramme in Rechenblättern

### 3.1.4 Spezielle Aspekte von Funktionen

Neben nichtnumerischen Funktionen begegnen die Schülerinnen und Schüler hier vermutlich auch erstmals Funktionen mit mehreren Argumenten, wie SUMME, POTENZ oder MITTELWERT. Bei Funktionen mit zwei Argumenten sollte hier auch die verbreitete Infixschreibweise unter Verwendung von Operatorsymbolen (z.B. „+“ für SUMME) angesprochen werden. Bei Funktionen mit einer beliebigen Anzahl von Argumenten wie SUMME oder MITTELWERT kann man als Argument meist auch rechteckige Zellbereiche angeben, die mit dem Bereichsoperator „:“ (z.B. B4:C12) spezifiziert werden.

Richtig interessant wird die Arbeit mit den Rechenblättern aber erst durch die Möglichkeit, Funktionen zu verketten. Dazu wird die Ausgabe einer Funktion als Argument einer anderen Funktion benutzt (siehe Abb. 3.4).



**Abb. 3.4.** Verkettung von Funktionen im Datenflussdiagramm

### 3.1.5 Die WENN-Funktion

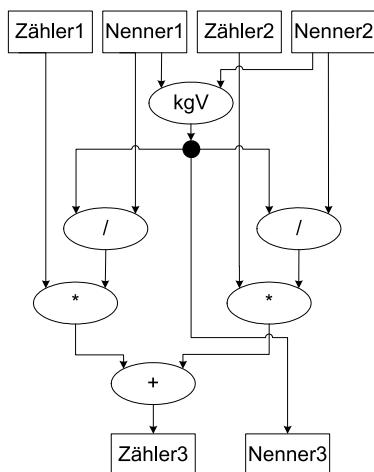
In vielen Algorithmen benötigt man bedingte Verarbeitungsschritte, für die Tabellenkalkulationen die dreistellige Funktion WENN zur Verfügung stellen:

WENN(B2=0; „FEHLER“; A2/B2).

Dieses Konzept bereitet den Schülerinnen und Schülern erfahrungsgemäß (siehe z.B. Hubwieser (2005)) erhebliche Probleme. Es sollte daher sehr behutsam, sorgfältig und ausführlich eingeführt werden. Als hilfreich hat sich die Betonung folgender Aspekte erwiesen:

- Man benötigt die WENN-Funktion immer dann, wenn je nach Wert einer Bedingung unterschiedliche Verarbeitungsstrukturen (d.h. Terme) zum Einsatz kommen sollen.
- Bei der Auswertung der WENN-Funktion wird zunächst die Bedingung ausgewertet. Ist ihr Wert WAHR, dann wird der Term im ersten Argument ausgewertet, andernfalls der Term im zweiten Argument. Der jeweils andere Term wird dann nicht ausgewertet, er muss also für die vorliegende Eingabe nicht unbedingt definiert sein (wie z.B. A2/B2 im obigen Beispiel).

Mit diesem Rüstzeug sind die Schülerinnen und Schüler nun in der Lage, auch komplexere Aufgabenstellungen zu bewältigen, wie z.B. die funktionale Darstellung der Addition zweier ungleichnamiger Brüche (siehe Abb. 3.5). Hierzu verwenden wir ein neues Element von Datenflussdiagrammen: einen fetten Punkt als Symbol für das Kopieren eines Wertes auf mehrere Datenflüsse.

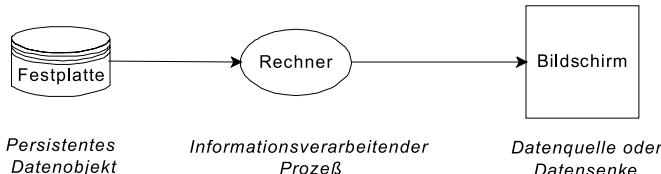


**Abb. 3.5.** Datenflussdiagramm für die Addition zweier Brüche

Als Abschluss der Thematik kann noch auf die Strategie der schrittweisen Verfeinerung eingegangen werden. Dabei erstellt man zunächst Datenflussdiagramme in einer relativ groben Granularität, die dann verfeinert werden, indem für jeden Prozess (z.B. kgV) wiederum ein eigenes Datenflussdiagramm entwickelt wird.

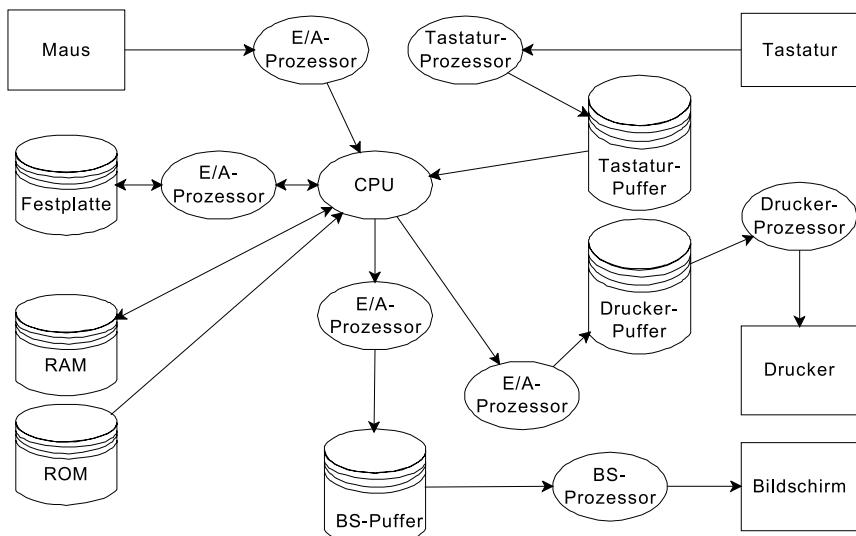
## 3.2 Funktionale Modelle von Hardware

Abschließend wollen wir noch ein Beispiel für eine Systembeschreibung anhand des funktionalen Modells eines Einzelplatzrechners erarbeiten. Wir konzentrieren uns dabei auf die Beschreibung der Aufteilung eines Systems in Subsysteme und der Datenflüsse zwischen diesen Subsystemen. Das Modell kann natürlich beliebig verfeinert werden. Wir verwenden weiter die (leicht abgewandelte) „Sprache“ der Datenflussdiagramme, wie sie in Abschnitt 3.7. eingeführt wurde.



**Abb. 3.6.** Elemente von Datenflussdiagrammen

Wendet man diese Technik auf einen Einzelplatzrechner an, dann könnte das Ergebnis ungefähr so aussehen, wie es in Abb. 3.7 dargestellt ist. Ein willkommener Nebeneffekt dieser Modellierungsaufgabe liegt in der Wiederholung und Systematisierung der Vorkenntnisse über die Funktionsweise von Einzelcomputern.



**Abb. 3.7.** Funktionales Modell eines Rechners

In den folgenden Lernsequenzen werden wir auf diese Technik der funktionalen Modellierung immer wieder zurückgreifen, wenn es darum geht, Datenflüsse zwischen den Komponenten eines Systems zu modellieren.

# 4 Datenmodellierung und Datenbanken

Während wir uns in den vorausgegangenen Unterrichtssequenzen auf die Modellierung vorgegebener Informatiksysteme beschränkt haben, gehen wir jetzt zur Modellierung von Systemen aus dem Alltags- bzw. Berufsleben über. Dafür verwenden wir eine Reihe von Techniken, die jeweils einen speziellen Aspekt dieser Ausschnitte besonders betonen. Den Anfang macht die *datenorientierte Modellierung* mit Hilfe von Entity-Relationship-Modellen (kurz: ER-Modellen) in diesem Kapitel, die sich auf die statischen Aspekte der beschriebenen Systeme beschränkt und daher den Einstieg in die Modellierung erleichtert.

Zur Simulation solcher statischer Modelle eignen sich relationale Datenbanksysteme, deren Behandlung wegen ihrer großen Verbreitung in der Praxis (Allgemeinheitsklasse 3, siehe Abschnitt 4.3.1 in Teil B) bereits an sich einen gewissen Bildungswert aufweist.

Anhand der Verwaltung einer Bibliothek wollen wir die Vorgehensweise bei der datenorientierten Modellierung sowie die Simulation der Modelle veranschaulichen. Natürlich können wir im Rahmen des Unterrichts kein vollständiges Bibliothekssystem konstruieren, sondern lediglich kleine Ausschnitte daraus.

Die vordringliche Intention dieses Kapitels liegt im Aufzeigen eines schülergemäßen Weges durch den Themenbereich „Datenmodellierung und Datenbanken“, was an manchen Stellen ein detailliertes Eingehen auf den Lehrstoff erfordert. Dabei geht es uns nicht um die Vermittlung von Grundlagenwissen für Lehrkräfte, weshalb im Zweifelsfall immer Exaktheit und Kompaktheit zugunsten von Verständlichkeit zurückgestellt werden. Solches Hintergrundwissen kann z.B. den Werken von Korth u. Silberschatz (1991), Meier (1998), Kemper u. Eickler (1999) oder Horn u. Kerner (1995, 1997) entnommen werden. Ein alternativer Lehrweg sowie zahlreiche Aufgaben und Beispiele finden sich in unserem Unterrichtswerk Hubwieser et. al. (2007).

## 4.1 Beschreibung der Anforderungen

Anfangs untersuchen wir in einer ausgiebigen Diskussionsphase die gesamte für uns relevante Domäne. Sehr hilfreich wäre wegen der hohen anschaulichkeit von originalen Begegnungen (siehe Teil A, Abschnitt 2.5) der Besuch in einer (möglichst großen) Bibliothek. Anlässlich eines solchen Unterrichtsgangs könnte man durch Interviews mit Angestellten detaillierte Informationen über die Anforderungen an unser System einholen. Falls die besuchte Bibliothek über elektronische Unterstützung verfügt, müssen wir allerdings darauf achten, uns nicht allzu sehr von der Struktur dieses speziellen Systems beeinflussen zu lassen. Schließlich

wollen wir hier eine „reale“ Bibliothek modellieren und nicht ein vorgegebenes Informatiksystem. Bei der Erstellung des Pflichtenheftes leistet uns das Konzept der *Nutzungsfälle* (*use-cases*, siehe Booch, Rumbaugh, Jacobson (1997)) gute Dienste. Schließlich einigen wir uns auf die folgenden Anforderungen an unser System:

- Verwaltung von Informationen über Buchbestand, Lieferanten, Autoren und Kunden,
- Unterstützung von Kundenberatung, Ausleihe, Bestandskontrolle und Nachbestellung.

## 4.2 Datenmodellierung

Nach der informellen Beschreibung der Anforderungen ist es nun an der Zeit, das geplante System formal zu beschreiben. Für die Datenmodellierung setzen wir die Entity-Relationship-Technik nach Chen (1976) ein. Im Hinblick auf die Implementierung erzeugen wir anschließend auf zwei verschiedenen Wegen jeweils ein relationales Modell (als Satz von Tabellen) und zeigen dann, dass man dabei annähernd dasselbe Resultat erhält:

1. „naive“ relationale Modellierung und Transformation dieser Tabelle(n) in die 3. Normalform,
2. schematische Umsetzung des ER-Modells in ein relationales.

Die Schüler erkennen so, dass die letztendlich verwendete Datenstruktur weniger von der angewandten Technik als von den Vorgaben der Domäne, also von der zu modellierenden Informationsstruktur, abhängt.

### 4.2.1 Das Entity-Relationship Modell

Wir beginnen mit dem Entwurf des Entity-Relationship-Modells für unser System. Es besteht aus einer Menge von Klassen und Beziehungen zwischen diesen Klassen. Je nach unterrichtlicher Vorgeschiede kann man dabei die (ursprünglichen) Bezeichnungen der Datenbanktechnik oder objektorientierte Begriffe verwenden:

- *Entitäten* oder *Objekte* (als Instanzen von Klassen),
- *Entitätsmengen* oder *Klassen von Objekten*.

Das Fehlen von Methoden steht dabei der Verwendung des Objektbegriffes nicht entgegen. Wir verwenden diese Begriffe wie Horn u. Kerner (1995):

Eine Entitätsmenge ist eine benannte Zusammenfassung von Objekten (Individuen, Sachen, Begriffen, Ereignissen) des Realitätsausschnittes mit gleichen Eigenschaften, die zu einem bestimmten Zeitpunkt existieren. Eine Entität ist ein eindeutig identifizierbares Element aus einer Entitätsmenge.

Kemper (1999) verwendet statt „Entitätsmenge“ die Begriffe „Entitymenge“ oder (m.E. besser) „Entitytyp“. Falls die objektorientierten Begriffe den Schülern bereits aus dem Anfangsunterricht vertraut sind, empfehlen wir, diese auch weiterhin zu verwenden und zur Verständigung mit der Datenbankwelt ein „Übersetzungs-wörterbuch“ anzulegen. Ein weiteres Argument für objektorientierte Terminologie ist die häufig unsaubere Verwendung des Begriffs „Entität“ (statt „Entitätsmenge“) in der einschlägigen Datenbankliteratur (z.B. in Schwinn (1992)).

**Objekte (Entitäten).** Der erste Schritt besteht in der Identifizierung der beteiligten Objekte. Hierbei handelt es sich um individuelle und identifizierbare Elemente, Individuen, Sachen, Begriffe, Ereignisse o.ä. innerhalb des Systems, die durch ihre Eigenschaften (Attribute) beschrieben werden. Tabelle 4.1 zeigt drei mögliche Objekte unserer Bibliothek.

**Tabelle 4.1.** Einige Objekte aus unserer Bibliothek

Objekt	Attributname	Attributwerte
Buch1	Titel	Fräulein Smillas Gespür für Schnee
	Autor	Peter Høeg
	Verlag	Carl Hanser
	Ort	München
	Jahr	1994
	ISBN	3-499-13599-x
	Preis	19,90
	Exemplare	3
	Zustand	gut, sehr gut, gut
	Standort	Zimmer 3, Regal 5, Fach 7,7 und 8, Platz 3, 15,12
Autor1	Name	Høeg
	Vorname	Peter
	Land	Dänemark
	andere	Der Plan von der Abschaffung des Dunkels,
	Buchtitel	Die Frau und der Affe
Kunde1	Name	Meier
	Vorname	Peter
	Geburtsdatum	24.4.1966
	Straße	Tegtmüllerweg 9
	PLZ	80089
	Ort	München
	Telefon	(089) 383 245 12
	Ausleihsperrre	keine

**Klassen (Entitätsmengen).** Objekte (Entitäten) mit gleicher Attributstruktur werden dann unter einem Oberbegriff zu *Klassen* (Entitätsmengen) zusammengefasst. Dabei stellen wir u.a. fest, dass die Möglichkeit, mehrere Exemplare eines Buchtitels zu verwalten, bei der Klassifizierung Probleme macht. Wir führen also eine eigene Klasse für die *Exemplare* ein. Bei den Personen stellen wir Unterschiede bei den Attributen von *Autoren* und *Kunden* fest. Damit erhalten wir vorerst die folgenden Klassen, wobei die Attribute in Klammern notiert werden:

- *Buchtitel* (Titel, Autor, Verlag, Ort, Jahr, ISBN, Preis),
- *Exemplar* (Bezeichnung, Zustand, Aufnahmedatum),
- *Standort* (Zimmer, Regal, Fach, Platz),
- *Autor* (Name, Vorname, Land, andere),
- *Kunde* (Name, Vorname, Geburtsdatum, Straße, PLZ, Ort, Telefon, Sperre).

Allerdings fehlen uns noch einige Informationen: Wie sollen die Zuordnungen von Buchtiteln zu Autoren, Exemplaren und Kunden geregelt werden? Wie soll man mit der Liste weiterer Veröffentlichungen („andere“) des jeweiligen Autors umgehen?

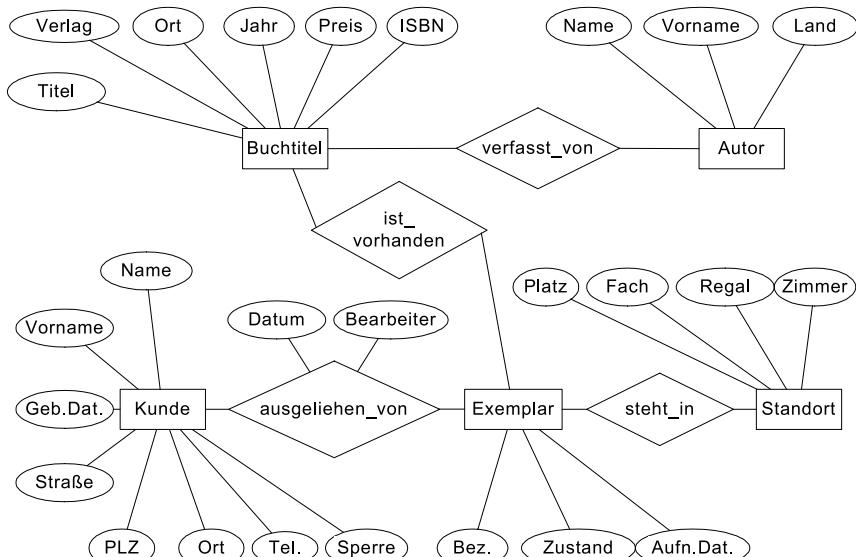


Abb. 4.1. ER-Modell der Bibliothek

**Beziehungen.** Die Lösung dieser Probleme wird durch das Konzept der Beziehungen (*relationships*) zwischen Klassen geliefert. *Buchtitel* und *Autor* werden durch die Beziehung *verfasst\_von* verbunden. Damit erübriggt sich gleichzeitig die Verwaltung von „weiteren Exemplaren“. *Buchtitel* und *Exemplar* können durch die Beziehung *ist\_vorhanden* verknüpft werden, *Exemplar* und *Kunde* durch die Beziehung *ausgeliehen\_von*, *Exemplar* und *Standort* durch *steht\_in*.

Bei näherer Betrachtung zeigt sich zudem, dass auch Beziehungen Eigenschaften (Attribute) haben können. So ist es etwa sinnvoll, unsere Beziehung *ausgeliehen\_von* mit den Attributen *Datum* und *Bearbeiter* zu versehen.

**ER-Diagramme.** Um die Übersicht zu bewahren, stellt man ER-Modelle meist grafisch in ER-Diagrammen dar. Deren Elemente sind *Klassen* (Rechtecke), *Beziehungen* zwischen diesen Klassen (Rauten) und *Eigenschaften* (Attribute) der Klassen und Beziehungen (Ellipsen). In unserem Fall erhalten wir das in Abb. 4.1. dargestellte ER-Diagramm. In umfangreicheren ER-Diagrammen lässt man die Attribute zugunsten der Übersichtlichkeit oft weg.

Zur Umsetzung von Beziehungen in Datenbanktabellen ist es von großer Bedeutung, wie viele Objekte der einen Seite jeweils mit Objekten der anderen Seite verknüpft werden. Diese Eigenschaft heißt *Kardinalität* der Beziehung. Dafür gibt es im Wesentlichen drei Möglichkeiten (die ggf. noch weiter verfeinert werden könnten):

**Kardinalität 1:1.** Einem Objekt der einen Seite wird *genau ein* Objekt der anderen Seite zugeordnet und umgekehrt. Tabelle 4.2 zeigt ein Beispiel dafür.

**Tabelle 4.2.** Eine Beziehung mit der Kardinalität 1:1 zwischen *Exemplar* und *Standort*

<i>Exemplar</i> (Nr.)	<i>steht_in</i>	<i>Standort</i>
10220	↔	Zimmer 2, Regal 2, Fach 12, Platz 14
12110	↔	Zimmer 3, Regal 3, Fach 1, Platz 12
23311	↔	Zimmer 1, Regal 12, Fach 12, Platz 2

**Kardinalität 1:n.** Einem Objekt der einen Seite können mehrere Objekte der anderen Seite zugeordnet werden, umgekehrt aber höchstens ein Objekt. Diese Kardinalität weist z.B. die Relation *ist\_vorhanden* auf, wie in Tabelle 4.3 erläutert wird.

**Tabelle 4.3.** Ein Beispiel für die Kardinalität 1:n

<i>Buchtitel</i>	<i>ist_vorhanden</i>	<i>Exemplar</i> (Nr.)
Fräulein Smillas Gespür für Schnee	↔	10223
Fräulein Smillas Gespür für Schnee	↔	10224
Fräulein Smillas Gespür für Schnee	↔	10225

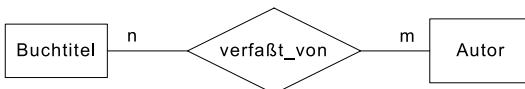
**Kardinalität m:n.** Einem Objekt der einen Seite können mehrere Objekte der anderen Seite zugeordnet werden und umgekehrt. Dieser Fall kann immer in zwei Beziehungen mit den Kardinalitäten 1:m bzw. n:1 aufgelöst werden. Die Beziehung *verfasst\_von* ordnet beispielsweise dem Schulbuchtitel „Physik, Jahrgangsstufe 8“ seine drei Autoren Herbert Knauth, Siegfried Kühnel und Hubert Schafbauer zu, während der Autor Peter Høeg mit seinen Werken „Fräulein Smillas Gespür für Schnee“ und „Der Plan von der Abschaffung des Dunkels“ verknüpft wird (siehe Tabelle 4.4).

**Tabelle 4.4.** Eine m:n-Beziehung

Buchtitel	verfasst_von	Autor
Physik, Jahrgangsstufe 8	↔	Herbert Knauth
Physik, Jahrgangsstufe 8	↔	Siegfried Kühnel
Physik, Jahrgangsstufe 8	↔	Hubert Schafbauer
Fräulein Smillas Gespür für Schnee	↔	Peter Høeg
Der Plan von der Abschaffung des Dunkels	↔	Peter Høeg

Diese Beziehung könnte man auflösen in zwei getrennte 1:n-Beziehungen, etwa *hat\_den\_Autor* und *ist\_Mitautor\_von*.

Im ER-Diagramm notieren wir die Kardinalitäten am linken und rechten Anknüpfungspunkt der Beziehungen. In Abb. 4.2 ist das für eine n:m-Kardinalität ausgeführt.

**Abb. 4.2.** Kardinalität im ER-Diagramm

### 4.2.2 Relationale Modellierung

Überall dort, wo es um die Aufbewahrung oder Verarbeitung von großen Datensätzen geht, sind Datenbanksysteme im Spiel. Der Betrieb dieser Systeme ist solange unproblematisch, wie die zugrunde liegende Datenstruktur gut an das Problem angepasst ist und gewisse Randbedingungen erfüllt, wie etwa die Vermeidung von Redundanz. Die Einhaltung von so genannten *Normalformen* sorgt dafür, dass am Ende eine brauchbare Datenstruktur entsteht. In der Regel ist das Ergebnis einer solchen *Normalisierung* nahezu identisch mit der Struktur, die man durch die Transformation eines (guten) ER-Modells in ein relationales Modell erhält, wie sie im Abschnitt 4.2.3 beschrieben wird.

**Relationen.** Die Bezeichnung „relationales Modell“ entspringt dem mathematischen Begriff „Relation“. Eine solche *mathematische Relation R* ist eine Menge von Tupeln  $r = (r_1, \dots, r_n)$ , deren Komponenten  $r_i$  jeweils aus einer bestimmten Grundmenge  $R_i$  stammen. Eine Relation  $R$  ist also eine Teilmenge des kartesischen Produktes aller dieser Grundmengen:

$$R = \{r \mid r = (r_1, \dots, r_n) \wedge r_1 \in R_1, \dots, r_n \in R_n\} \subset R_1 \dots R_n$$

Ein entscheidendes Merkmal einer Relation ist ihre (feste) Stelligkeit, d.h. die Anzahl der Elemente ihrer Tupel. In der Informatik verwendet man als Grundmengen neben Zahlenmengen oft auch Mengen von *Texten*. Die Grundmengen

erhalten meist je einen Namen und werden damit zu *Sorten* (siehe Broy (1997)), zum Beispiel:

*Ort* = { „Bad Aibling“, „Kolbermoor“, „München“, „Rosenheim“, „Großkarolinenfeld“}

oder

*Männer* = { „Ernst Huber“, „Franz Muxenauer“, „Hanno Buckmann“}.

Spätestens an dieser Stelle muss klargestellt werden, dass man in der Informatik zwischen *Texten als Konstanten* und *Texten als Bezeichnung für Werte* (im Sinne von Variablen) unterscheiden muss. So bezeichnet *Ort* eine Menge von (konstanten) Texten, während „Bad Aibling“ ein konstanter Text ist. Zur Unterscheidung dieser beiden Bedeutungen setzen wir konstante Texte in Anführungszeichen, falls die Gefahr einer Missdeutung besteht. In Tabellen können wir diese Anführungszeichen daher meist weglassen.

Jetzt arbeiten wir uns mit den Schülern ausgehend vom mathematischen Relationsbegriff schrittweise bis zur Vorstellung einer Relation als *Tabelle* vor:

1. Wir betrachten die Mengen  $A = \{2, 3, 4\}$  und  $B = \{5, 6, 8\}$  sowie die Relation  $R_1 = \{(a, b) \in A \times B \mid a \text{ teilt } b\}$ . Dann gilt  $A \times B = \{(2, 5), (2, 6), (2, 8), (3, 5), (3, 6), (3, 8), (4, 5), (4, 6), (4, 8)\}$  und  $R_1 = \{(2, 6), (2, 8), (3, 6), (4, 8)\}$ .
2. Die Relation  $R_2 = \text{ist\_verheiratet\_mit}$  ist eine Teilmenge des 2-stelligen Mengenproduktes *Männer*  $\times$  *Frauen* (Sorten) mit z.B. („Anna Müller“, „Ernst Huber“)  $\in R_2$ .
3. Die 3-stellige Relation  $R_3 = \text{Adresse}$  ist ein Teil des Mengenproduktes *Straße*  $\times$  *PLZ*  $\times$  *Ort*. Dazu könnten z.B. die folgenden 3-Tupel gehören: („Sudetenstr. 16“, 83022, „Rosenheim“), („Westendstr. 4a“, 83043, „Bad Aibling“).

Je höher die Stelligkeit einer Relation ist, desto übersichtlicher erscheint es, sie in Tabellenform zu schreiben. Die Sorten schreiben wir dabei in einer eigenen Kopfzeile. Unser letztes Beispiel könnte dann etwa so aussehen wie in Tabelle 4.5 gezeigt.

**Tabelle 4.5.** Eine dreistellige Relation als Tabelle

<i>Straße</i>	<i>PLZ</i>	<i>Ort</i>
Sudetenstr. 16	83022	Rosenheim
Westendstr. 4a	83043	Bad Aibling
Römerstr. 11	80333	München

**Relationales Modell.** Ein *relationales Modell* besteht aus einer Menge von Tabellen  $\{T_1, \dots, T_n\}$ , die jeweils aus einer (unbeschränkten) Menge von Datensätzen (Tupeln) mit (innerhalb einer Tabelle) gleicher Struktur bestehen. Für eine dieser Tabellen  $T \in \{T_1, \dots, T_n\}$  gilt also  $T_i = \{d_1, \dots, d_k\}$ . Ein Datensatz  $d_k$  enthält wiederum eine Reihe von Attributwerten:  $d_k = (x_{k1}, \dots, x_{km})$ . Die Struktur einer Tabelle wird in der Kopfzeile durch eine Liste von Attributnamen  $(a_1, \dots, a_m)$  festgelegt (siehe Tabelle 4.6).

**Tabelle 4.6.** Struktur der Tabelle *Autor*

$a_1 = \text{Name}$	$a_2 = \text{Vorname}$	$a_3 = \text{Land}$	
$d_1 = ($ „Høeg“,	„Peter“,	„Dänemark“	)
$d_2 = ($ „Knauth“,	„Herbert“,	„Deutschland“	)
$d_3 = ($ „Kühnel“,	„Siegfried“,	„Deutschland“	)

**Wertemengen.** Jedes Attribut  $a_i$  kann (wie oben bereits ausgeführt) Werte aus einer bestimmten Wertemenge (*Sorte* oder *Typ*)  $S_i$  annehmen. Zum Beispiel gilt in der Tabelle  $T_2 = \text{Verlag}$ :

- $S_1 = \text{Text}$  (als Standardsorte);
- $S_2 = \text{Verlagsname} = \{\text{Springer, Hanser, Oldenbourg, O'Reilly, ...}\};$
- $S_3 = \text{Ortsbezeichnung} = \{\text{München, Wien, Zürich, New York, ...}\};$
- $S_4 = \text{Jahresangabe} = \{1500, 1501, 1502, \dots, 9999\}.$

**Schema.** Die Struktur einer Tabelle  $T$  (bzw. einer Relation  $R$ ) wird sehr kompakt durch das so genannte *Schema* beschrieben:

$$T(a_1:S_1, \dots, a_m:S_m),$$

z.B. *Autor*(Name:*Text*, Vorname:*Text*, Land:*Landname*).

**Punktnotation.** Falls man es mit mehreren Tabellen zu tun hat, wird zur Vermeidung von Unklarheiten die Tabelle, zu der das Attribut gehört, durch einen Punkt getrennt dem Attributnamen vorangestellt.  $T.a_i$  bezeichnet somit den Namen des  $i$ -ten Attributs der Tabelle  $T$ , beispielsweise *Autor.Name* oder *Buchtitel.Verlag*. Dieselbe Punktnotation verwenden wir aber auch zur Auswahl eines Attributwertes aus einem Tupel: Falls  $r = (x_1, \dots, x_n)$  und  $r \in R$ , wobei  $R(a_1, \dots, a_n)$  das Schema von  $R$  ist, dann ist  $r.a_i = x_i$  eine wahre Aussage. So gilt z.B. in der Tabelle 4.6:  $d_1.a_2 = \text{Peter}$ .

**Schlüssel.** Wenn wir annehmen, dass es in unserer obigen Tabelle *Autoren* zwei (verschiedene) deutsche Autoren mit dem Namen „Hans Meier“ gibt (siehe Tabelle 4.6), stoßen wir beim Zugriff auf das Problem, dass diese beiden Datensätze nicht unterscheidbar sind. Will man beispielsweise die Anzahl der Buchtitel eines der beiden Autoren feststellen, so werden automatisch auch die Bücher des anderen Hans Meier mitgezählt. Jede Tabelle (Relation) muss also ein Attribut oder eine Kombination von Attributen enthalten, die sicherstellen, dass auf jeden einzelnen Datensatz zugegriffen werden kann. Ein solches Attribut bzw. eine solche Kombination von Attributen heißt *Schlüssel* der Tabelle  $T$  bzw. der Relation  $R$ . Mathematisch ausgedrückt gilt (im Fall eines einzelnen Attributs als Schlüssel):

$$a_i \text{ heißt } \text{Schlüssel einer Relation } R \Leftrightarrow \text{für zwei Tupel } r, s \in R \\ \text{mit } r = (r_1, \dots, r_m) \text{ und } s = (s_1, \dots, s_m) \text{ gilt: } r_i = s_i \Rightarrow r = s.$$

Zumindest die Kombination *aller* Attribute sollte auf jeden Fall ein Schlüssel der Tabelle sein. Andernfalls enthält die Tabelle identische Datensätze. Im Fall unserer beiden Autoren namens „Meier“ müssen wir also mindestens ein neues Attribut einführen, um einen Schlüssel zu erhalten. Da die Eindeutigkeit „natürlicher“ Attribute selten absolut sicher ist, greift man in der Datenbanktechnik meist auf *künstliche Schlüsselattribute* (Identifikationsnummern) zurück (siehe Kemper (1999)), die per definitionem eindeutig sind. Wir ändern also das Schema unserer Tabelle *Autor* wie folgt ab:

*Autor (Autor\_nr: Text, Name: Text, Vorname: Text, Land: Ländername).*

Damit haben wir den künstlichen Schlüssel *Autor\_nr* eingeführt. In der tabellarischen Darstellung von Relationen unterstreichen wir ab jetzt die Schlüsselattribute, falls diese von Bedeutung sind.

In der einschlägigen Literatur wird oft zwischen *Superschlüssel*, *Schlüsselkandidat* und *Primärschlüssel* unterschieden. Korth u. Silberschatz (1991) z.B. verwenden die Bezeichnung *superkey* für jede Kombination von Attributen mit unserer obigen Schlüsseleigenschaft, *candidate key* für minimale Superschlüssel und *primary key* für die (aufgrund der Entscheidung des Datenbankentwicklers) tatsächlich zum Zugriff verwendeten Schlüsselkandidaten. Ob man die Schüler mit dieser Unterscheidung belasten soll, erscheint zumindest zweifelhaft.

### 4.2.3 Normalformen des relationalen Modells

Ohne vorausgehende ER-Modellierung sind die ersten Entwürfe eines relationalen Modells oft in mehrfacher Hinsicht verbessерungsbedürftig. Um diese Schwachstellen zu beseitigen, gibt es eine Reihe von Kriterien, welche die Brauchbarkeit der Datenstruktur sicherstellen. Traditionell werden solche Kriterien in der Datenbanktechnik als *Normalformen* bezeichnet. Wir beschränken uns im allgemein bildenden Unterricht auf die (wichtigsten) ersten drei Normalformen.

**Erste Normalform (1NF).** Eine Tabelle ist in erster Normalform, falls alle Attribute nur *atomare Werte* annehmen können. Mengen, Aufzählungstypen oder Wiederholungsgruppen dürfen also nicht als Werte der Attribute auftreten. Nicht in erster Normalform befindet sich z.B. die Relation aus Tabelle 4.7.

**Tabelle 4.7.** Verletzung der ersten Normalform

Name	Vorname	Adresse
Müller	Anna	Sudetenstr. 18, 83222, Rosenheim
Huber	Karl	Knallerweg 19, 86321, Ganselham

Für eine Überführung in die erste Normalisierung müsste das Attribut *Adresse* daher in drei Attribute *Straße*, *PLZ* und *Ort* mit jeweils atomaren Werten aufgeteilt werden.

**Zweite Normalform (2NF).** Beim ersten Entwurf eines Bibliothekssystems könnte beispielsweise die in Tabelle 4.8 gezeigte Relation *Bücher* entstanden sein.

**Tabelle 4.8.** Verletzung der 2. Normalform

Titel_nr	Titel	Autor	Exemplar_nr	Ort
1222	Fräulein Smilla	Peter Høeg	01	2/3/12/3
1222	Fräulein Smilla	Peter Høeg	02	2/3/12/4
1222	Fräulein Smilla	Peter Høeg	03	2/3/12/5
1333	Das Parfüm	Patrick Süßkind	01	2/3/15/2
1333	Das Parfüm	Patrick Süßkind	02	2/3/15/3

Bei der Arbeit mit dieser Tabelle treten u.a. die folgenden Probleme auf:

- **Datenredundanz:** Für jedes Exemplar werden Titel und Autor erneut eingetragen, obwohl diese Informationen bereits durch das erste Exemplar festgelegt sind.
- **Ungewollter Datenverlust:** Sind vortübergehend (etwa wegen Beschädigung) keine Exemplare eines Buches mehr vorhanden, so gehen alle Informationen über Titel und Autor verloren. Bei der Neuanschaffung müssen diese Informationen wieder aufgenommen werden.
- **Mögliche Inkonsistenz:** Falls bei der Eintragung eines neuen Exemplars von „Das Parfüm“ die Datentypistin beispielsweise aus Versehen den Autor „Edgar Wallace“ einträgt, existiert dieser Titel unter zwei verschiedenen Autoren.

Das Problem geht offensichtlich darauf zurück, dass für die gesamte Tabelle (mindestens) die Attributkombination *Titel\_nr* und *Exemplar\_nr* als Schlüssel verwendet werden muss, während für einige Attribute wie *Titel* und *Autor* bereits *Titel\_nr* als Schlüssel ausreicht.

Um dieses Problem kompakter beschreiben zu können, führen wir den Begriff der *funktionalen Abhängigkeit* ein: Ein Attribut  $a_i$  einer Relation  $R$  heißt *funktional abhängig* von einem Attribut  $a_k$ , falls für alle Tupel  $r, s \in R$  gilt:

$$r_k = s_k \Rightarrow r_i = s_i.$$

Wir schreiben dann kurz:  $a_k \rightarrow a_i$ . Für die funktionale Abhängigkeit von einer Kombination zweier Attribute gilt dann entsprechend:

$$a_k a_m \rightarrow a_i \Leftrightarrow \forall r, s \in R : r_k = s_k \wedge r_m = s_m \Rightarrow r_i = s_i.$$

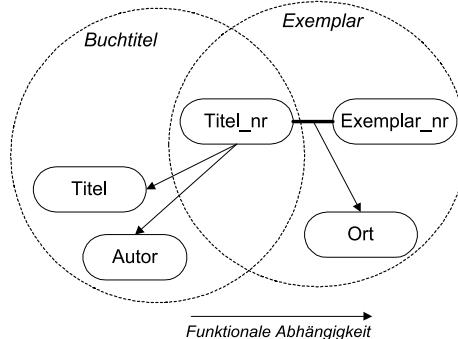
In unserem Beispiel sind die Attribute *Autor*, *Titel* von *Titel\_nr* funktional abhängig. Dagegen hängt *Ort* funktional nur von der Kombination aus *Titel\_nr* und *Exemplar\_nr* ab:

$$\text{Titel\_nr} \rightarrow \text{Titel}, \text{Titel\_nr} \rightarrow \text{Autor}, \text{Titel\_nr}, \text{Exemplar\_nr} \rightarrow \text{Ort}.$$

Als Schlüssel in unserer Tabelle ist daher nur die Kombination aus *Titel\_nr* und *Exemplar\_nr* brauchbar. Das Problem liegt offensichtlich darin, dass einige Attribute bereits von einem Teil der Schlüsselkombination funktional abhängig sind. Genau diese Abhängigkeit wird durch die 2. Normalform ausgeschlossen: Eine Tabelle befindet sich in der zweiten Normalform, wenn

- sie die erste Normalform erfüllt und zusätzlich
- jedes Attribut, das zu keiner Schlüsselkombination gehört, nur von der *gesamten Schlüsselkombination*, nicht jedoch bereits von einem Teil davon funktional abhängig ist.

Die Umformung in die zweite Normalform bringt daher eine Abspaltung neuer Tabellen mit den kritischen Attributen mit sich, wobei jeweils der Teilschlüssel, von dem die problematische Abhängigkeit vorlag, als neuer Schlüssel verwendet wird. Die anderen Teile des ursprünglichen Schlüssels verbleiben dennoch in der Ausgangstabelle, um die ursprünglichen Zuordnungen zu erhalten.



**Abb. 4.3.** Aufspaltung einer Tabelle zur Einhaltung der 2. Normalform

In unserem Beispiel wird eine Tabelle *Buchtitel* mit den Attributen *Titel\_nr*, *Titel* und *Autor* abgespalten. Der verbleibende Rest von *Bücher* wird sinnvollerweise in *Exemplare* umbenannt. Schlüssel von *Exemplare* bleibt weiterhin die Kombination aus *Exemplar\_nr* und *Buchtitel\_nr* (siehe Abb. 4.3 und Tabelle 4.9).

**Tabelle 4.9.** Aufspaltung der Tabelle *Bücher* in *Buchtitel* und *Exemplare*

<i>Buchtitel</i>			<i>Exemplare</i>		
<i>Titel_nr</i>	<i>Titel</i>	<i>Autor</i>	<i>Titel_nr</i>	<i>Ex_nr</i>	<i>Ort</i>
1222	Fräulein Smillas ...	Peter Høeg	1222	01	2/3/12/3
1333	Das Parfüm	Patrick Süßkind	1222	02	2/3/12/4
			1222	03	2/3/12/5
			1333	01	2/3/15/2
			1333	02	2/3/15/3

**Dritte Normalform (3NF).** Trotz Einhaltung der 2. Normalform kann es immer noch zu Datenredundanzen (mit den damit verbundenen bereits genannten Problemen) kommen. Wir betrachten dazu unsere anfängliche Tabelle *Kunde*, die wir um einen künstlichen Schlüssel *Kunde\_nr* ergänzt haben (siehe Tabelle 4.10).

**Tabelle 4.10.** Verletzung der 3. Normalform

<i>Kunde</i>					
<u>Kunde_nr</u>	Name	Vorname	Straße	PLZ	Ort
00012	Müller	Anna	Sudetenstr. 18	83222	Rosenheim
00013	Huber	Karl	Knallerweg 19	86321	Ganselham
00014	Meier	Amelie	Körberweg 18	83222	Rosenheim
00015	Hanser	Kurt	Kästnerstr. 10 A	83222	Rosenheim

Wegen des atomaren Schlüssels ist die Tabelle in 2. Normalform. Das Attribut *Ort* enthält jedoch redundante Daten, da bereits das Attribut *PLZ* die Werte von *Ort* eindeutig festlegt. Aus der Sicht funktionaler Abhängigkeit gilt hier:

$$\text{Kunde\_nr} \rightarrow \text{PLZ} \rightarrow \text{Ort}$$

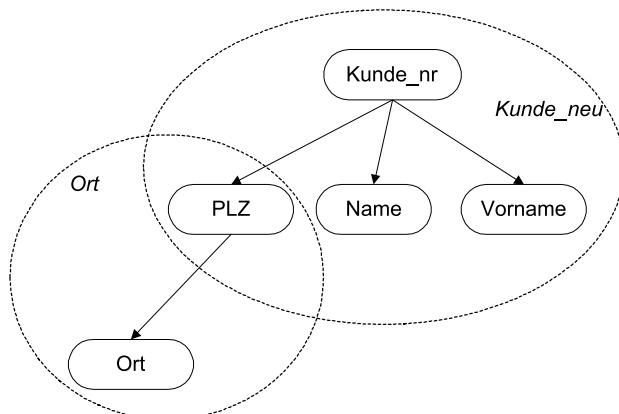
Eine solche Kette funktionaler Abhängigkeiten heißt *transitive funktionale Abhängigkeit*. Genau diese Eigenschaft der Tabelle verursacht die Datenredundanzen. Wir fordern also für die dritte Normalform: Eine Tabelle ist in dritter Normalform, wenn

- sie sich in zweiter Normalform befindet und
- kein Nichtschlüsselattribut *transitiv abhängig* von einem Schlüsselattribut ist.

Die Umformung in die dritte Normalform zwingt zu einer erneuten Aufspaltung, wobei aus dem „mittleren“ Schlüssel in der Kette der Abhängigkeiten zusammen mit den davon abhängigen Attributen eine neue Tabelle gebildet wird. Das neue Schlüsselattribut verbleibt aber auch in der ursprünglichen Tabelle, um Informationsverluste zu vermeiden. Die davon funktional abhängigen Attribute werden jedoch aus der ursprünglichen Tabelle ausgegliedert. In unserem Beispiel zerlegen wir *Kunde* in die beiden neuen Relationen *Kunde\_neu* und *Ort* (siehe Tabelle 4.11 und Abb. 4.4).

**Tabelle 4.11.** Überführung der Tabelle *Kunde* in die 3. Normalform durch Aufteilung

<i>Kunde_neu</i>					<i>Ort</i>	
<u>Kunde_nr</u>	Name	Vorname	PLZ	Straße	<u>PLZ</u>	Name
00012	Müller	Anna	83222	Sudetenstr. 18	83222	Rosenheim
00013	Huber	Karl	86321	Knallerweg 19	86321	Ganselham
00014	Meier	Amelie	83222	Körberweg 18		
00015	Hanser	Kurt	83222	Kästnerstr. 10 A		



**Abb. 4.4.** Aufspaltung einer Tabelle zur Einhaltung der 3. Normalform

#### 4.2.4 Umsetzung von ER-Modellen in relationale Modelle

Zum Vergleich wollen wir jetzt unser ER-Modell aus 4.2.1 in ein System von Relationen umsetzen. Die Umwandlung erfolgt schematisch mit Hilfe einfacher Regeln.

**Klassen und Objekte.** Entitätsmengen bzw. Klassen werden zu Tabellen und Attribute zu Spalten. In jeder Tabelle wird ein Attribut (oder eine Kombination von Attributen) als Schlüssel definiert. Die Objekte entsprechen dann den Zeilen (Datensätzen) der Tabellen. Aus den Klassen unseres ER-Modells aus Abb. 4.1 erhalten wir die folgenden Tabellenschemata:

*Buchtitel* (Buchtitel\_nr, Titel, Autor, Verlag, Ort, Jahr, ISBN, Preis),  
*Exemplar* (Exemplar\_nr, Bezeichnung, Zustand, Aufnahmedatum),  
*Standort* (Standort\_nr, Zimmer, Regal, Fach, Platz),  
*Autor* (Autor\_nr, Name, Vorname, Land, andere Werke),  
*Kunde* (Kunde\_nr, Name, Vorname, Geburtsdatum, Straße, PLZ, Ort).

**Beziehungen.** Die Umsetzung von Beziehungen in das Tabellensystem hängt von ihrer Kardinalität ab:

- Kardinalität 1:1. Diese Beziehungen können in eine der beiden Tabellen der beteiligten Objekte eingebaut werden, indem man das Schlüsselattribut der anderen beteiligten Tabelle aufnimmt. Dieses Attribut heißt dort *Fremdschlüssel*. Die 1:1-Beziehung *steht\_in* wird über den Fremdschlüssel *Standort\_nr* in die Tabelle *Exemplar* aufgenommen: *Exemplar* (Exemplar\_nr, Bezeichnung, Zustand, Aufnahmedatum, Standort\_nr)

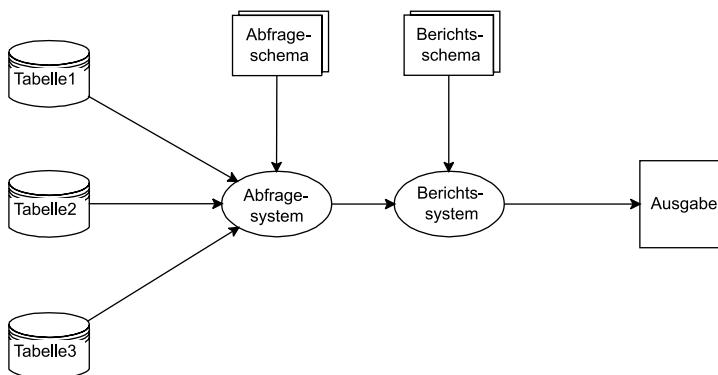
- Kardinalität 1:n. Solche Beziehungen werden umgewandelt, indem man das Schlüsselattribut der 1-Seite als zusätzliches Attribut in der Tabelle der n-Seite aufnimmt. Unsere 1:n-Relation *ist\_vorhanden* wird durch den Fremdschlüssel *Buchtitel\_nr* in der Tabelle *Exemplare* realisiert: *Exemplar* (Exemplar\_nr, Bezeichnung, Zustand, Aufnahmedatum, *Buchtitel\_nr*)
- Kardinalität m:n. Hier entsteht aus der Beziehung eine eigene Tabelle, die mindestens die Schlüsselfelder der beiden beteiligten Objekttabellen enthält. So wird aus unserer Beziehung *verfasst\_von* nun die Tabelle *Verfasst\_von* (*Buchtitel\_nr*, *Autor\_nr*).

## 4.3 Abfragen und Berichte

Die beste Datenhaltung bleibt sinnlos, solange man nicht wenigstens minimale Operationen wie Ein- und Ausgabe zulässt. Wir wollen uns hier auf dieses Minimum beschränken. Bezüglich der Eingabe gehen wir von einer komfortablen Benutzerschnittstelle aus, die ein direktes Eingeben und Editieren der Daten in den Tabellen erlaubt (wie z.B. unter *MS-Access*<sup>®</sup>). Für die Ausgabe wollen wir uns mit Techniken zum Entwurf von Abfragen beschäftigen. Am Gymnasium sollte man dabei unserer Ansicht nach vor allem Wert auf ein solides mathematisches Fundament in Form von Grundzügen der Relationenalgebra legen. Ob man sich mit der Syntax von Abfragesprachen wie SQL beschäftigt, hängt von den näheren Umständen wie den Interessen der Schüler oder dem benutzten System ab.

### 4.3.1 Funktionsprinzipien

Anfangs muss geklärt werden, was bei der Definition von Verarbeitungsvorgängen wie Abfragen oder Berichten überhaupt vor sich geht. Abb. 4.5 zeigt ein funktionales Modell dafür: Eine Abfrage erscheint als Auswahlprozess, der aus dem gesamten Datenbestand einer oder mehrerer Tabellen eine durch die Vorgaben eines speziellen Dokumentes (*Abfrageschema*) bestimmte Teilmenge auswählt und diese an das Berichtssystem weitergibt. Letzteres erstellt daraus ein Berichtsformular, dessen Format durch ein weiteres spezielles Dokument (*Berichtsschema*) festgelegt wird. Die Vorgehensweise bei der Erstellung des Berichtsschemas hängt stark vom verwendeten System ab, weshalb wir hier nicht weiter darauf eingehen wollen. Die Erstellung des Abfrageschemas erscheint im Hinblick auf Allgemeinbildung dagegen weitaus ergiebiger.



**Abb. 4.5.** Funktionsprinzip von Abfragen und Berichten

### 4.3.2 Relationale Algebra

Nachdem das Funktionsprinzip von Abfragen grundsätzlich geklärt ist, erhebt sich die Frage, wie man bei der Festlegung des Abfrageschemas vorzugehen hat. Technisch gesehen verwendet man dafür spezielle *Abfragesprachen* wie SQL (siehe Abschnitt 4.3.3), deren Sprachkonstrukte entweder manuell eingegeben oder durch interaktive Kombination von Auswahlkriterien automatisch generiert werden. Für das Verständnis der Auswahlprinzipien von Abfragen erweist es sich als sehr vorteilhaft, vor (oder sogar anstelle) ihrer Behandlung einige mathematische Grundlagen mit Hilfe des relationalen Modells zu erarbeiten. Die folgende Beschreibung in mathematischer Notation empfehlen wir allerdings nur für das Gymnasium und dort unter günstigsten Voraussetzungen (Pflichtfach, gute mathematische Vorbildung) frhestens ab Jahrgangsstufe 9, ansonsten erst ab Jahrgangsstufe 10. Alternativ könnte man die folgenden Operationen auch exemplarisch an Beispielen oder rein visuell einführen.

**Projektion.** Mit Hilfe der Projektion  $P(a_1, \dots, a_k)$  kann man die Attribute (Spalten)  $a_1, \dots, a_k$  aus einer Relation (Tabelle)  $R$  extrahieren:

$$P(a_1, \dots, a_k)(R) = \{(x_1, \dots, x_k) \mid r \in R \wedge r.a_i = x_i\},$$

so z.B. aus der Tabelle *Buchtitel* (siehe Tabelle 4.9):

$$\begin{aligned} P(\text{Titel}, \text{Autor})(\text{Buchtitel}) = \\ \{(\text{„Fräulein Smillas“}, \text{„Peter Høeg“}), (\text{„Das Parfüm“}, \text{„Patrick Süßkind“})\}. \end{aligned}$$

**Auswahl.** Mit der sehr mächtigen Operation  $W_{(P)}$  kann die Teilmenge aller Tupel einer Relation  $R$  ausgewählt werden, die eine bestimmte Aussageform  $P$  erfüllen:

$$W_{(P)}(R) = \{r \mid r \in R \wedge P(r) = \text{wahr}\}.$$

Wir vermeiden hier bewusst die Bezeichnung „Selektion“, um keine Interferenzen mit dem SQL-Wort SELECT (siehe Abschnitt 4.3.3) zu erzeugen. Mit  $P = [\text{PLZ} = 83222]$  extrahiert man beispielsweise aus *Kunde\_neu* (siehe Tabelle 4.11):

$$\begin{aligned} W(P)(\text{Kunde\_neu}) &= \\ &\{(00012, „Müller“, „Anna“, 83222, „Sudetenstr. 18“), \\ &(00014, „Meier“, „Amelie“, 83222, „Körberweg 18“), \\ &(00015, „Hanser“, „Kurt“, 83222, „Kästnerstr. 10 A“)\}. \end{aligned}$$

**Kreuzprodukt.** Das Kreuzprodukt kombiniert jedes Tupel aus  $R$  mit allen Tupeln aus  $S$ , wobei das Schema der neuen Tabelle durch Vereinigung aus den beiden Ausgangsschemata entsteht:

$$R \times S = \{(r_1, \dots, r_n, s_1, \dots, s_m) \mid (r_1, \dots, r_n) \in R \wedge (s_1, \dots, s_m) \in S\}.$$

Für einzelnen Elemente des Kreuzproduktes schreiben wir auch kurz  $rs$ :

$$rs = (r_1, \dots, r_n, s_1, \dots, s_m) \text{ mit } r = (r_1, \dots, r_n) \in R \text{ und } s = (s_1, \dots, s_m) \in S.$$

Zur Veranschaulichung zeigt Tabelle 4.12 das Kreuzprodukt der Tabellen *Kunde\_neu* und *Ort* aus Tabelle 4.11. Die Attribute ohne Angabe der Ursprungstabelle stammen dabei aus *Kunde\_neu*.

**Tabelle 4.12.** Das Kreuzprodukt der Tabellen *Kunde\_neu* und *Ort*

<i>Kunde_neu</i>		<i>Ort</i>				
Kunde_nr	Name	Vorname	PLZ	Straße	Ort.PLZ	Ort.Name
00012	Müller	Anna	83222	Sudetenstr. 18	83222	Rosenheim
00013	Huber	Karl	86321	Knallerweg 19	83222	Rosenheim
00014	Meier	Amelie	83222	Körberweg 18	83222	Rosenheim
00015	Hanser	Kurt	83222	Kästnerstr. 10 A	83222	Rosenheim
00012	Müller	Anna	83222	Sudetenstr. 18	86321	Ganselham
00013	Huber	Karl	86321	Knallerweg 19	86321	Ganselham
00014	Meier	Amelie	83222	Körberweg 18	86321	Ganselham
00015	Hanser	Kurt	83222	Kästnerstr. 10 A	86321	Ganselham

Auf den ersten Blick scheint diese Operation ein ziemlich sinnloses Ergebnis zu liefern. So werden z.B. Kundennamen mit den Postleitzahlen von Orten verknüpft, an denen sie nicht wohnen oder Postleitzahlen mit Orten, zu denen sie nicht gehören. Die Daseinsberechtigung des Kreuzproduktes ergibt sich erst in Verbindung mit den folgenden Join-Operationen, denen es als Grundmenge dient.

**Equi-Join.** Der Equi-Join liefert die Menge aller Tupel aus  $R \times S$ , bei denen die Werte der angegebenen Attribute identisch sind:

$$\begin{aligned} R \times S [R.a_i = S.b_k] &= \\ &\{(r_1, \dots, r_n, s_1, \dots, s_m) \mid (r_1, \dots, r_n) = r \in R \wedge (s_1, \dots, s_m) = s \in S \wedge r.a_i = s.b_k\}. \end{aligned}$$

Dabei kann eine beliebige Anzahl von Gleichheitsforderungen angegeben werden, die dann alle gleichzeitig zu erfüllen sind. Das Ergebnis des Equi-Joins

$$Kunde\_neu \quad Ort [Kunde\_neu.PLZ = Ort.PLZ]$$

der beiden Tabellen aus 4.11 ist beinahe identisch mit Tabelle 4.10, mit der kleinen Abweichung, dass das Feld PLZ (als *Kunde\_neu.PLZ* und *Ort.PLZ*) zweifach auftritt. Mit Hilfe eines Equi-Joins kann man also die Aufspaltung von Tabellen (z.B. aus Gründen der Normalisierung) zum Zwecke der Datenextraktion wieder „rückgängig machen“.

Eine Verallgemeinerung des Equi-Join ist der Theta-Join, bei dem man anstatt einer Gleichheitsforderung eine beliebige Vergleichsoperation wie  $\geq$ ,  $<$  etc. zulässt. Darauf wollen wir jedoch nicht weiter eingehen.

**Natural Join.** Die Ergebnismenge des Natural-Join  $R \otimes S$  der beiden Relationen  $R$  und  $S$  besteht aus allen Tupeln aus  $R \times S$ , bei denen diejenigen Attribute, die in  $R$  und in  $S$  vorkommen, den gleichen Wert haben. Der Natural-Join ist also ein Equi-Join, der alle „gemeinsamen“ Attribute zweier Tabellen erfasst, ohne dass man diese explizit angeben muss. Er liefert die Kombination von Tabellen, die man am häufigsten benötigt. Die gemeinsamen Attribute werden dann sinnvollerweise nur noch einmal in der Tabelle aufgeführt. In unserem Beispiel ist

$$Kunde\_neu \otimes Ort = Kunde\_neu \quad Ort [Kunde\_neu.PLZ = Ort.PLZ],$$

da PLZ das einzige gemeinsame Attribut der beiden Tabellen darstellt.

Eine gewisse Problematik liegt in der Bestimmung der gemeinsamen Attribute, falls diese nicht völlig identische Namen haben, wie z.B. bei den folgenden Tabellen:

$$\begin{aligned} & \text{Buchtitel } (\underline{\text{Buchtitel\_nr}}, \text{Titel}, \text{Autor}, \text{Verlag}, \text{Ort}, \text{Jahr}, \text{ISBN}, \text{Preis}), \\ & \text{Autor } (\underline{\text{Autor\_nr}}, \text{Name}, \text{Vorname}, \text{Land}, \text{andere}). \end{aligned}$$

Bei der Berechnung von *Buchtitel*  $\otimes$  *Autor* ist nicht klar, ob die Attribute *Buchtitel*.*Autor* und *Autor*.*Name* als identisch zu betrachten sind. Im Zweifelsfall sollte man daher auf einen Equi-Join zurückgreifen, bei dem die betreffenden Attribute explizit angegeben werden können.

Die folgenden drei Operationen (Vereinigung, Durchschnitt, Differenz) liefern nur dann wieder eine sinnvolle Relation, wenn man sich beim Ergebnis auf den Durchschnitt der Schemata von  $R$  und  $S$ , also auf die gemeinsamen Attribute, beschränkt. Der Einfachheit halber vereinbaren wir, diese drei Operationen nur auf Relationen mit *identischen Schemata* anzuwenden, z.B. reduzieren wir dazu die Tabellen *Kunde* und *Autor* aus Abschnitt 4.2.4 auf die gemeinsamen Attribute *Name* und *Vorname*.

**Vereinigung.** Die Vereinigungsmenge zweier Relationen besteht aus allen Tupeln, die entweder der einen Relation oder der anderen Relation angehören:

$$R \cup S = \{ r \mid r \in R \vee r \in S \}.$$

**Durchschnitt.** Der Durchschnitt zweier Relationen besteht aus allen Tupeln, die sowohl der einen Relation als auch der anderen Relation angehören:

$$R \cap S = \{ r \mid r \in R \wedge r \in S \}.$$

In unserem Beispiel liefert *Kunde*  $\cap$  *Autor* die Namen und Vornamen aller Personen, die sowohl als Autor wie auch als Kunde registriert sind.

**Differenz.** Die Differenz von  $R$  und  $S$  ist die Menge aller Tupel, die  $R$ , aber nicht  $S$  angehören:

$$R - S = \{ r \in R \wedge r \notin S \}.$$

*Kunde*  $-$  *Autor* ergibt daher die Namen und Vornamen aller Personen, die Kunde, aber nicht Autor sind.

### 4.3.3 Abfragen mit SQL

Zur Durchführung von Abfragen in realen Datenbanksystemen verwendet man allerdings nicht den oben beschriebenen mathematischen Relationenkalkül, sondern spezielle Abfragesprachen. Am meisten Verbreitung darunter hat wohl die Abfragesprache *SQL (Structured Query Language)* gefunden, die aus der von IBM Anfang der 70er Jahre für das Datenbanksystem R entwickelten Sprache *Sequel* entwickelt wurde (siehe Korth u. Silberschatz (1991)). Ob man die Syntax von SQL mit den Schülern explizit behandeln soll, ist zweifelhaft. Wenn der oben eingeführte Relationenkalkül zur Verfügung steht, erscheinen die SQL-Abfragen nur noch als Übersetzung der entsprechenden mathematischen Ausdrücke, weshalb man sich ohne weiteres auf die interaktive Definition von Abfragen beschränken kann (etwa in *MS-Access*<sup>®</sup>). Beim folgenden kurzen Abriss der wichtigsten Ausdrücke von SQL unterscheiden wir nach der Anzahl der Ausgangstabellen. Außerdem verzichten wir auf Beispiele, da sich diese bei den entsprechenden Operationen des Relationenkalküls im vorigen Abschnitt finden.

#### 1. Abfragen aus einer Tabelle

**SELECT-Anweisung.** Die Hauptkomponente einer SQL-Abfrage ist die SELECT-Anweisung, die Daten extrahiert und auf einen Standardausgabekanal weiterleitet. Ihre Syntax lautet:

```
SELECT <Attributliste>
FROM <Tabellenliste>
WHERE <Bedingungsliste>.
```

Eine Abfrage nach den Titeln und dem Erscheinungsjahr aller Bücher, die von Peter Høeg nach 1990 erschienen sind, wird in SQL folgendermaßen formuliert:

```
SELECT Titel, Jahr
FROM Buchtitel
WHERE Autor = 'Peter Høeg' AND Jahr > 1990
```

**Projektion.** Ein Vergleich mit der Projektion aus Abschnitt 4.3.2 zeigt die Formulierung einer Projektion in SQL:

```
SELECT a, b
  FROM T
```

ist offensichtlich gleichbedeutend mit der Projektion  $P(a, b)(T)$ . Diese Anweisung liefert also die Spalten (Attribute)  $a, b$  aus der Tabelle  $T$ . Ein \* gibt an, dass man alle Attribute der Tabelle ausgeben will.

**Auswahl.** Die Anweisung

```
SELECT *
  FROM T
 WHERE P
```

ist gleichbedeutend mit der Auswahl  $W_{(P)}(T)$ . Wir erhalten damit alle Datensätze, für welche die Aussageform  $P$  wahr ist.

## 2. Verknüpfung mehrerer Tabellen

**Kartesisches Produkt.** Die Angabe mehrerer Tabellen nach FROM ohne einschränkende WHERE-Klausel (s.unten) liefert das kartesische Produkt der angegebenen Tabellen:

```
SELECT *
  FROM S, T
```

ist daher äquivalent mit  $S \times T$ . Zur Projektion von Spalten aus dem Produkt kann auch hier eine Punktnotation verwendet werden:

```
SELECT Buchtitel.Titel, Buchtitel.Jahr, Autor.*
  FROM Buchtitel, Autor.
```

**Equi-Join.** Der Equi-Join wird durch die Angabe der Vergleichsbedingung hinter WHERE realisiert:

```
SELECT *
  FROM R, S
 WHERE R.a = S.b
```

ist gleichbedeutend mit  $R \text{ } S [R.a = S.b]$ .

## 4.4 Datenmodellierung eines Fahrplansystems

Abschließend wollen wir nach unserem Bibliothekssystem einen weiteren Vorschlag für ein Unterrichtsprojekt zur Datenmodellierung unterbreiten, nämlich die Simulation eines Systems zur Verwaltung von Fahrplandaten der Bundesbahn.

### 4.4.1 Problemstellung

Am besten holen wir uns ein Fahrplanplakat der Deutschen Bahn AG (siehe Abb. 4.6) aus dem nächsten Bahnhof und untersuchen es eingehend.

Zugnr.	Abfahrt	Nach	Gleis
IR 1223	11:38	17:10 Karlsruhe (Gleis 13), 12:03 Augsburg (S), →, 10 ↵ 12345	12
		13:20 Ulm, 15:10 Stuttgart (Gleis 2) (S), →	

**Abb. 4.6.** Ausschnitt eines Fahrplanplakates

Am Ende unseres Projektes soll mit Hilfe eines relationalen Datenbanksystems eine ähnliche Ausgabe generiert werden.

### 4.4.2 Informelle Beschreibung

Anhand eines beispielhaften Zuglaufs beschreiben wir alle zu entnehmenden Informationen aus dem Fahrplan und diskutieren Randbedingungen, Abhängigkeiten, Wertebereiche, etc.

- Zugnummer: Zugart (2 Buchstaben aus RB, RE, SE, IR, IC, EC, D), vierstellige Identifikationszahl, Zusatzinformationen: Schlaf/Liegewagen, Speisewagen, nur 2.Klasse, Wochentage 1–7,
- Abfahrt: Uhrzeit, (mod 24) kleiner als alle Ankunftszeiten,
- Nach: Zielbahnhof und Ankunftszeit (> Abfahrtszeit (mod 24)), Zwischenstationen mit: Ankunftszeit, S-Bahnanschluss, Flughafen,
- Gleis: Zweistellige Zahl, evtl. mit Zusatz a/b/c.

### 4.4.3 Datenmodellierung

Bei der Erstellung des ER-Modells drängen sich die Klassen *Zug* und *Bahnhof* geradezu auf. Weiter bemerken wir, dass einige Attribute aus dem Zuglauf (Name, S-Bahn, Flughafen) für mehrere Züge verwendbar sind, während andere (Ankunft, Abfahrt, Gleis) nur für den jeweiligen Zug gelten und daher der Beziehung *hält\_in* zugeteilt werden (siehe Abb. 4.7).

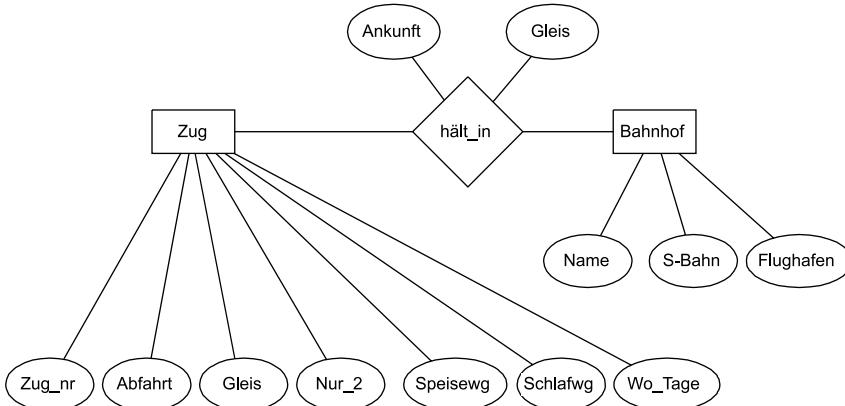


Abb. 4.7. ER-Modell des Fahrplansystems

Bezüglich der Kardinalität der Beziehung **hält\_in** stellen wir fest, dass die meisten Züge in mehr als einem Bahnhof halten, ebenso wie an einem Bahnhof viele Züge Station machen. Es handelt sich also um eine m:n-Beziehung.

#### 4.4.4 Realisierung

Gemäß unseres Schemas aus Abschnitt 4.2.4 ergeben sich aus dem ER-Modell die drei Tabellen **Zug**, **Bahnhof** und **hält\_in** mit den entsprechenden Attributen:

*Zug* (Zug\_nr, Art, Abfahrt, Gleis, Nur\_2, Speisewg, Schlafwg, Wo\_Tage),  
*Hält\_in* (Zug\_nr, Bahnhof\_nr, Ankunft, Gleis),  
*Bahnhof* (Name, S\_Bahn, Flughafen).

Mit Hilfe des Abfragegenerators erstellen wir schließlich die passende SQL-Abfrage:

```
SELECT Zug.* , Hält_in.Ankunft, Hält_in.Gleis, Bahnhof.Name,
       Bahnhof.S_bahn, Bahnhof.Flughafen;
FROM Zug, Hält_in, Bahnhof;
WHERE Hält_in.Zug_nr = Zug.Zug_nr
      AND Bahnhof.name = Hält_in.Bahnhof;
ORDER BY Zug.abfahrt, Hält_in.Ankunft;
```

Als Ziel der Abfrage geben wir einen Standardbericht an, den wir dann nach unseren Wünschen gestalten. Dazu gruppieren wir die Datensätze der drei beteiligten Tabellen nach der Zug-Nummer, bringen einige optische Verbesserungen an und erhalten schließlich das in Abb. 4.8. gezeigte Ergebnis.

Nr.	Art	Ziel	Ab	Gleis	
1121	RB	Rosenheim	11:10	11	12345
	An	Ab			
	11:20	11:25	München Ost	S	
	11:48	11:50	Grafling	S	
	12:05		Rosenheim		
1122	IR	Karlsruhe	12:02	12	123456 ΦH
	An	Ab			
	12:14	12:16	Augsburg	S	

**Abb. 4.8.** Die Ausgabe unserer Fahrplanabfrage (Ausschnitt)

Abschließend stellen wir die üblichen Überlegungen zur Wertung unseres Projektes an (siehe Abschnitt 4.3.3 aus Teil B).

# 5 Zustandsorientierte Modellierung

Nachdem wir uns im vorigen Kapitel ausführlich mit der statischen Modellierung von Datenstrukturen beschäftigt haben, wollen wir uns nun der Beschreibung von zeitlichen Abläufen mit Hilfe von Zustands-Übergangsdiagrammen zuwenden. Zur Simulation der Modelle greifen wir auf imperative Sprachen zurück, da sich diese wegen ihres (zustandsorientierten) Variablenkonzeptes besonders gut dafür eignen. Zudem vermittelt die imperative Programmierung am ehesten einen Einblick in die eigentliche „innere“ Arbeitsweise der Rechner auf der Ebene von Speicherzellen, Registern und Verarbeitungseinheiten. Da wir hier zwei der wichtigsten Kerngebiete der Informatik (Automaten und Programmierung) berühren, wollen wir besonders sorgfältig vorgehen und alle für das Verständnis notwendigen gedanklichen Schritte möglichst detailliert beschreiben.

Die Schüler befinden sich in der 9–10. Jahrgangsstufe und haben bereits einen Überblick über grundlegende Techniken der Repräsentation von Information hinter sich. Dabei haben sie einige Datenstrukturen und in umgangssprachlicher Form bereits die Strukturelemente von Algorithmen (Sequenz, Auswahl, Wiederholung) kennen gelernt.

## 5.1 Programmierung als Dilemma

Spätestens an dieser Stelle unseres Unterrichtskonzeptes werden wir gezwungen, im engeren Sinn des Wortes zu „programmieren“. Dabei stoßen wir auf das bereits in Teil B, Abschnitt 5.4.2 ausführlich geschilderte Dilemma:

- Einerseits sind Grundkenntnisse über die Funktionsweise imperativer Programme ein unverzichtbarer Baustein zum Verständnis elektronischer Informations- und Kommunikationssysteme. Diese Grundkenntnisse können aber nur erfolgreich vermittelt werden, wenn die Schülerinnen und Schüler tatsächlich selbst Programme schreiben, übersetzen lassen und so deren Ablauf verfolgen können.
- Andererseits gehören spezielle Kenntnisse über die Syntax der dabei verwendeten konkreten Programmiersprache zu Klasse 4 in Bezug auf ihre Allgemeingültigkeit (siehe Teil B, Abschnitt 5.3.1) und damit *nicht* zum Kanon allgemein bildender Lerninhalte. Ein reiner Programmierkurs muss im Rahmen eines Pflichtfachs deshalb auf jeden Fall vermieden werden.

Als Lösung dieses Problems schlagen wir vor, die Programmierung im Unterricht auf die Implementierung von Zustands-Übergangsdiagrammen zu beschränken. Damit kann sichergestellt werden,

- dass das geplante System vor der eigentlichen Codierung einigermaßen sauber modelliert wird,
- dass die behandelten Probleme in einem begrenzten Komplexitätsbereich bleiben: Einfache Syntaxspielereien werden dadurch ebenso ausgeschlossen wie sehr komplexe Aufgabenstellungen.

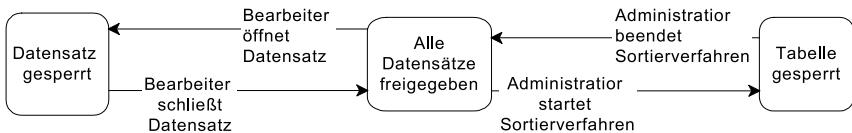
Über die Automatensicht als Meta-Ebene erhält man eine klare Systematik und eine intuitiv verständliche (operationelle) Semantik der einzelnen Sprachelemente, weil das Programm lediglich den Ablauf eines Automaten simuliert. Der prüfungsrelevante Lernstoff beschränkt sich dabei auf die Erstellung von Zustands-Übergangsdiagrammen und Grundkonzepten ihrer Umsetzung in Programme (z.B. das Variablenkonzept), während die Bedeutung spezieller Syntaxkenntnisse relativiert wird. Wir wollen den Schülern schließlich grundlegende Kenntnisse über Informations- und Kommunikationssysteme beibringen, ohne sie dabei zu Programmierern auszubilden.

## 5.2 Zustandsmodellierung

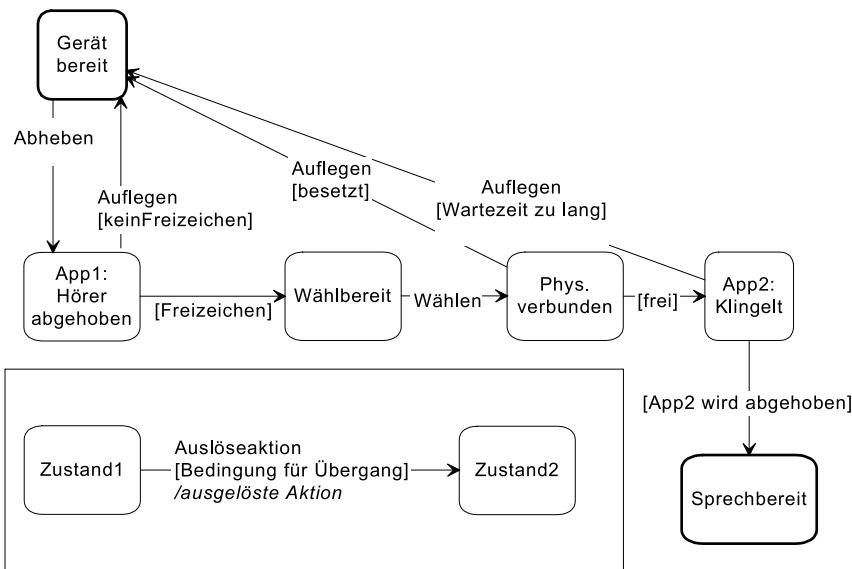
Bei der unmittelbar vorausgegangenen Behandlung der datenorientierten Modellierung haben wir uns die Sache ziemlich einfach gemacht und auf die Modellierung zeitlicher Abläufe weitgehend verzichtet. Bei der Betrachtung von Datenbanktransaktionen wurde dann offensichtlich, dass wir eine Methode benötigen, um die zeitliche Abfolge von Zuständen beschreiben zu können. Dies führt uns auf Zustands-Übergangsdiagramme. Ein System, das damit beschrieben werden kann, nennen wir im Folgenden *Automat*.

### 5.2.1 Einführung von Zustands-Übergangsdiagrammen

Zunächst stellen wir den Anschluss an die Vorkenntnisse aus dem Datenbankbereich her und verwenden einfache Zustands-Übergangsdiagramme (siehe Teil B, Abschnitt 5.4.3) zur Modellierung von Zuständen eines Datenbanksystems, z.B. hinsichtlich der Sperrung von Datensätzen oder ganzen Tabellen bei Zugriff durch mehrere Benutzer (siehe Abb. 5.1). Mit den dabei verwendeten Syntaxelementen können wir allerdings nur endliche Automaten ohne Ausgabe modellieren. Um auch den Ablauf von Automaten mit Ausgabe, Kellerautomaten oder Turingmaschinen beschreiben zu können, müssen wir unsere Notation erweitern. Am Beispiel des Aufbaus einer Telefonverbindung kann im nächsten Schritt die Einführung *bedingter Übergänge* (notiert durch eine eckig geklammerte Bedingung) veranschaulicht werden (siehe Abb. 5.2). Zusätzlich kann als dritte Markierung der Transitionen (nach einem Schrägstrich) noch der Name einer Aktion angebracht werden, die durch den Übergang ausgelöst wird.



**Abb. 5.1.** Datenbanksperren im Zustands-Übergangsmodell



**Abb. 5.2.** Zustands-Übergangsdiagramm einer Telefonverbindung

### 5.2.2 Exkurs: Beschreibung abstrakter Maschinen

Als Hintergrundinformation für Lehrkräfte und als Anregung für einen Anschlusskurs in der gymnasialen Oberstufe soll hier kurz erläutert werden, wie man den Ablauf einiger wichtiger Automaten bzw. abstrakter Maschinen mit Hilfe der oben eingeführten Zustands-Übergangsdiagramme beschreiben kann. Die Modellierung von Turing-Maschinen ermöglicht dabei (zumindest theoretisch) die Berechnung aller überhaupt berechenbaren Funktionen.

Auf der Grundlage der Notation aus Abb. 5.1 stellen wir die Argumente und Werte der Zustands-Übergangsfunktion  $\delta$  vom Zustand  $z_1$  zum Zustand  $z_2$  (siehe z.B. Broy (1998)) in unseren Diagrammen durch Tripel  $(A, B, G)$  von Transitionsmarkierungen dar, wobei  $A$  für die *auslösende Aktion*,  $B$  für die *Übergangsbedingung* und  $G$  für die *ausgelöste Aktion* steht. Die auslösende Aktion ist bei den folgenden Maschinen immer die Eingabe eines Zeichens  $k$ . Der Einfachheit halber schreiben wir daher wir kurz  $A = k$  statt  $A = [\text{EINGABE} = k]$ . Eine fehlende Markierung symbolisieren wir durch einen Unterstrich ( $\_$ ).

**Mealy-Automat.** Der Übergang von  $z_1$  nach  $z_2$  als Reaktion auf die Eingabe von  $a$  mit gleichzeitiger Ausgabe von  $b$  wird wie folgt notiert:

$$(z_2, b) \in \delta(z_1, a) \Leftrightarrow (A, B, G) = (a, \_, b).$$

**Kellerautomat.** Als (nicht explizit erscheinende) Hilfsdatenstruktur benutzen wir einen Kellerspeicher mit unendlicher Kapazität, der mit Zeichen aus einem Alphabet  $T$  gefüllt werden kann, wobei ein bestimmtes Zeichen aus  $T$  als Kellerbegrenzer dienen soll. Auf diesem Kellerspeicher sollen die Funktionen push() und pop() in der üblichen Weise (siehe etwa Duden Informatik (1993)) operieren. Der Übergang von  $z_1$  nach  $z_2$  bei Eingabe von  $a$  und oberstem Kellersymbol  $k$  mit gleichzeitigem Ablegen der Zeichenfolge  $w$  auf dem Keller sieht dann so aus:

$$(z_2, w) \in \delta(z_1, a, k) \Leftrightarrow (A, B, G) = (a, \text{pop}() = k, \text{push}(w)).$$

**Turingmaschine.** Als Hilfsdatenstruktur verwenden wir hier ein unendlich langes Band mit linear angeordneten Zeichen aus einem Alphabet  $T$ , wobei eines dieser Zeichen als Leerzeichen verwendet wird. Die von der Eingabe von  $a$  ausgelöste Transition mit Schreiben (write) von  $b$  und anschließender Bewegung (move) des Schreib-Lesekopfes in Richtung  $c \in \{\leftarrow, \downarrow, \rightarrow\}$  beschreiben wir folgendermaßen:

$$(z_2, b, c) \in \delta(z_1, a) \Leftrightarrow (A, B, G) = (a, \_, \text{write}(b); \text{move}(c)).$$

Für  $G$  haben wir dabei eine *Aktionssequenz* eingesetzt, deren Einzelaktionen jeweils durch einen Strichpunkt getrennt werden.

### 5.2.3 Ein Getränkeautomat als Anschauungsobjekt

Nun sind wir in der Lage, einfache reale Automaten mit Hilfe von Zustandsübergangsdiagrammen zu beschreiben. Der Getränkeautomat in der Aula der Schule könnte dafür beispielsweise als Anschauungsobjekt herhalten. Wir stellen uns dazu Fragen wie z.B.:

- Wie verläuft die Kommunikation zwischen Menschen und dieser Maschine ?
- Welche Dienste stellt uns die Maschine zur Verfügung ?
- Wie unterscheidet die Maschine zwischen den Stationen eines Bedienungsablaufs?
- Wie verläuft ein Nutzungsfall des Automaten?

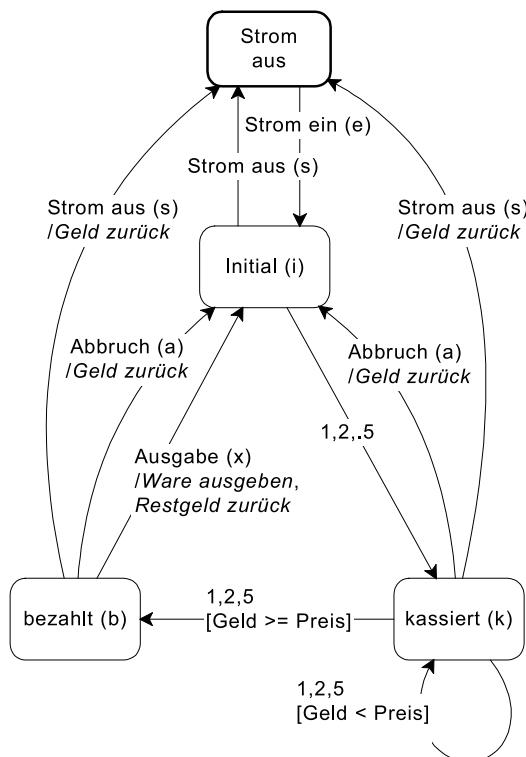
Nun probieren wir den Automaten systematisch aus und versuchen,

- die Reaktionen auf alle möglichen Folgen von Eingaben zu erfassen,
- alle möglichen Abläufe der Bedienungsprozesse festzuhalten und
- alle Randbedingungen und Zusicherungen aufzuspüren.

Die Ergebnisse dieser Untersuchungen halten wir in schriftlicher Form fest, wobei wir uns gleich einige Vereinfachungen leisten:

- Der Automat kann mit einem Netzschalter ein- und ausgeschaltet werden.
- Er akzeptiert Geldeinwurf, bis der festeingestellte Preis (z.B. 11 €) erreicht oder überschritten ist.
- Dann gibt er auf Befehl die Ware und das Wechselgeld aus.
- Zusätzlich kann jeder Vorgang mit einem Abbruchknopf vorzeitig beendet werden. In diesem Fall wird die gesamte Einnahme wieder ausgegeben.

Anschließend erstellen wir auf der Grundlage dieser Vereinbarungen ein Zustands-Übergangsdiagramm, wie es in Abb. 5.3 gezeigt ist.



**Abb. 5.3.** Unser Getränkeautomat (mit den in Abschnitt 5.3.6 verwendeten Abkürzungen)

Wir nehmen uns nun vor, diesen Automaten mit Hilfe des Rechners zu simulieren. Dazu müssen wir uns allerdings vorher einige Grundkenntnisse über Programmierung aneignen.

## 5.3 Simulation von Automaten

Das Ziel dieses Abschnittes ist ein lauffähiges Programm zur Simulation unseres Automaten. Dabei arbeiten wir auf vier verschiedenen Abstraktionsebenen: Automaten-, Algorithmus-, Programm-, und Prozessebene. Die jeweilige Abstraktionsebene muss den Schülern stets klar vor Augen geführt werden. Ebenso sollen die Schüler erkennen, dass ein bestimmter Automat, ebenso wie ein Algorithmus oder ein Programm, in Abhängigkeit von verschiedenen Eingaben oft zahlreiche mögliche Abläufe (Aktionsfolgen) aufweist.

Obwohl mittlerweile stark aus der Mode gekommen, verwenden wir als Programmiersprache hier *Pascal*, da diese Sprache im Schulbereich sehr gut verfügbar und unter den Lehrkräften vermutlich immer noch am bekanntesten ist. Wir wollen damit auch eine Alternative zu den (früher) üblichen „Pascal-Kursen“ vorführen, indem wir das Thema „Programmierung“ mit derselben Sprache völlig anders angehen. Ein dritter Vorteil ist die Erleichterung der Migration zu *Modula* oder *Oberon*, die beide für den Schulbereich sehr geeignet scheinen.

### 5.3.1 Algorithmen und Programme

Neben den Zustands-Übergangsdiagrammen stellen *Algorithmen* eine alternative Methode zur Beschreibung zeitlicher Abläufe dar. Ihre Grundelemente haben wir bereits früher (siehe Abschnitt 1.5 und 2.5) kennen gelernt. Hier zur Erinnerung nochmals eine Definition des Begriffs (nach Broy (1997)):

Ein *Algorithmus* ist ein Verfahren mit einer *präzisen* (d.h. in einer genau festgelegten Sprache abgefassten) *endlichen* Beschreibung unter Verwendung *effektiver* (d.h. tatsächlich ausführbarer) Verarbeitungsschritte.

Als *Sprache* kann dabei ein (präzisierter) Ausschnitt einer natürlichen Sprache ebenso wie eine (synthetische) formale Sprache verwendet werden. Unter die zweite Kategorie fallen *Programmiersprachen* wie z.B. Pascal, C, C++, Java. Ein *Programm* ist nichts anderes als die Formulierung eines Algorithmus in einer solchen Programmiersprache. Die einzelnen Verarbeitungsschritte werden (in imperativen Programmiersprachen) in Form von *Anweisungen* oder *Befehlen* an den Rechner formuliert. Ein Beispiel für einen in einer natürlichen Sprache formulierten Algorithmus:

*Falls* Nagel und Hammer vorhanden:

    Stecke den Nagel mit der Hand senkrecht zur Oberfläche ins Holz

    Nimm den Hammer in die rechte Hand

*Wiederhole* bis der Kopf des Nagels das Holz berührt

        Schlage mit dem Hammer auf den Nagelkopf

*Falls* der Nagel seitlich ausweicht

            Biege ihn wieder gerade

Hier erkennt man bereits, dass die Forderung nach Präzision nicht leicht zu erfüllen ist. Auch die Frage, wann Ausführungsschritte elementar sind, ist nicht immer

eindeutig zu beantworten. Dagegen ist leicht zu erkennen, dass Algorithmen einige wenige grundlegende Strukturelemente aufweisen:

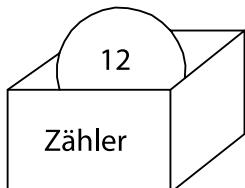
- *Sequenzen* (Folgen von Verarbeitungsschritten),
- *Alternativen* (Wenn, dann, Falls ...),
- *Wiederholungen* (Wiederhole bis ...).

Man nennt diese Strukturelemente auch *algorithmische Kontrollstrukturen*, da sie den Ablauf des Handlungsgeschehens kontrollieren.

### 5.3.2 Zustände und Variable

Für die Simulation der Zustände unserer Automaten verwenden wir *Variable*. Diese betrachten wir als mit einem Namen versehene Container, die jeweils einen Wert aus einer bestimmten *Wertemenge* (*Sorte* oder *Typ*) aufnehmen können. Dabei müssen wir auf Abgrenzung gegenüber dem mathematischen Variablenkonzept achten.

Ein Beispiel dazu: Die Variable mit dem Namen *Zähler* soll als Container für die Sorte der natürlichen Zahlen dienen. Sie könnte dann als Wert etwa die Zahl 12 enthalten. Der Name Variable (im Gegensatz zu *Konstanten*) kommt daher, dass dieser Wert *geändert* werden kann. Weitere Beispiele zeigt Tabelle 5.1.

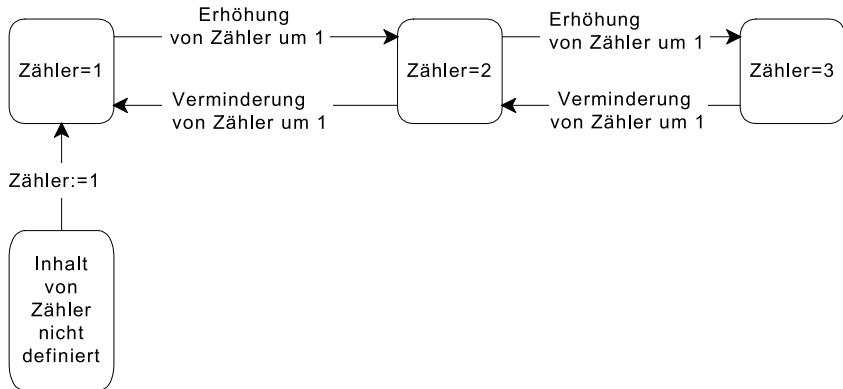


**Abb. 5.4.** Eine Variable aus der Sicht der Schüler

**Tabelle 5.1.** Beispiele für Variable und Inhalte

Name	Wertebereich	Werte z.B.
Buchtitel	Text (String)	„Fräulein Smillas Gespür für Schnee“, „Faust“
Geburtsdatum	Datum	24.3.1966, 18.12. 1783
Abteilung	Zeichen (Character)	,2‘, ,h‘, ,X‘, ,+‘

Der Zusammenhang zwischen Variablen und Zustands-Übergangsdiagrammen wird hergestellt, indem man jedem Zustand, wie in Abb. 5.5 gezeigt, einen bestimmten Wert einer Variablen (oder eine Kombination von Werten mehrerer Variablen) zuordnet.



**Abb. 5.5.** Variablen und Zustände

Übergänge werden dann durch Veränderung des Wertes der jeweiligen Variable ausgelöst: Der Variablen *Zähler* wird zum Beispiel der (veränderte) Wert 1 zugewiesen. Eine solche Aktion heißt deshalb *Zuweisung*. Wir schreiben dafür

*Zähler := 1.*

Zuweisungen sind charakteristisch für *imperative Programmiersprachen*.

Die *Aktion* *Zähler := 1* muss dabei streng von der *Aussage* *Zähler = 1* unterschieden werden. Letztere behauptet, dass die Variable *Zähler* gegenwärtig den Wert 1 hat und kann die Werte „wahr“ oder „falsch“ annehmen. Diese Unterscheidung wird den Schülern mit Hilfe der Zustands-Übergangsdiagramme schnell klar.

### 5.3.3 Imperative Programmierung

Ein imperatives Programm besteht aus einer *Kombination von Befehlen oder Anweisungen*, von denen jeder eine *spezielle Aktion* auslöst. Wir kennen bisher nur den Zuweisungsbefehl (s. Tabelle 5.2). Man kann den Ablauf imperativer Programme grundsätzlich als eine Folge von Zustandskombinationen der betroffenen Variablen ansehen.

**Sequenzen von Befehlen.** Um eine Folge von Anweisungen hintereinander ausführen zu können, gibt es die Möglichkeit, diese in Form einer *Sequenz* anzurufen. In vielen Programmiersprachen werden syntaktisch an bestimmten Stellen, etwa innerhalb einer Auswahlanweisung (s. unten) nur einzelne Befehle zugelassen. In einem solchen Fall gibt es meist die Möglichkeit, eine Sequenz von Befehlen mit Hilfe einer *Sequenzklammer* zu einem Befehl zusammenzufassen. In Pascal werden dazu die Wörter *begin* und *end* verwendet. In C, C++ und Java verwendet man stattdessen die geschweiften Klammern { und }. Die einzelnen Befehle innerhalb der Sequenz werden meist (wie in Pascal) durch Strichpunkte voneinander getrennt.

**Tabelle 5.2.** Der Zuweisungsbefehl in Pascal

Aktionstyp	Zuweisung
Syntax	<code>&lt;Variable&gt; := &lt;Term&gt;</code>
Beispiel	<code>Zähler := Zähler/3</code>
Wirkung	Der Wert des Terms auf der rechten Seite von <code>:=</code> wird berechnet. Anschließend wird das Ergebnis der auf der linken Seite bezeichneten Variablen als Wert zugewiesen.

Ein möglicher Ablauf unseres Automaten aus Abb. 5.5 wäre dann etwa durch die folgende Pascal-Befehlsfolge festgelegt:

```
zähler := 1;
zähler := zähler+1;
zähler := zähler+1;
zähler := zähler-1;
zähler := zähler-1;
```

Hierbei ist allerdings Vorsicht geboten: Diese Befehlsfolge repräsentiert *nicht* den obigen Automaten, sondern nur *einen* seiner möglichen Abläufe!

**Tabelle 5.3.** Das erste vollständige Programm

Befehl	Wirkung
program	Programmdeklaration: Das Programm erhält einen Namen
var Zähler: Integer;	Variablen-deklaration: Die Container werden eingerichtet
begin zähler := 1; zähler := zähler+1; zähler := zähler+1; zähler := zähler-1; zähler := zähler-1; end.	Beginn der Befehlsfolge Zuweisungsbefehl B1 Zuweisungsbefehl B2 Zuweisungsbefehl B3 Zuweisungsbefehl B4 Zuweisungsbefehl B5 Ende des Programms

**Programme und Prozesse.** Nun ergänzen wir unsere Befehlssequenz zu einem ersten vollständigen Pascal-Programm Nummer1 (siehe Tabelle 5.3). Dieser *Programmtext* ist zunächst für den Rechner nur ein Text wie jeder andere, z.B. wie Weihnachtsgrüße an Tante Emma. Um das Programm ausführen zu können, muss dieser Text erst durch ein speziell für diesen Zweck konstruiertes Programm (*Compiler* oder *Interpreter*) in eine Folge von direkt ausführbaren Maschinenbefehlen (*Maschinencode*) übersetzt werden. Diese Befehle können hier als Kombinationen aus Nullen oder Einsen betrachtet werden, z.B. 1001 1111 0101 1100. Später werden wir mehr darüber erfahren.

Dieser Maschinencode wird dann im Hauptspeicher des Rechners oder als ausführbare Datei auf der Festplatte aufbewahrt, bis seine tatsächliche Abarbeitung gestartet wird. Dann entsteht daraus ein (aktiver) *Prozess*. Wir fassen die verwendeten Begriffe in Tabelle 5.4 nochmals zusammen.

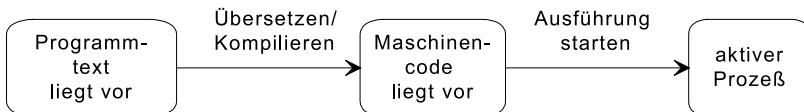
**Tabelle 5.4.** Vom Programm zum Prozess

---

Programm	Formulierung eines Algorithmus in einer bestimmten Programmiersprache, z.B. Pascal, Basic, C++, Java
Maschinencode	Folge von Maschinenbefehlen, welche vom Rechner unmittelbar ausgeführt werden können
Prozess	Folge von Aktionen des Rechners
Compiler	Übersetzungsprogramm, das vor dem Start des Prozesses den gesamten Programmtext in Maschinencode übersetzt
Interpreter	Übersetzungsprogramm, das die Befehle des Programmtexes einzeln übersetzt und sofort deren Ausführung startet

---

Der Übergang vom Programmtext zum Prozess kann (für den Fall der Verwendung eines Compilers) wiederum mit einem Zustands-Übergangsdiagramm beschrieben werden (siehe Abb. 5.6).



**Abb. 5.6.** Vom Programmtext zum Prozess

### 5.3.4 Variablen- und Modellzustände

Bei Verwendung mehrerer Variablen wird ein Ablauf eines Programms durch eine Spur im Zustandsraum beschrieben. Dieser Raum wird durch die Variablen aufgespannt, wobei jede Variable eine Dimension beiträgt. Ein Punkt in diesem Raum entspricht dabei der Kombination je eines Zustandes aller *Variablen*. Wir betrachten ein kleines Programm Nr.2 mit zwei Variablen:

```

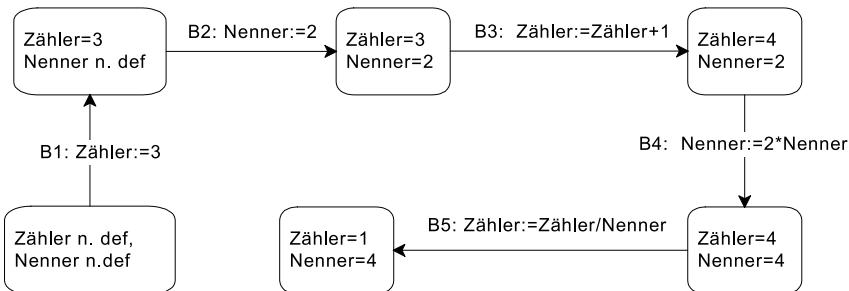
begin
Zähler :=3;           Zuweisungsbefehl B1
Nenner :=2;           Zuweisungsbefehl B2
Zähler := Zähler +1;  Zuweisungsbefehl B3
  
```

```

Nenner := 2*Nenner;      Zuweisungsbefehl B4
Zähler := Zähler/Nenner; Zuweisungsbefehl B5
end.

```

Abbildung 5.7 zeigt das Zustands-Übergangsdiagramm dieses Programms.



**Abb. 5.7.** Programm Nr.2 im Zustands-Übergangsdiagramm

	Zähler n. def	Zähler = 1	Zähler = 2	Zähler = 3	Zähler = 4
Nenner n. def	$z_0$			$z_1$	
Nenner = 1					
Nenner = 2					$z_2$
Nenner = 3					
Nenner = 4			$z_5$		$z_4$
Nenner = 5					

**Abb. 5.8.** Der Ablauf von Programm Nr. 2 als Spur im Zustandsraum

Die Spur des (einzig möglichen) Ablaufs von Programm Nr. 2 in seinem zweidimensionalen Zustandsraum ist in Abb. 5.8 dargestellt. Zustandsräume von Programmen mit mehr als drei Variablen sind offensichtlich nicht mehr direkt visualisierbar.

**Modellzustände.** Letztlich wollen wir mit Hilfe von Zustands-Übergangsdiagrammen das zeitliche Verhalten von Automaten modellieren, um dann ihren Ablauf mit Hilfe einer Programmiersprache simulieren zu können. Hierbei kommt aus der Sicht des Modells nicht allen Punkten des Variablenzustandsraumes eine sinnvolle Bedeutung zu. Die dafür relevanten Zustände des Modells entsprechen meist Unterräumen des Variablenzustandsraums. In diesen Fällen fassen wir deshalb eine zusammenhängende Menge von Punkten (einen Unterraum) des Variablenzustandsraums zu einem *Modellzustand* zusammen. Im Folgenden müssen wir also zwischen *Variablen-* und *Modellzuständen* unterscheiden. Die Variablen-

zustände entstehen erst durch die Implementierung in einer Programmiersprache, die Modellzustände entsprechen dagegen den relevanten Zuständen des zu simulierenden Systems. Bei unserem Getränkeautomaten aus Abb. 5.3 werden beispielsweise viele Variablenzustände (nämlich alle mit  $Geld \geq 0,5 \text{ €}$ ) der Variablen *Geld* zum Modellzustand *bezahlt* zusammengefasst. Zur Implementierung von *Modellzuständen* führt man praktischerweise oft eine eigene Variable für den Namen des aktuell vorliegenden Zustandes ein, z.B.:

$$\textit{aktueller\_Zustand} := \text{„bezahlt“}.$$

Eine Transition kann also durch verschiedene Aktionstypen ausgelöst werden, darunter:

1. durch *jede* Änderung des Wertes *irgendeiner* Variablen (wie in Abb. 5.7),
2. durch *jede* Änderung des Wertes einer „signifikanten“ Variablen (z.B. der Variablen *aktueller\_zustand*),
3. durch das Über- bzw. Unterschreiten eines gewissen Schwellenwertes durch den Wert einer Variablen (wie z.B. Geldeingabe  $\geq$  Preis),
4. durch eine (oder mehrere) bestimmte Benutzereingabe(n) (z.B. „Ausschalten“).

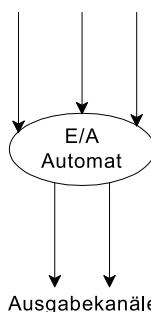
Für die Simulation des letztgenannten Typs fehlt uns noch die Möglichkeit, mit unserem Automaten zu kommunizieren, was uns zur Betrachtung von Ein- und Ausgabekonzepten führt.

### 5.3.5 Automaten mit Ein- und Ausgabe

In der Regel werden Automaten durch Eingaben, d.h. durch Signale aus ihrer Umgebung auf bestimmten Kanälen, gesteuert. Aufgrund der Regeln des Zustands-Übergangsdiagramms produzieren sie in Reaktion auf die Eingaben ganz bestimmte Signale auf speziellen Ausgabekanälen. Diese Signale können wiederum (als deren Eingaben) Aktionen anderer Automaten auslösen. Das Datenflussdiagramm des Automaten in Abb. 5.9 zeigt z.B. drei Eingabe- und zwei Ausgabekanäle. In unserer Umgebung finden sich unzählige Maschinen, die auf diese Weise arbeiten, wie etwa Kaffee-, oder Kaugummiautomaten, Fernsprechleinrichtungen oder Computer. Unser Getränkeautomat aus Abb. 5.3 weist die folgenden Kanäle auf:

- Eingabekanäle: Netzschalter, Geldeinwurf, Getränkewahltafel, Rückgabeknopf,
- Ausgabekanäle: Netzsignallämpchen, Geldausgabe, Getränkeausgabe.

Eingabekanäle

**Abb. 5.9.** Datenflussdiagramm eines E/A-Automaten

Entsprechend den Kanälen eines Automaten muss auch eine Programmiersprache Möglichkeiten für die Kommunikation zwischen Benutzern und Prozessen bereitstellen. Tabelle 5.5 zeigt einige Ein- bzw. Ausgabeanweisungen unserer Programmiersprache Pascal.

**Tabelle 5.5.** Ein- und Ausgabe in Pascal

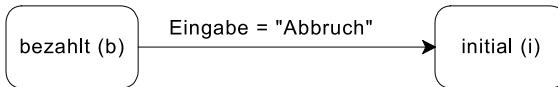
Aktionstyp	Eingabe	Ausgabe
Syntax	<code>readln(&lt;Variablenname&gt;)</code>	<code>writeln(&lt;Term&gt;)</code>
Beispiel	<code>readln(zähler)</code>	<code>writeln(zähler+1)</code>
Wirkung	Die Eingabe von der Tastatur bis zum Betätigen der return-Taste wird der genannten Variablen als Wert zugewiesen.	Der Wert des Terms wird am Bildschirm ausgegeben.

Im Zustands-Übergangsdiagramm von E/A-Automaten werden Übergänge in der Regel durch Eingaben ausgelöst (siehe Abschnitt 5.4.2). Zur Vereinfachung notieren wir dann an den Übergangspfeilen meist nur die *Eingabewerte*, nicht mehr die gesamte Aktion. Wir schreiben also nur noch „Cola“ anstatt „Auswahlknopf für Cola betätigt“.

### 5.3.6 Bedingte Übergänge

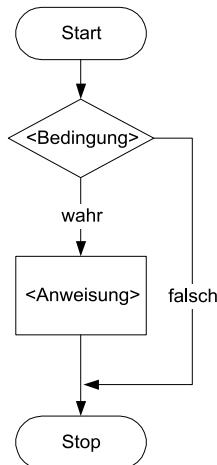
Bisher haben unsere Programme nur den Ablauf von Automaten simuliert, deren Zustände jeweils einem Punkt im *Variablenzustandsraum* entsprachen. In diesem Fall führt jede Zuweisung eines veränderten Wertes an irgendeine Variable automatisch zur Änderung des Zustandes. Bei *Modellzuständen* gilt das unter Umständen nicht mehr, nämlich dann, wenn der Wert einer für den Zustand irrelevanten Variablen verändert wird, etwa der Wert von *Geld* im Zustand *kassiert* bei unse-

rem Getränkeautomaten aus Abb. 5.3. Vor der Auslösung solcher Übergänge muss also erst explizit nachgeprüft werden, ob eine bestimmte Bedingung erfüllt ist: „Wenn eine bestimmte Aussage wahr ist, dann gehe zum Zustand  $Z_x$  über“. Als Aussage könnten dabei z.B. auftreten: „Der Wert der Variablen  $V$  ist größer als 10“ oder „die Eingabe lautet ‚Abbruch‘“ (siehe Abb. 5.10).



**Abb. 5.10.** Steuerung eines Übergangs durch eine bedingte Anweisung ohne Alternative

**Bedingte Anweisung.** Für die Implementierung solcher bedingter Übergänge benötigen wir ein geeignetes Sprachmittel, das imperative Programmiersprachen in Form der *bedingten Anweisung* (ohne Alternative) zur Verfügung stellen. Tabelle 5.6 zeigt diese Anweisung in Pascal. Zur Veranschaulichung der Wirkungsweise kann man zusätzlich Ablaufpläne (Kontrollflussdiagramme) einsetzen, wie in Abb. 5.11 gezeigt.



**Abb. 5.11.** Ablaufplan für eine bedingte Anweisung

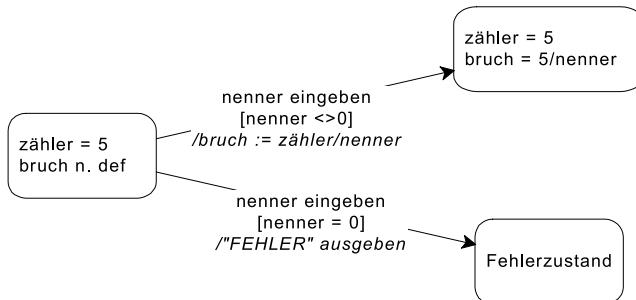
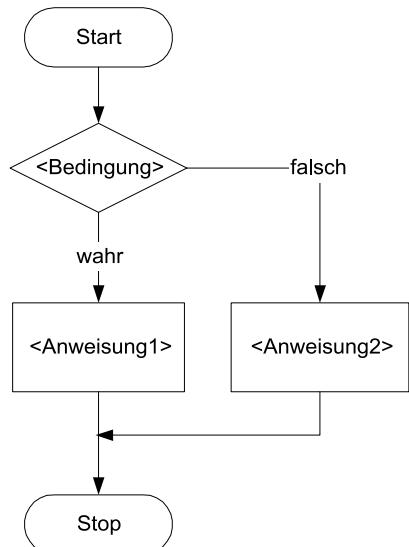
**Alternative.** Leider reichen die Möglichkeiten unserer bedingten Anweisung noch nicht aus, um Automaten befriedigend simulieren zu können. Oft wäre es günstig, nicht nur zwischen der Ausführung einer Anweisung und keiner Aktion, sondern zwischen zwei Anweisungen auswählen zu können, wie zum Beispiel bei der Reaktion auf fehlerhafte Eingaben in Abb. 5.12. Diese Möglichkeit bietet die *Alternative*. Eine typische Anwendung solcher Alternativen sind Reaktionen auf fehlerhafte Eingaben. In Tabelle 5.7 findet sich neben der Pascal-Syntax der Anweisung auch das Beispiel aus Abb. 5.12.

**Tabelle 5.6.** Die bedingte Anweisung in Pascal

---

Aktionstyp	bedingte Anweisung
Syntax	If <Bedingung> then <Anweisung>
Beispiel	if Eingabe = „Abbruch“ then Zustand := „Initial“
Wirkung	Falls die <Bedingung> wahr ist, wird die <Anweisung> ausgeführt. Ansonsten wird zum nächsten Befehl nach der Auswahlanweisung übergegangen.

---

**Abb. 5.12.** Reaktion auf Eingabefehler**Abb. 5.13.** Ablaufplan für eine bedingte Anweisung mit Alternative

**Tabelle 5.7.** Die bedingte Anweisung mit Alternative

Aktionstyp	bedingte Anweisung mit Alternative
Syntax	If <Bedingung> then <Anweisung1> else <Anweisung2>
Beispiel	if nenner <> 0 then bruch:=zähler/nenner else writeln(,FEHLER!')
Wirkung	Falls <Bedingung> wahr ist, wird die <Anweisung1> ausgeführt, ansonsten <Anweisung2>.

**Fallunterscheidung.** Natürlich finden sich auch Beispiele, in denen es nicht nur eine Alternative, sondern gleich mehrere Alternativen zu einer Aktion gibt. Tabelle 5.8 listet beispielsweise die Alternativen für Übergänge aus dem Zustand „kas-siert“ unseres Getränkeautomaten aus Abb. 5.3 auf.

**Tabelle 5.8.** Mehrfache Alternativen für Übergänge

Eingabe	Bedingung	Endzustand	Ausgabe
Abbruch	keine	initial	Geld zurück
1, 2, 5	Geld >= Preis	bezahlt	keine
1, 2, 5	Geld < Preis	kassiert	keine
Stop	keine	Strom aus	Geld zurück

Um in diesen Fällen komplizierte Schachtelungen von bedingten Anweisungen (z.B. if ... then if ... then ... else ... else ) zu vermeiden, bieten Programmiersprachen meist ein Sprachkonstrukt für multiple Fallunterscheidungen an. Leider kann in Pascal die Anweisung für die mehrfache Auswahl nur durch Variablen gesteuert werden, deren Wertemenge linear geordnet ist. Dazu gehören die Typen Zeichen (char), ganze Zahlen (integer), aber nicht Gleitkommazahlen (real) oder Texte (string). Diese Einschränkung zwingt uns, die Eingaben durch einzelne Zeichen zu beschreiben:

‘1’, ‘5’ für Münzen im Wert von 1 bzw. 5 € (2 € und 50 Cent lassen wir vorerst weg), ‘a’ für „Abbruch“, ‘s’ für „Stop“.

Damit können wir die Übergänge aus Tabelle 5.8 wie folgt programmieren:

```
program getraenke-teil;
var zustand: String; eingabe: Char; geld: Integer;
const preis = 3;
begin
  ...
  zustand := 'kassiert';
  case eingabe of
    '1', '5': begin
      geld:=geld+eingabe;
      if geld >= preis then zustand:='bezahlt';
    end;
    'a': begin
      zustand := 'initial';
    end;
    's': begin
      zustand := 'initial';
    end;
  end;
end.
```

```

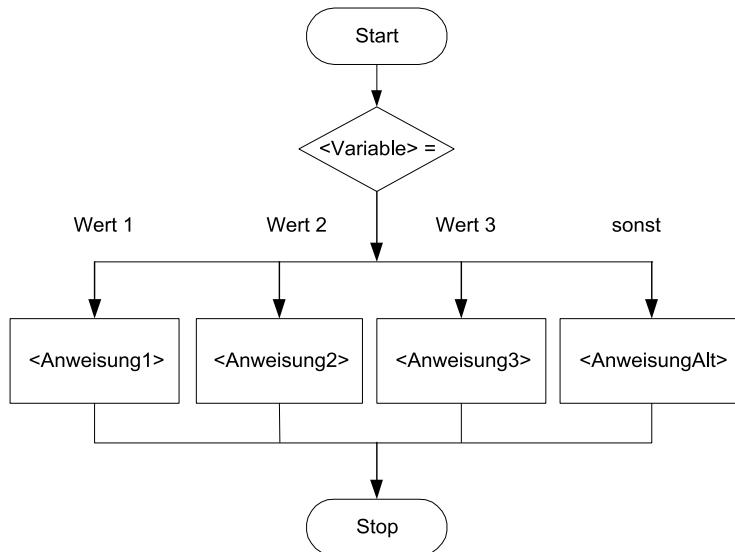
end;
'a':
begin
zustand:='initial'; ausgabe:=geld;
end;
`s':
begin
zustand:='Strom aus'; ausgabe:=geld;
end;
else
writeln /,,FEHLER!“)
end; ... end.

```

Tabelle 5.9 beschreibt die Pascal-Anweisung zur mehrfachen Fallunterscheidung, die wir in unserem Programmfragment verwendet haben. Die Standardalternative (*else*-Zweig) kann weggelassen werden. Anstatt der Angabe von Werten können auch Listen von Werten ('5', '1', '0') oder Bereiche (2 ... 8) angegeben werden.

**Tabelle 5.9.** Fallunterscheidung in Pascal (siehe auch Abb. 5.14)

Aktionstyp	Fallunterscheidung
Syntax	case <Ordinalvariable> of <Wert1>: <Anweisung1>; <Wert2>: <Anweisung2>; ... else <AnweisungAlt> end;
Wirkung	Falls <Ordinalvariable> einen der aufgelisteten Werte <Wertn> annimmt, wird die dahinter stehende <Anweisung> ausgeführt, ansonsten kommt <AnweisungAlt> zur Ausführung.



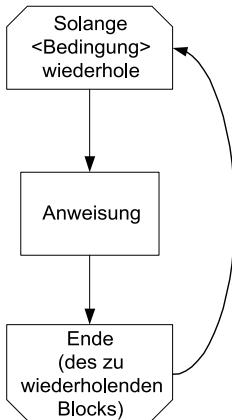
**Abb. 5.14.** Kontrollfluss der Fallunterscheidung

### 5.3.7 Wiederholungen

Zur Simulation unseres Automaten fehlt uns nun nur noch die Fähigkeit, wiederholt auf Eingaben reagieren zu können. Bisher können wir nur auf eine einzige Eingabe reagieren. Wir benötigen also zusätzlich noch eine Möglichkeit, Anweisungen zu wiederholen. Natürlich muss diese Wiederholung abgebrochen werden können, um Endlosschleifen zu vermeiden. Wir wollen deshalb nur solange wiederholen, als eine bestimmte Bedingung wahr ist.

**Tabelle 5.10.** Die Wiederholung in Pascal

Aktionstyp	Wiederholung
Syntax	while <Bedingung> do <Anweisung>
Beispiel	while zahl < 10 do begin writeln(zahl*zahl); zahl:=zahl+1; end;
Wirkung	Solange die Bedingung wahr ist, wird die Anweisung ausgeführt.



**Abb. 5.15.** Kontrollfluss bei der Wiederholung

Jetzt verfügen wir endlich über alle Hilfsmittel, um unseren Automaten aus Abb. 5.3 vollständig mit Hilfe eines Pascal-Programms simulieren zu können, wobei wir den Einschaltvorgang des Automaten durch den *Start* des Programmablaufs modellieren:

```
program Automat;
uses WinCrt;
const
  preis = 11;
  var
    zustand: Char;
    eingabe: Char;
    geld: Integer;
    restgeld: Integer;
    power: String;

begin
  zustand := 'i'; power := 'on';
  while power = 'on' do
    begin
      gotoXY(10,1); write('Eingabe:'); eingabe:=readkey;
      Case zustand of
        'i': begin
          geld := 0;
          Case eingabe of
            '1','2','5': begin
              geld:= Ord(eingabe)-48;zustand:='k';
              end;
            's': power := 'off';
            end;
          end;
        'k': Case eingabe of
          '1','2','5': begin
            geld:= geld+Ord(eingabe)-48;
            if geld >= preis then zustand:='b';
            end;
          'a': begin
            restgeld := geld;
            zustand := 'i';
            gotoxy(20,10);writeln('Restgeld:',restgeld);
            end;
          's': begin
            restgeld := geld;
            power := 'off';
            gotoxy(20,10);writeln('Restgeld:',restgeld);
            end;
          end;
        'b': Case eingabe of
          'x': begin
            restgeld := geld-preis;
            zustand := 'i';
            gotoxy(20,10);writeln('Restgeld:',restgeld);
            gotoXY(20,11);writeln('Warenausgabe:','KAUGUMMI!');
            end;
```

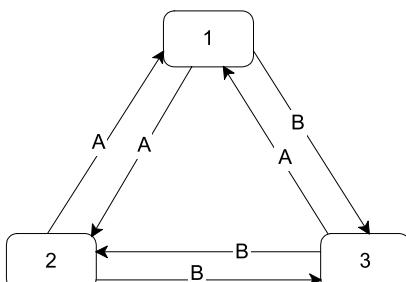
```

'a': begin
    restgeld := geld;zustand := 'i';
    gotoxy(20,10);writeln('Restgeld:',restgeld);
    end;
's': begin
    restgeld := geld;power := 'off';
    gotoxy(20,10);writeln('Restgeld:',restgeld);
    end;
end;
gotoxy(30,1);write('Geld:',geld);
gotoxy(30,2);write('Zustand:',zustand);
end;
end.

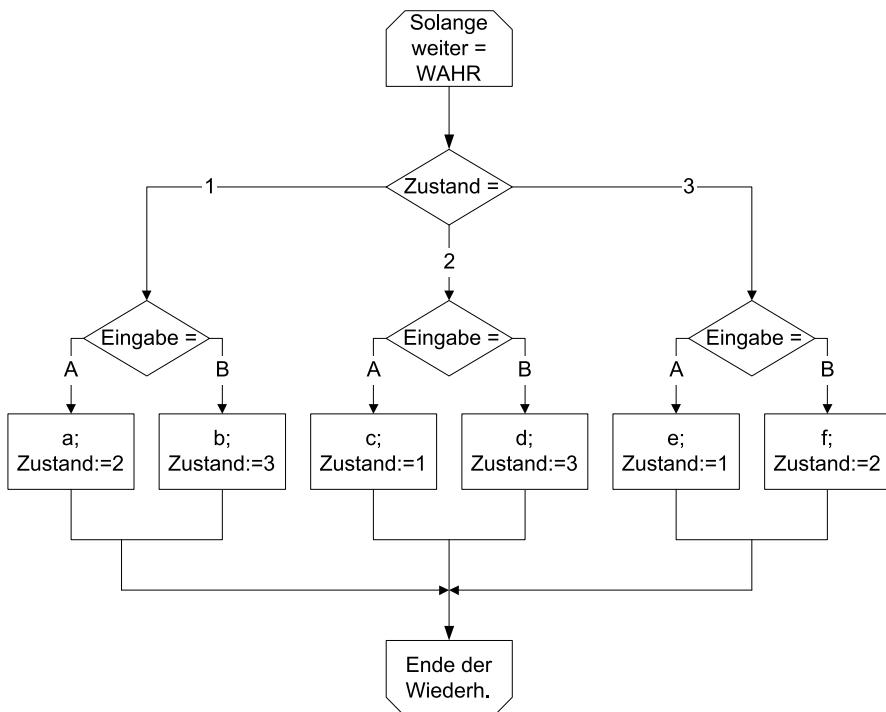
```

Wir erkennen in diesem Programmtext sehr leicht eine Struktur, die für die Simulation von endlichen Automaten typisch ist: eine Wiederholung, innerhalb derer eine Fallunterscheidung für die Zustände geschachtelt ist, die wiederum in jeder Alternative jeweils eine Fallunterscheidung für die Eingaben enthält. Am Beispiel des schematischen Automaten aus Abb. 5.16 verdeutlichen wir diese strukturellen Merkmale nochmals, indem wir ein Kontrollflussdiagramm (siehe Abb. 5.17) und einen Algorithmus dazu entwerfen:

Wiederhole solange weiter = WAHR  
Falls Zustand =  
1:      Falls Eingabe =  
          A: Aktion a; Zustand = 2  
          B: Aktion b; Zustand = 3  
2:      Falls Eingabe =  
          A: Aktion c; Zustand = 1  
          B: Aktion d; Zustand = 3  
3:      Falls Eingabe =  
          A: Aktion e; Zustand = 1  
          B: Aktion f; Zustand = 2



**Abb. 5.16.** Schematischer Automat



**Abb. 5.17.** Schema des Kontrollflusses bei der Simulation von Automaten

## 5.4 Ausbau und Wertung

Nachdem unsere Implementierung des ursprünglichen Getränkeautomaten ziemlich spartanisch ausgefallen ist, könnte man interessierte Schüler mit entsprechendem Vorwissen dazu anregen, eine etwas luxuriösere Version zu basteln. Als Plattform wäre z. B. *Java-Script* (in Verbindung mit *HTML*) oder *Visual Basic®* denkbar. Mögliche Ergebnisse zeigen die Abbildungen 5.18 und 5.19. Die erste dieser beiden Versionen hat dabei den Vorteil, dass die Simulation über das Intranet der Schule zugänglich ist und daher in darauffolgenden Jahren als Anschauungsmaterial, u.U. sogar als Grundlage für Hausaufgaben, zur Verfügung steht. Die *Visual-Basic*-Version weist dagegen den bereits in Teil B, Abschnitt 3.4.4 angesprochenen Nachteil einer gewissen Undurchschaubarkeit seiner Funktionsweise auf.

Natürlich endet auch diese Unterrichtssequenz mit einer ausführlichen Reflexionsphase:

- Wo haben wir vereinfacht?
- Was passiert bei Fehlbedienungen?

- Was macht der Automat, wenn der Geldspeicher voll ist?
- Sind alternative Konzeptionen denkbar?

Als Vertiefung bietet sich schließlich die Berechnung der Stückelung des Wechselgeldes an.

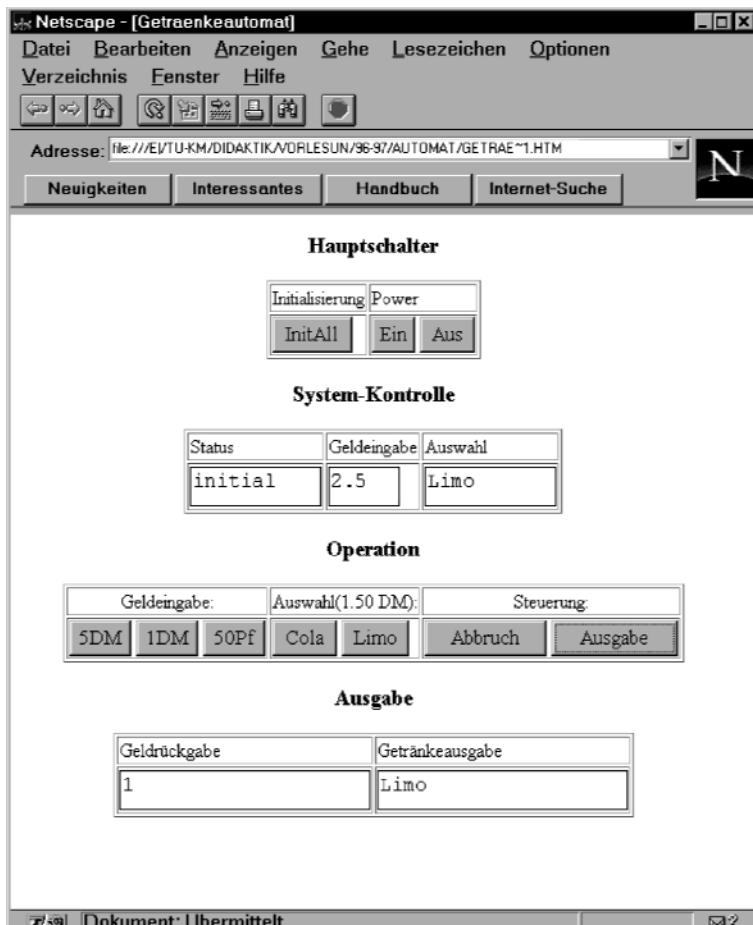


Abb. 5.18. Ein Automat in HTML/Java-Script

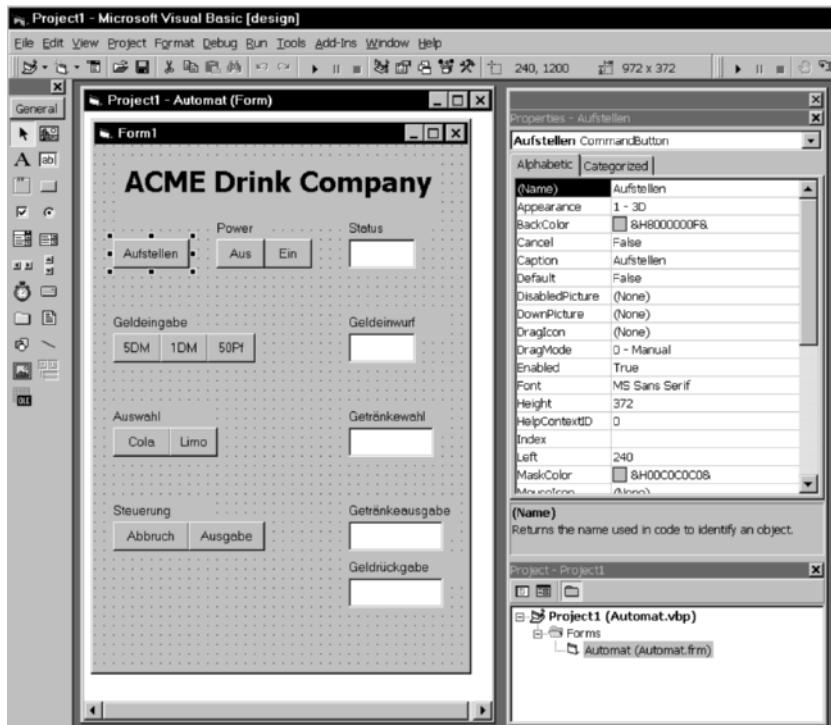


Abb. 5.19. Eine Implementierung in *Visual Basic*®

# 6 Funktionale Modellierung, Teil 2

Die bereits in Kapitel 3 ausführlich behandelten Datenflussdiagramme greifen wir nun anhand eines einfachen Beispiels zur Datenverschlüsselung wieder auf und überlegen uns, wie man Modelle dieser Art simulieren kann. Dies führt uns auf das Funktions- bzw. Prozedurkonzept von Programmiersprachen, wozu wir im realen Unterricht möglichst die bisher benutzte Programmiersprache weiter verwenden. Um dem Leser eine alternative Implementierungsplattform vorzustellen, wollen wir in diesem Buch dagegen eine makroprogrammierbare Tabellenkalkulation benutzen.

## 6.1 Problemstellung

Die primären Lernziele dieser Lektion liegen im Bereich der funktionalen Modellierung, betreffen also die Modularisierung komplexer Systeme und die Kommunikation zwischen den Modulen und weniger spezielle Verfahren zur Datenverschlüsselung oder deren mathematische Aspekte. Wir wählen daher ein relativ einfaches, symmetrisches Chiffrierverfahren.

Zur Motivierung demonstrieren wir den Schülern etwa, wie leicht man unter Einsatz des Administrator-Passwortes die elektronische Post von anderen Rechnerbenutzern lesen kann. Beim Transport sensibler Daten ist es daher dringend anzuraten, diese angemessen zu verschlüsseln und damit das Lesen für Unbefugte zumindest zu erschweren.

## 6.2 Problembeschreibung

Anfangs betrachten wir verschiedene Methoden zur Verschlüsselung von Texten. Beginnend mit der bereits in der Antike angewandten Verschiebungsmethode (um eine feste Schrittweite) entwickeln wir eine auf der polygraphischen Caesar-Addition (siehe Bauer (1997)) basierende Verschlüsselungsmethode. Das Verfahren und die verwendeten Fachbegriffe werden zunächst an einem Beispiel vorgestellt: Zwischen zwei Partnern soll eine Nachricht über die Zeit eines Treffens („heute Nachmittag“) so übermittelt werden, dass zufällige Mithörer ihren Sinn nicht verstehen können.

Vorab vereinbaren wir (auf einem sicheren Kanal) mit dem Partner ein Schlüsselwort, z.B. QUELLE. Anschließend identifizieren wir alle Buchstaben des Alphabets in aufsteigender Folge mit je einer Zahl zwischen 0 und 25 (wobei wir uns auf Großbuchstaben beschränken):

A	B	C	D	..	Z
0	1	2	3	..	25

Auf dieser Grundlage transformieren wir das *Schlüsselwort* und die zu verschlüsselnde Nachricht in je eine Folge von Zahlen (Leerzeichen lassen wir dabei einfach weg):

H	E	U	T	E	N	A	C	H	M	I	T	T	A	G
7	4	20	19	4	13	0	2	7	12	8	19	19	0	6
Q	U	E	L	L	E									
16	20	4	11	11	4									

Dann erfolgt die eigentliche Chiffrierung: Die Zahlenfolge der Nachricht (des *Klartextes*) wird in Abschnitte von der Länge des Schlüsselwortes unterteilt. Anschließend wird jeder dieser Abschnitte als Vektor betrachtet und dazu (komponentenweise) die ebenfalls als Vektor angesehene Zahlenfolge des Schlüssels addiert (unvollständige Vektoren werden mit beliebigen Zahlen, z.B. Nullen aufgefüllt):

$$\begin{array}{lcl} (7, 4, 20, 19, 4, 13) & + & (16, 20, 4, 11, 11, 4) \\ (0, 2, 7, 12, 8, 19) & + & (16, 20, 4, 11, 11, 4) \\ (19, 0, 6, \_, \_) & + & (16, 20, 4, \_, \_, \_) \end{array} = \begin{array}{l} (23, 24, 24, 30, 15, 17); \\ (16, 22, 11, 23, 19, 23); \\ (35, 20, 10, \_, \_, \_) \end{array}$$

Im nächsten Schritt werden die Ergebnisvektoren wieder zu einer Zahlenfolge der ursprünglichen Länge zusammengesetzt. „Überstehende“ Teile des letzten Vektors werden dabei ggf. ignoriert. Schließlich wird die so entstandene Zahlenfolge wieder in eine Zeichenkette umgewandelt, wobei von zu großen Zahlenwerten ( $> 25$ ) vorher 26 subtrahiert wird (wir rechnen also modulo 26). Wir erhalten damit die folgende verschlüsselte Nachricht (*Geheimtext*):

23	24	24	4	15	17	16	22	11	23	19	23	9	20	10
X	Y	Y	E	P	R	Q	W	L	X	T	X	J	U	K

Zur Entschlüsselung subtrahiert der Empfänger von den (analog zum Klartext gewonnenen) Zahlenvektoren des Geheimtextes jeweils den Vektor des Schlüssels. Nachdem ggf. zu negativen Zahlen 26 addiert wurde, kann man durch Zusammensetzen der Vektoren und Umrechnung der einzelnen Zahlen in Zeichen den Klartext rekonstruieren.

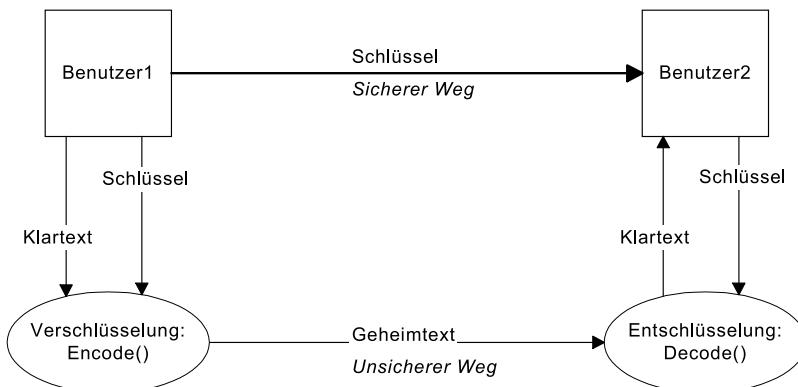
## 6.3 Modellierung

Wir verzichten (hier und im Unterricht) auf eine streng mathematische Formulierung des Verschlüsselungsverfahrens, die man ausführlich in Bauer (1997) nachlesen kann und beschränken uns auf die Entwicklung eines geeigneten Algorithmus. Zuvor klären wir aber mit Hilfe eines Datenflussdiagramms die Struktur des Ge-

samtsystems. Rechenfertigkeiten in Restklassen (modulo) wären allerdings sehr hilfreich. Außerdem beschränken wir uns hier auf die Entwicklung des Verschlüsselungsalgorithmus. Als Lernzielkontrolle könnte man die Schüler abschließend selbstständig einen Entschlüsselungsalgorithmus entwickeln lassen.

### 6.3.1 Datenflüsse und Prozesse

Zunächst überlegen wir uns den Datenfluss bei einem Ver- und Entschlüsselungsvorgang.



**Abb. 6.1.** Datenfluss bei verschlüsselter Nachrichtenübertragung

Die einzelnen Elemente des Datenflussdiagramms werden hier ebenfalls nochmals besprochen (siehe auch Abschnitt 2.8):

- Prozesse,
- Datenflüsse als Ein- und Ausgabekanäle von Prozessen,
- Datenquellen und -senken als Schnittstellen des Gesamtsystems.

### 6.3.2 Der Verschlüsselungsalgorithmus

Jetzt überlegen wir uns einen Algorithmus zur Verschlüsselung nach dem oben besprochenen Prinzip, den wir zunächst in Umgangssprache formulieren:

Eingabe von Klartext und Schlüsselwort,  
initialisiere den Geheimtext,  
wiederhole vom ersten bis letzten Buchstaben des Klartextes:  
bestimme die Kennzahl des Klartextbuchstaben,  
bestimme die Kennzahl des entsprechenden Schlüsselbuchstaben,  
addiere die beiden Zahlen zu einer neuen Kennzahl,  
bestimme den Buchstaben zu dieser neuen Kennzahl,  
hänge diesen Buchstaben an den Geheimtext an.

Eine gewisse Ungenauigkeit zeigt sich hier bei der Festlegung, welcher Schlüsselbuchstabe dem jeweiligen Klartextbuchstaben entspricht, da in der Regel der Klartext länger ist als der Schlüssel. Die Lösung erfolgt im nächsten Schritt bei der Formalisierung des Algorithmus mit Hilfe einer geeigneten Datenstruktur.

### 6.3.3 Die Datenstruktur der Zeichenketten

Es bietet sich an, (spätestens) in diesem Zusammenhang die Menge  $Z$  der Zeichenketten (*String*) als Sequenzen von Einzelzeichen aus einem bestimmten Alphabet  $A$  (mit max Zeichen) einzuführen:

$$Z = \{t \mid t = „z_0 z_1 z_2 \dots z_n“ \text{ mit } z_i \in A \text{ und } i \in \{0, \dots, n\}\}.$$

„“ steht dabei für die leere Zeichenkette. Um zwischen den Erfordernissen der eigentlichen Datenstrukturen und (mehr oder weniger zufälligen) Details ihrer Implementierung in der jeweils verwendeten Programmiersprache unterscheiden zu können, führen wir eine eigene Notation ein. Damit definieren wir dann die benötigten Funktionen auf  $Z$  und  $A$  (siehe Tabelle 6.1). Dabei müssen wir vor allem auf die Unterscheidung zwischen der *Kennzahl* eines Buchstabens innerhalb des Alphabets und seiner *Position* innerhalb eines Textes achten.

**Tabelle 6.1.** Funktionen auf den Mengen  $Z$  und  $A$  Zeichen ( $t \in Z, z \in A, x, y \in N$ )

Funktion	Bedeutung	Randbedingungen und Werte
$\text{Länge}(t)$	Anzahl der Zeichen von $t$	$\text{Länge}(„“) = 0, \text{Länge}(„z“) = 1$
$\text{Teilzeichen}(t, x)$	Zeichen aus $t$ an der Position $x$	$\text{Teilzeichen}(t, x) = „“$ falls $t = „“$ oder $x > \text{Länge}(t)$
$\text{Anhängen}(t, z)$	Zeichenkette, die man durch Anfügen von $z$ am hinteren Ende von $t$ erhält	$\text{Anhängen}(„z_1 \dots z_n“, z) = „z_1 \dots z_n z“;$ $\text{Länge}(\text{Anhängen}(t, z)) = \text{Länge}(t) + 1$
$\text{Kennzahl}(z)$	Kennzahl des Buchstabens $z$ im Alphabet $A$	$0 \leq \text{Kennzahl}(z) \leq \max - 1$
$\text{Buchstabe}(x)$	Buchstabe des Alphabets $A$ zur Kennzahl $x$	$x \in \{0, \dots, \max - 1\}$

Nun können wir unseren Algorithmus sehr kompakt beschreiben, wobei wir die Zeichenketten  $k$  für den Klartext,  $s$  für den Schlüssel und  $g$  für den Geheimtext verwenden:

Eingabe von  $k$  und  $s$ ;

$g := „“;$

Wiederhole für  $i = 0$  bis  $\text{Länge}(k) - 1$ :

$g = \text{Anhängen}(g, \text{Buchstabe}(\text{Kennzahl}(\text{Teilzeichen}(k, i)) + \text{Kennzahl}(\text{Teilzeichen}(s, i \bmod \text{Länge}(s))) \bmod \text{max}))$ ;

Ausgabe von  $g$ .

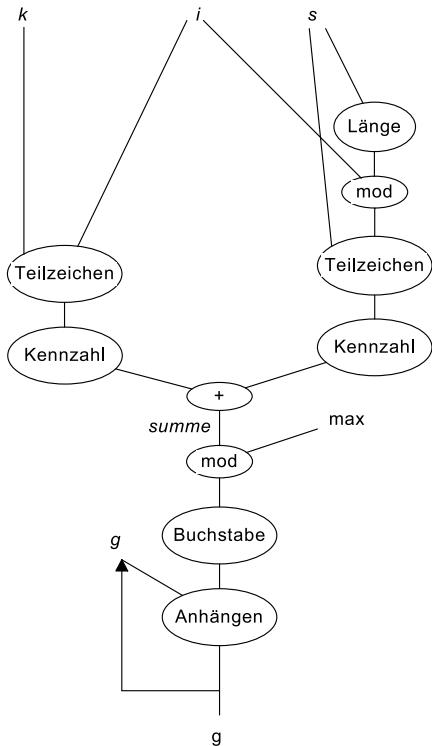


Abb. 6.2. Verfeinerung des Verschlüsselungsprozesses

Im Wesentlichen ist also neben der Ein- und Ausgabe und der Initialisierung nur noch eine Wiederholung und eine Zuweisung übrig geblieben, in deren Term der Rest des Algorithmus verborgen ist. Dieser Term gibt uns Gelegenheit, unser Datenflussdiagramm aus Abb. 6.1 zu verfeinern, indem wir einen Teil der „Glass-Box“-Sicht der Verschlüsselungsfunktion erarbeiten (siehe Abb. 6.2). Dabei symbolisieren wir Funktionen durch Datenverarbeitungsprozesse mit genau einem Ausgangskanal. Schnell wird hier auch die Beschränkung unserer funktionalen Mittel auf die Beschreibung von Termen klar. Die Wiederholungsanweisung können wir wegen mangelnder Darstellungsmöglichkeit für zeitliche Abläufe nicht

modellieren. Lediglich die rekursive Verwendung von  $g$  kann durch einen zyklischen Datenfluss angedeutet werden. Zuweisungen können dagegen durch Bezeichnung des Ausgangskanals einer Funktion mit dem Namen der Variablen auf der linken Seite der Zuweisung angedeutet werden. Auf diese Weise kann man auch die Verwendung von Hilfsvariablen zur Zerlegung des komplexen Terms zwecks besserer Übersicht visualisieren, etwa durch eine Hilfsvariable *summe* für das Ergebnis der Addition in Abb. 6.2, mit der sich unser Algorithmus folgendermaßen schreibt:

```

 $g = „;“;$ 
Wiederhole für  $i=0$  bis Länge( $k$ )-1:
 $summe = Kennzahl(\text{Teilzeichen}(k, i)) + Kennzahl(\text{Teilzeichen}(s, i \text{ mod } \text{Länge}(s)));$ 
 $g = \text{Anhängen}(g, \text{Buchstabe}(summe \text{ mod } \text{max}));$ 
Ausgabe von  $g$ .

```

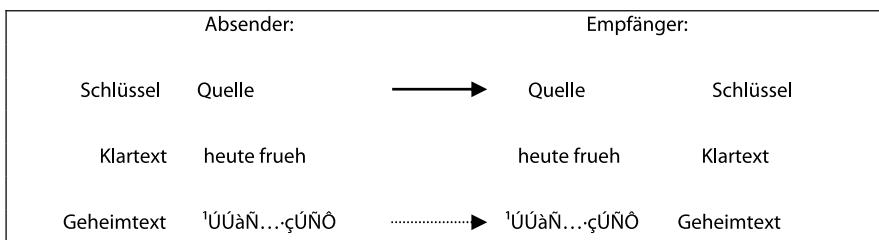
## 6.4 Implementierung

Nach dieser ausführlichen Vorarbeit kann uns die Implementierung nicht mehr schrecken, weil die Nebensächlichkeit der technischen Details der jeweiligen Programmiersprache deutlich zutage tritt. Die Aufgabe der Implementierung reduziert sich auf die Übertragung der erarbeiteten Strukturen in Konstrukte der Programmiersprache. Wir schlagen dafür zur Abwechslung eine makroprogrammierbare Tabellenkalkulation (z.B. *MS-Excel*<sup>®</sup> in Verbindung mit *Visual-Basic*<sup>®</sup>) vor, da wir damit von uninteressanten Ein- und Ausgabedetails (z.B. der lästigen Pascal-Prozedur *gotoxy()* aus Kapitel 4) befreit werden.

**Tabelle 6.2.** Übersetzung von Datenstrukturen und Funktionen

Funktion	in Visual-Basic	Bemerkung
Länge( $t$ )	Len( $t$ )	
Teilzeichen ( $t, x$ )	Mid( $t, x, 1$ )	Mid( $t, 1, 1$ ) entspricht Teilzeichen( $t, 0$ )
Anhängen( $t, z$ )	$t + z$	Überladener Operator, auch zur Verkettung von Zeichenketten verwendbar
Kennzahl( $z$ )	Asc( $z$ )	Ergebnis ist die Position des ersten Zeichens der angegebenen Zeichenfolge im ASCII-Code. z.B. Asc(„A“) = Asc(„Anna“) = 65
Buchstabe( $x$ )	Chr( $x$ )	Ergebnis ist eine Zeichenkette aus dem Zeichen mit dieser ASCII-Nummer, z.B. Chr(66) = „B“

---



**Abb. 6.3.** Das Ergebnis in MS-Excel®

Natürlich müssen wir zuerst unsere Datenstrukturen und Funktionen in die Programmiersprache übersetzen (siehe Tabelle 6.2). An dieser Stelle kann man auch anhand der Basic-Funktionen Asc() und Chr() einige in der Technik verwendete Zuordnungen zwischen Buchstaben und Zahlen (z. B. ASCII-Code) besprechen.

Schließlich setzen wir unser Datenflussdiagramm aus Abb. 6.1 in eine Tabelle um (siehe Abb. 6.3). Die Daten verarbeitenden *Prozesse* (Funktionen) erscheinen darin als Zellen, von denen aus die jeweils benötigte *Funktion* aufgerufen wird, wobei dieser meist gewisse Steuergrößen (*Parameter*) übergeben werden. Wir verwenden die beiden folgenden Visual-Basic-Funktionen *Encode()* und *Decode()*, wobei z.B. *Encode()* die Parameter *s* (für den Schlüssel), *k* (für den Klartext), beide von der Sorte *String* erwartet und als Ausgabewert den Geheimtext zurück liefert:

```

Function Encode(s, k As String) As String
    g = ""
    For i = 0 To Len(k) - 1
        summe = Asc(Mid(k, i + 1, 1)) + Asc(Mid(s, i Mod Len(s) + 1, 1))
        g = g + Chr(summe Mod 255)
    Next
    Encode = g
End Function

Function Decode(s, g As String) As String
    k = ""
    For i = 0 To Len(g) - 1
        diff = Asc(Mid(g, i + 1, 1)) - Asc(Mid(s, i Mod Len(s) + 1, 1))
        k = k + Chr(diff Mod 255)
    Next
    Decode = k
End Function

```

## 6.5 Wertung und Ausblick

An dieser Stelle könnte man z.B. auf die Sicherheit von Verschlüsselungsverfahren (Stichwort Häufigkeitsverteilung der Buchstaben) eingehen und dann wirksamere und damit kompliziertere Methoden zur Textverschlüsselung besprechen

(siehe dazu etwa Bauer (1997)). Ob man das im Jahre 1978 von Rivest, Shamir und Adleman entwickelte RSA-Verfahren (siehe Bauer (1997)) in dieser Jahrgangsstufe erfolgreich vermitteln kann, darf allerdings bezweifelt werden.

Anschließend wäre eine Diskussion der Problematik einer Freigabe wirksamer Verschlüsselungsmethoden durch staatliche Stellen angebracht. Der Staat könnte sich dabei leicht selbst jeder Kontrollmöglichkeit berauben. Das US-Pentagon gibt dazu ein gutes Beispiel für eine ziemlich restriktive Freigabepolitik ab.

# 7 Objektorientierte Modellierung

Eine eingehende Besprechung aller wichtigen didaktischen Aspekte der objektorientierten Modellierung würde vermutlich ein eigenes Buch füllen. Deshalb wollen wir uns hier darauf beschränken, die im Kontext unseres Gesamtvorschlages relevanten Aspekte dieser Modellierungstechnik herauszustellen, zumal die Schüler bereits im Fundamentum (siehe Kapitel 1) und u.U. auch anlässlich der datenorientierten Modellierung (in Kapitel 4) bereits von Klassen, Instanzen und Methoden gehört haben.

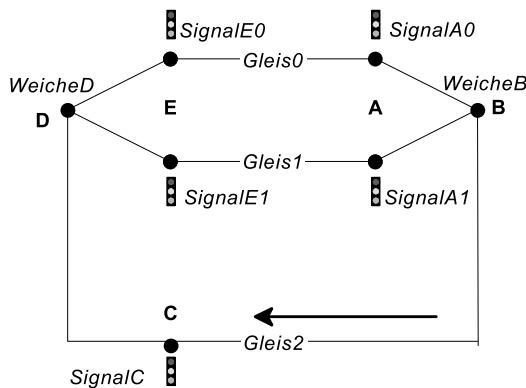
Anhand eines (relativ) einfachen Beispiels aus dem Bereich des Eisenbahnverkehrs wollen wir also die wesentlichen Techniken objektorientierter Modellierung kennen lernen. Die ansonsten oft sehr schwierige Identifikation der beteiligten Objekte stellt hierbei keine größeren Anforderungen an die Schüler, so daß sie sich voll auf die Systematik der Modellierung konzentrieren können. Dabei wollen wir einerseits den Anschluss an die vorausgegangenen Beschreibungstechniken (Zustandsmodellierung, siehe Kapitel 5, und funktionale Modellierung, siehe Kapitel 6) herstellen, andererseits soll der abschließende Übergang zu nebenläufig agierenden Objekten einen Ausblick auf eine mögliche Behandlung paralleler Prozesse in der Oberstufe anbieten. Mit Hilfe von Interaktionsdiagrammen kann dieser Übergang sehr anschaulich beschrieben werden.

Das Modell wird in *Java* implementiert, vor allem weil diese Sprache (auch wenn sie einige didaktische Nachteile aufweist, siehe Abschnitt 3.4.4 in Teil B) einen eleganten Übergang zur Nebenläufigkeit anbietet.

## 7.1 Problemstellung

Die Einführung könnte anlässlich einer Exkursion in einen großen Bahnhof oder der Betrachtung einer kleinen Modelleisenbahnanlage erfolgen. Wir suchen uns einen kleinen Ausschnitt der Anlage und setzen uns das Ziel, den Verkehr von Zügen auf diesem Ausschnitt zu simulieren. Dabei wollen wir nicht eine zentrale Steuerung durch ein Stellwerk o.ä. anstreben, sondern einen automatischen Ablauf, der alleine durch die Kommunikation der beteiligten Objekte ausgehandelt wird. Genau hier liegt der Unterschied der objektorientierten Sichtweise im Vergleich zu einer prozeduralen Modellierung mit Hilfe einer „allmächtigen“ Steuerprozedur.

Anfangs zeichnen wir einen Gleisplan (siehe Abb. 7.1) als Grundlage für die anschließende Diskussion.



**Abb. 7.1.** Der Gleisplan für unser Modell

Nun stellen wir mit Hilfe einer Textverarbeitung alle Randbedingungen in freier Formulierung auf:

Der Zugverkehr verläuft unter Benutzung aller vorhandenen Gleise grundsätzlich im Uhrzeigersinn über die Punkte A–E. Im Bahnhof führen zwei parallele Gleise (Gleis0 und Gleis1) nach je einer Signalanlage (SignalA0, SignalA1) über WeicheB auf ein gemeinsames Gleis2. Dieses kommt über WeicheD zum Bahnhof zurück, wo es wiederum aufgeteilt wird. Die Weiche wird durch SignalC abgesichert, die Einfahrt in den Bahnhof durch die Signale E0, E1.

Auf dieser Anlage wollen wir drei Züge (Zug1, Zug2, Zug3) verkehren lassen, wobei wir von folgender Anfangssituation ausgehen:

SignalA0 steht auf FAHRT, SignalA1 auf HALT, SignalC auf FAHRT, die Signale E0, E1 auf HALT. Zug1 steht auf Gleis0 vor A, Zug2 auf Gleis1 vor A, Zug3 auf Gleis2 vor C. Beide Weichen verbinden Gleis0 und Gleis1.

Dabei schließen wir Nebenläufigkeit zunächst aus: Jedes Objekt, das eine Methode eines anderen Objektes aufruft, führt bis zu deren Ausführung keine eigenen Aktionen aus. Zusätzlich machen wir deutlich, wo wir Abstraktionen vornehmen:

Wenn sich die Züge bewegen, dann immer mit der gleichen konstanten Geschwindigkeit. Beschleunigungs- und Bremsphasen werden vernachlässigt. Die Züge schalten Weichen und Signale selbst. Auf Gleis0 oder Gleis1 darf sich jeweils maximal ein Zug aufhalten. Gleis2 hat dagegen Platz für alle verkehrenden Züge. Für die Verhinderung von Auffahrunfällen soll es darauf ein spezielles Sicherheitssystem (z.B. Induktionssicherung) geben, von dessen Behandlung wir hier absehen.

Am Beispiel von Zug1 überlegen wir uns anschließend den Ablauf eines exemplarischen vollen Umlaufs (als *Szenarios* im Sinne von UML, siehe Booch, Rumbaugh, Jacobson (1997)):

Zug1 startet (passiert A), falls SignalA0 auf FAHRT steht, schaltet dann WeicheB auf Gleis0, SignalA0 und SignalA1 auf HALT sowie SignalE0 auf FAHRT, passiert B, schaltet SignalA1 auf FAHRT, überfährt Punkt C, falls SignalC auf FAHRT steht, schaltet dann SignalC auf HALT und fährt über D wieder in den Bahnhof ein, falls eines der Signale E auf FAHRT steht, wobei er zuvor die Weiche D auf das freie Gleis stellt und SignalC wieder auf FAHRT schaltet. Dann passiert er Punkt E und schaltet das SignalE auf seinem Gleis auf HALT.

Daraufhin müssen wir uns noch die Steuerbedingungen für den Gesamtablauf überlegen, also für die Verknüpfung der Aktivitäten unserer Züge:

Die Züge rücken jeweils um eine Position vor und geben dann (u.U. über eine zentrale Funkleitstelle) dem Zug mit der (zyklisch) nächsten Nummer die Aufforderung zum Vorrücken weiter.

Damit haben wir (implizit) eine *Taktung* des Systems eingeführt.

## 7.2 Modellierung

Nach der informellen Beschreibung sollen die gewonnenen Ergebnisse jetzt formalisiert werden. Als Erstes entwickeln wir dazu ein Modell für die Klassen, Attribute, Methoden und Instanzen (Objektmodell), anschließend modellieren wir das zeitliche Verhalten der Objekte durch Zustands-Übergangsdiagramme, was die Objekte als (*gekapselte*) Automaten erscheinen lässt.

### 7.2.1 Das Objektmodell

Zunächst identifizieren wir die handelnden Objekte:

- drei Züge: Zug1, Zug2, Zug3,
- fünf Signale: A0, A1, C, E0, E1,
- zwei Weichen: B, D.

Diese Identifikation der beteiligten Objekte dürfte nach der ausführlichen Vorarbeit keine Probleme aufwerfen. Anschließend stellen wir fest, dass man diese Objekte nach ihren Eigenschaften zu *Klassen* zusammenfassen kann (siehe Tabelle 7.1).

Die *Attribute* erhalten wir dabei auf einfache Weise aus der Überlegung, welche Informationen über die Objekte für die Steuerung des Ablaufs benötigt werden. Die *Methoden* ergeben sich zum Teil aus der Notwendigkeit des Zugriffs auf die Attribute unter Einhaltung der Kapselungsbedingung, zum anderen aus dem Aktionsmuster des Gesamtablaufs.

Diese Modellierungstechnik ist den Schülern bereits seit längerem (aus dem Fundamentum, siehe Kapitel 1) bekannt. Auch die Bezeichnung „Instanz“ für die Ausprägung einer Klasse haben sie dort kennen gelernt. Falls man bei der vorausgegangenen Behandlung von Datenbanksystemen (siehe Kapitel 3) ebenfalls ob-

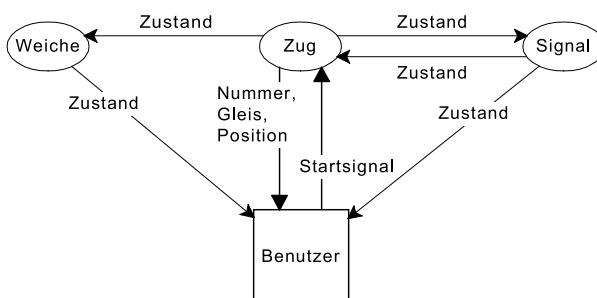
jeorientierte Bezeichnungen verwendet hat, erweist sich das an dieser Stelle als sehr nützlich. Andernfalls sollte man die entsprechenden Datenbankbezeichnungen jetzt den objektorientierten gegenüberstellen: „Klasse“ statt „Entitätsmenge“ bzw. „Tabelle“, „Objekt“ statt „Entität“ bzw. „Datensatz“.

**Tabelle 7.1.** Die Klassen unseres Modells

Klasse	Attribute	Methoden	Objekte
Zug	Nummer, Gleis, Position	Position mitteilen, Vorrücken	Zug1, Zug2, Zug3
Signal	Zustand	Zustand mitteilen, Zustand setzen	SignalA0, SignalA1, SignalC, SignalE0, SignalE1
Weiche	Zustand	Zustand mitteilen, Zustand setzen	WeicheB, WeicheD

Im nächsten Abstraktionsschritt könnten wir die gemeinsamen Eigenschaften verschiedener Klassen herausarbeiten und Oberklassen bilden, falls wir Klassenhierarchien an dieser Stelle behandeln wollen. Wir stellen dabei fest, dass die Klassen *Signal* und *Weiche* einiges gemeinsam haben, nämlich je zwei Zustände sowie Methoden zum Setzen und Lesen der beiden Zustände. Es wäre also zu überlegen, ob wir eine gemeinsame Oberklasse *Steuerelement* einführen.

Anschließend beschreiben wir mit einem Diagramm den Datenfluss zwischen den Subsystemen (Prozessen), die in unserem Fall identisch mit den Objekten sind. Allerdings notieren wir die Datenflüsse zwischen den Klassen (siehe Abb. 7.2), da wir sie sonst (d.h. im Diagramm der Instanzen) zwischen jeder Instanz der Senderklasse und jeder Instanz der Empfängerklasse einzeichnen müssten. Dabei lassen wir die Kommunikation zwischen verschiedenen Instanzen der Klasse *Zug* (Methode „Aktivieren“) zunächst außer Acht.

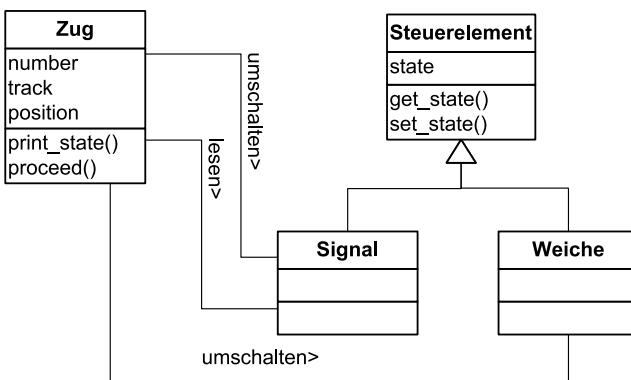


**Abb. 7.2.** Kommunikation zwischen den Objekten

Bevor wir unser Objektdiagramm erstellen, systematisieren wir nochmals die Elemente des Objektmodells:

- Aus (abstrakten) *Klassen* werden (konkrete) *Instanzen* gebildet. Erst diese weisen Werte für die Attribute auf und können Handlungen vornehmen.
- Einige Klassen können u.U. zu (gemeinsamen) *Oberklassen* verallgemeinert werden. Die Unterklassen *erben* dabei die Attribute und Methoden der Oberklassen.
- Objekte kommunizieren über *Botschaften*: Dabei ruft ein Objekt (Sender) eine Methode eines anderen Objektes (Empfänger) auf. Diese Botschaften notieren wir wiederum (gestrichelt) zwischen den *Klassen* (analog zu den Datenflüssen in Abb. 7.2), obwohl sie eigentlich zwischen *Instanzen* ausgetauscht werden.

Jetzt zeichnen wir das vollständige Klassendiagramm (siehe Abb. 7.3).



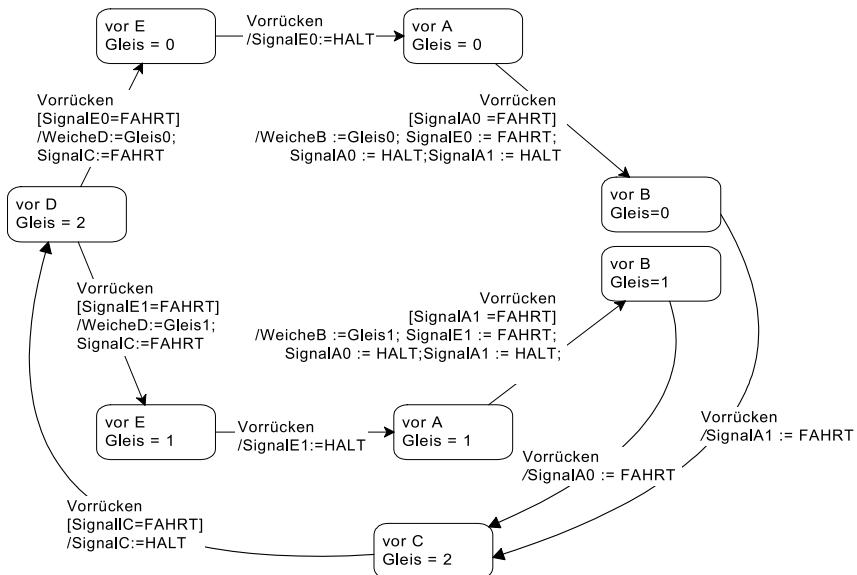
**Abb. 7.3.** Objektmodell der Eisenbahn

## 7.2.2 Zeitliche Abläufe

Unser Objektmodell enthält (bisher) noch keine Aussagen über den zeitlichen Ablauf der einzelnen Methoden, den wir im Folgenden mit Hilfe von Zustands-Übergangsdiagrammen (siehe Kapitel 4) festlegen. Besonders einfach gestalten sich die Diagramme für Signale und Weichen, auf die wir deshalb an dieser Stelle verzichten. Beim dynamischen Modell der Züge müssen wir gedanklich etwas mehr investieren (siehe Abb. 7.4). Zur Erinnerung: Die Transitionen werden markiert mit

1. der auslösenden Aktion,
2. ggf. einer Übergangsbedingung (in eckigen Klammern) und
3. einer durch die Transition ausgelösten Aktion bzw. Folge von Aktionen (nach Schrägstrich)

Auslösende Aktion ist in unserem Fall für alle Transitionen das Signal „Vorrücken“. Um Modell und Implementierung gegeneinander abzugrenzen, verwenden wir hier bewusst noch nicht die (englischen) Bezeichnungen aus dem späteren Programm.



**Abb. 7.4.** Das Zustandsmodell der Klasse Zug.

## 7.3 Implementierung

Nun setzen wir unser Modell mittels einer geeigneten (möglichst objektorientierten) Sprache in ein lauffähiges Programm um. Wir stellen dabei nur eine sehr einfache Implementierung (als *Java-Applikation*) vor, wobei wir auf alle nicht unbedingt nötigen Komplikationen wie z.B. Vererbung oder grafische Ausgabe verzichtet haben, um die Programmstruktur möglichst deutlich hervortreten zu lassen.

Es folgt der Programmcode, der gemäß der Konventionen von *Java* in fünf Dateien (eine je Klasse) abgelegt wird. Die Klasse *Input* dient zur Entlastung der Schüler von (hier) uninteressanten technischen Details der Programmiersprache (wie *Eingabeströmen* und *Java-Exceptions*) und sollte deshalb von der Lehrkraft erstellt werden:

```
package Eisenbahn;  
import java.io.*;  
public class Input {  
    public Input() {  
    }  
    public static void readkey(){  
        try {
```

---

```

        int val = System.in.read();
    }
    catch (IOException e){
    } } }
```

Zur Sicherstellung des gegenseitigen Zugriffs werden die Klassen zu einem *package* zusammengefasst. Außerdem führen wir eine aus technischen Gründen nötige (Hauptklasse) *Railway* ein, von der aus die anderen Objekte angelegt und aktiviert werden:

```

package Eisenbahn;
public class Railway {
    static byte b;
    public static Signal[] signalA = {new Signal("FAHRT"),new Signal("HALT!")};
    public static Signal signalC = new Signal("FAHRT");
    public static Signal[] signalE = {new Signal("HALT!"),new Signal("HALT!")};
    public static Switch switchB = new Switch(0);
    public static Switch switchD = new Switch(0);
    public static Train[] train = {new Train(1,0,'A'),new Train(2,1,'A'),
        newTrain(3,2,'D')};
    public static void main(String argv[]){
        System.out.println("Zug,Gleis,Pos.| Signale A0, A1, C, E0, E1 | Weichen B, C");
        for (int i=0; i<=6; i++){
            for (int zugnr=0; zugnr <=2; zugnr++){
                train[zugnr].proceed();
            }
            Input.readkey();
        }
    }
}
```

Nach diesen beiden aus technischen Gründen notwendigen Klassen folgen jetzt die durch das eigentliche *Modell* vorgegebenen Klassen *Switch*, *Signal* und *Train*:

```

public class Switch{
    public int state;
    public Switch (int s){
        state = s;
    }
    public void set_state(String s){
        state = s;
    }
    public int get_state(){
        return state;
    }
}
public class Signal{
    public String state;
    public Signal (String s){
        state = s;
    }
    public void set_state(String s){
        state = s;
```

```
        }
        public String get_state(){
            return state;
        }
        public void set_state(int s){
            state = s;
        }

public class Train {
    int number;
    int track;
    char position;
    int other_track;

    public Train(int n, int t, char p) {
        number = n;
        track = t;
        position = p;
    }
    public void proceed(){
        switch (position){
            case 'A':
                other_track = (track + 1)% 2;
                if (Railway.signalA[track].get_state() == "FAHRT"){
                    Railway.switchB.set_state(track);
                    Railway.signalE[track].set_state("FAHRT");
                    Railway.signalA[track].set_state("HALT!");
                    Railway.signalA[other_track].set_state("HALT!");
                    position = 'B';
                }
                break;
            case 'B':
                other_track = (track + 1)% 2;
                Railway.signalA[other_track].set_state("FAHRT");
                position = 'C';
                track = 2;
                break;
            case 'C':
                if (Railway.signalC.get_state() == "FAHRT"){
                    Railway.signalC.set_state("HALT!");
                    position = 'D';
                }
                break;
            case 'D':
                if (Railway.signalE[0].get_state() == "FAHRT"){
                    track = 0;
                    Railway.switchD.set_state(track);
                    Railway.signalC.set_state("FAHRT");
                    position = 'E';
                }
        }
    }
}
```

```

if (Railway.signalE[1].get_state() == "FAHRT"){
    track = 1;
    Railway.switchD.set_state(track);
    Railway.signalC.set_state("FAHRT");
    position = 'E';
}
break;
case 'E':
    Railway.signalE[track].set_state("HALT!");
    position = 'A';
    break;
}
print_state();
}
public void print_state(){
System.out.print(number + " auf " + track + " vor " + position + " | ");
System.out.print(Railway.signalA[0].get_state() + "
+ Railway.signalA[1].get_state() + " + Railway.signalC.get_state());
System.out.print(" " + Railway.signalE[0].get_state() + "
+ Railway.signalE[1].get_state() + " | ");
System.out.println(Railway.switchB.get_state() + " " + Railway.switchD.get_state());
}}

```

Den Ablauf des Programms zeigt Abb.7.5 (S. 220).

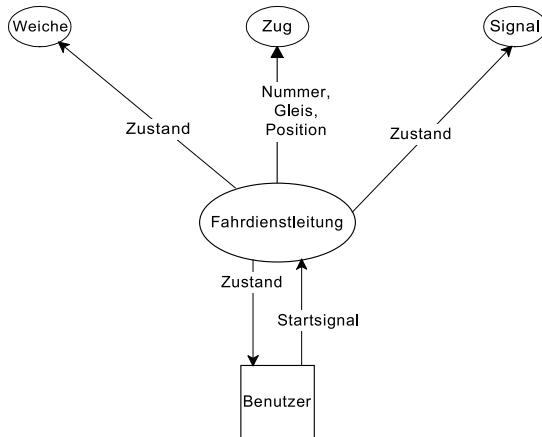
## 7.4 Wertung

Nach einigen Probelaufen diskutieren wir die üblichen Fragen zu unserem System, z.B.:

- Wo ist unser Modell verbesserungswürdig?
- An welchen Stellen sind wir bei der Implementierung vom Modell abgewichen?
- Welche Gefahren, Konsequenzen etc. bringt unser System mit sich?
- Wie hoch waren die Kosten für die Erstellung?

Zug, Gleis, Pos.	Signale A0, A1, C, E0, E1	Weichen B, C
1 auf 0 vor B	HALT! HALT! FAHRT FAHRT HALT!	0 0
2 auf 1 vor A	HALT! HALT! FAHRT FAHRT HALT!	0 0
3 auf 0 vor E	HALT! HALT! FAHRT FAHRT HALT!	0 0
1 auf 2 vor C	HALT! FAHRT FAHRT FAHRT HALT!	0 0
2 auf 1 vor B	HALT! HALT! FAHRT FAHRT FAHRT	1 0
3 auf 0 vor A	HALT! HALT! FAHRT HALT! FAHRT	1 0
1 auf 2 vor D	HALT! HALT! HALT! HALT! FAHRT	1 0
2 auf 2 vor C	FAHRT HALT! HALT! HALT! FAHRT	1 0
3 auf 0 vor B	HALT! HALT! HALT! FAHRT FAHRT	0 0
1 auf 1 vor E	HALT! HALT! FAHRT FAHRT FAHRT	0 1
2 auf 2 vor D	HALT! HALT! HALT! FAHRT FAHRT	0 1
3 auf 2 vor C	HALT! FAHRT HALT! FAHRT FAHRT	0 1
1 auf 1 vor A	HALT! FAHRT HALT! FAHRT HALT!	0 1
2 auf 0 vor E	HALT! FAHRT FAHRT FAHRT HALT!	0 0

3 auf 2 vor D	HALT! FAHRT HALT! FAHRT HALT!	0 0
1 auf 1 vor B	HALT! HALT! HALT! FAHRT FAHRT	1 0
2 auf 0 vor A	HALT! HALT! HALT! HALT! FAHRT	1 0
3 auf 1 vor E	HALT! HALT! FAHRT HALT! FAHRT	1 1
1 auf 2 vor C	FAHRT HALT! FAHRT HALT! FAHRT	1 1
2 auf 0 vor B	HALT! HALT! FAHRT FAHRT FAHRT	0 1
3 auf 1 vor A	HALT! HALT! FAHRT FAHRT HALT!	0 1

**Abb.7.5.** Ein Ablauf unseres Systems**Abb. 7.6.** Prozesse und Datenflüsse bei prozeduraler Modellierung

Neben der Beantwortung dieser allgemeinen Fragen sollte man an dieser Stelle auch den Unterschied zwischen prozeduraler und objektorientierter Sichtweise deutlich machen: Auch wenn wir hier ein „übergeordnetes“ Objekt zum Anstoß der anderen Objekte benötigen, so geht dessen Einfluss doch nicht so weit wie es bei prozeduraler Modellierung der Fall wäre. In unserem System entscheiden die Objekte aufgrund der Informationen, die sie durch die Kommunikation mit den anderen Objekten erhalten, wie sie auf den Anstoß zum Vorrücken reagieren. Die steuernde Klasse RAILWAY gibt im Wesentlichen nur den Takt und die Einsätze vor, wie der Dirigent eines Orchesters.

Aus prozeduraler Sicht hätten wir dagegen letztlich aus einem Algorithmus für den Gesamtablauf *eine* große, zentrale Steuerprozedur erhalten (siehe Abb. 7.6), die im Bild des Orchesters alle Instrumente selbst gespielt hätte.

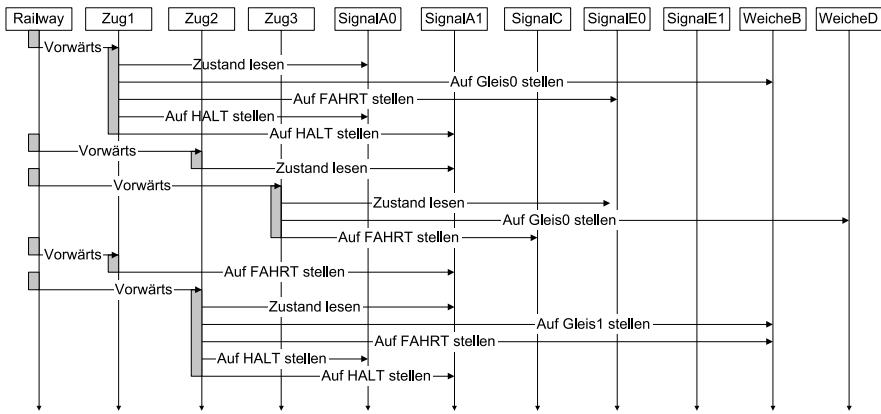


Abb. 7.7. Sequenzdiagramm unserer Eisenbahn

Wenn man vom Idealbild selbständiger *Agenten* ausgeht, dann mangelt es den Objekten in unserer Implementierung noch erheblich an „Handlungsfreiheit“, denn sie stellen ihre Aktivität nach jedem Vorrücken wieder ein und übergeben die Ablaufkontrolle an das Steuerobjekt, das dann als *Dispatcher* das nächste Objekt in der Reihe aufruft. Wie bei einem Staffellauf ist zu einem bestimmten Zeitpunkt immer nur *ein* Objekt (das mit dem Stab) aktiv. Diese Aktivitätsverteilung kann man in einem *Sequenzdiagramm* visualisieren (siehe Abb. 7.7). Darin zeichnen wir die beteiligten Objekte in der Kopfleiste nebeneinander und hängen an jedes Objekt eine senkrecht nach unten verlaufende Zeitleiste an. So werden aus Objekten *Prozesse*. Botschaften zwischen Objekten werden dann (unendlich kurze Laufzeit angenommen) mit waagrechten Pfeilen zwischen Sender und Empfänger symbolisiert. Die Aktivität von Objekten wird durch graue Rechtecke an den Zeitleisten symbolisiert.

Aus dieser Beschränkung auf strenge Sequentialität resultiert natürlich die Frage, ob es nicht auch andere Möglichkeiten gibt. Warum können nicht mehrere Objekte gleichzeitig aktiv sein? Das führt uns zwangsläufig zu parallelen Prozessen.

## 7.5 Nebenläufigkeit

Abschließend setzen wir uns das Ziel, die Züge gleichzeitig aktiv werden zu lassen. Dafür benötigen wir sowohl einige neue begriffliche Voraussetzungen als auch spezielle programmiertechnische Konzepte, die leider nicht in allen Programmiersprachen zu finden sind. Falls die verwendete Sprache (wie Pascal) keine Nebenläufigkeit zulässt, kann man sich oft damit behelfen, mehrere Instanzen desselben Programms gleichzeitig zu aktivieren und damit auf Betriebssystemebene (quasi-)parallel ablaufen zu lassen. Die Kommunikation zwischen diesen Prozessen muss dann natürlich auch auf die Ebene des Betriebssystems (etwa durch

gemeinsam benutzte Dateien oder Umgebungsvariable) verlagert werden (siehe dazu Hubwieser u. Broy (1996)).

### 7.5.1 Begriffsklärungen

Vor (oder während) der Beschäftigung mit nebenläufigen Prozessen müssen mit den Schülern die neuen Begriffe geklärt werden:

**Ereignisse und Aktionen.** *Ereignisse* sind Vorgänge, die zu einzelnen Raum-Zeitpunkten abstrahiert wurden. Diese treten naturgemäß nur einmalig auf, weshalb wir für gleichartige Ereignisse (z.B. in zyklischen Prozessen) eine weitere Abstraktion (analog zur Bildung von Klassen aus einer Menge von Objekten) benötigen. Wir fassen eine Menge von Ereignissen mit gemeinsamen Merkmalen zu einer *Aktion* zusammen. Damit kann ein Ereignis als eine *Instanz* einer Aktion betrachtet werden. In unserem Beispiel wäre eine Aktion etwa das Umstellen von SignalA1 auf "HALT". Die Instanzen dieser Aktion zu den Zeitpunkten der ersten, zweiten bzw. dritten Überquerung von B durch Zug 1 sind dagegen Beispiele für Ereignisse.

**Kausale Beziehungen.** Ereignisse (bzw. Aktionen) stehen oft in kausaler Beziehung zueinander. So muss ein Ereignis der Aktion WeicheB:=Gleis1 (Ereignis  $e_1$ ) abgeschlossen sein, bevor Zug 2 den Punkt B passieren kann (Ereignis  $e_2$ ). Wir sagen dann: „ $e_1$  ist Voraussetzung für  $e_2$ “.

**Prozess.** Ein *Prozess* besteht aus einer Menge von Ereignissen, die untereinander in kausaler Beziehung stehen (können). Wenn der Prozess nicht nur einmalig stattfinden soll, ist es sinnvoll, die den Ereignissen zugeordneten Aktionen zu betrachten. Wir haben es also meist mit *Aktionsstrukturen* zu tun. In unserem Beispiel stehen die Ereignisse während eines Umlaufes der beiden Züge zueinander in vielfältiger kausaler Beziehung.

**Teilprozess.** Ein Ausschnitt einer Aktionsstruktur heißt *Teilprozess*. Die Fahrt eines einzelnen Zuges stellt einen Teilprozess der gesamten Ablaufstruktur dar.

Mit Hilfe dieser Begriffe sind wir nun in der Lage, Nebenläufigkeit und Parallelität von Ereignissen und Prozessen zu besprechen:

**Nebenläufigkeit.** Wir sprechen von *nebenläufigen* Prozessen, wenn diese unabhängig voneinander sowohl nacheinander in beliebiger Reihenfolge als auch mit zeitlicher Überschneidung ausgeführt werden können.

**Parallelität.** Zwei Ereignisse heißen *parallel*, wenn Sie kausal unabhängig voneinander sind. Ein *Prozess* heißt parallel, wenn er parallele Ereignisse enthält. Unter Umständen kann man mehrere parallele Ereignisse zu parallelen Teilprozessen zusammenfassen. In unserem Fall könnten Zug1 und Zug2 zu parallelen Teilprozessen (des Gesamtprozesses) werden.

**Synchronisation:** Parallele Teilprozesse greifen oft auf gemeinsame (nur einmal vorhandene) *Ressourcen* zu. Dann müssen ihre Abläufe u.U. synchronisiert werden. In unserem Beispiel benutzen Zug1 und Zug2 gemeinsam die beiden Weichen und das Gleis2, können diese also nicht gleichzeitig überfahren.

**Semaphore** (Ampeln) und **Monitore**. Zur Synchronisation von Prozessen gibt es verschiedene Verfahren. Eine Möglichkeit ist die Einrichtung *gemeinsamer Variabler* (so genannter Semaphore), deren Belegung über die Zulässigkeit bestimmter Aktionen entscheidet. Eine alternative Möglichkeit bietet das Monitorkonzept, bei dem eine übergeordnete Prozedur (Monitor) den Zugriff auf gemeinsame Betriebsmittel regelt. In unserem Beispiel stellen die Signale eine Art Semaphore dar.

**Verklemmung.** Zwei Prozesse warten auf eine Aktion des jeweils anderen Prozesses. Notwendige (aber nicht hinreichende) Voraussetzung für das Auftreten einer Verklemmung ist eine wechselseitige Kausalbeziehung zwischen Ereignissen beider Prozesse.

### 7.5.2 Implementierung paralleler Prozesse

Java bietet mit dem *Thread*-Konzept eine elegante Möglichkeit zur Realisierung paralleler Prozesse. Diese *Threads* (engl. für „Fäden“) sind nebenläufig ausführbare Programmroutine, die allerdings im Gegensatz zu nebenläufigen *Prozessen* (im Sinne des Betriebssystems) in einem gemeinsamen Speicherbereich ablaufen (siehe etwa Tanenbaum (1995)). In unserem Fall wollen wir die Objekte der Klasse Zug zu Threads machen und ihnen damit erlauben, gleichzeitig aktiv zu sein. Natürlich ist diese „gleichzeitige“ Aktivität auf einem Einprozessorsystem nur eine Quasiparallelität und damit letztlich auf Maschinenebene auch nur eine (wenn auch eine sehr lehrreiche) Illusion.

Die Synchronisation der Zugriffe von Threads auf die Semaphore wird durch das interne *Monitorkonzept* von Java erleichtert. Mit dem Schlüsselwort *synchronized* können Programmabschnitte so gesperrt werden, dass zu einem bestimmten Zeitpunkt nur jeweils ein Thread darauf zugreifen kann. So kann z.B. der exklusive Schreib- und Lesezugriff der Threads auf Variablen gesichert werden. Ein didaktisches Problem stellt bei Threads die oft unumgängliche Mehrfachvererbung dar, die wir an dieser Stelle jedoch vermeiden.

Es folgen die Ausschnitte der Programmlistings, in denen sich durch die Einführung von Nebenläufigkeit Änderungen (kursiv gesetzt) ergeben. In der Klasse *Railway* müssen die Zug-Objekte jetzt nur noch einmalig aktiviert werden:

```
public class Railway {
    ..
    public static void main(String argv[]){
        System.out.println("Zug,Gleis,Pos.| Signale A0, A1, C, E0, E1 | Weichen B, C");
        train[0].start();
        train[1].start();
        train[2].start();
```

```

Input.readkey();
train[0].stop();
train[1].stop();
train[2].stop();
}
}

```

In den Klassen *Switch* und *Signal* muss der exklusive Schreibzugriff der Threads auf die inneren Zustände durch den Zusatz *synchronized* sichergestellt werden:

```

public synchronized void set_state(String s){
    state =s;
}

```

Schließlich wird die Klasse *Zug* als Unterklasse von *Thread* angelegt. Die zentrale Methode heißt konventionsgemäß *run()*, die nun (z.B. durch eine Zählerschleife) selbst für einen Abbruch des Ablaufs sorgen muss. Schließlich muss auch in dieser Klasse an den entscheidenden Stellen der Ablaufsteuerung und der Ausgabe (durch das Schlüsselwort *synchronized*) ein exklusiver Zugriff gewährleistet sein.

```

public class Train extends Thread {
    public void run(){
        for (int counter = 0; counter < 6; counter++) {
            synchronized (this){
                switch (position){
                    ..
                }
                print_state();
            }
        }
    }

    public void print_state(){
        synchronized (System.out){
            ..
        }
    }
}

```

### 7.5.3 Wertung

Nach diesen Änderungen lassen wir unser Programm einige Male ablaufen, wobei wir eine erstaunliche Entdeckung machen (siehe Abb. 7.8): Wir stellen fest, dass es plötzlich zu verschiedenen Reihenfolgen der Aktionen bei der Abarbeitung des Programms kommen kann.

Ein Rückgriff auf den alten, sequentiellen Programmcode bestätigt, dass dieser immer den gleichen Ablauf liefert. Woher kommt dieser Nichtdeterminismus? Die Antwort liefert eine sorgfältige Interpretation des Sequenzdiagramms für die parallele Variante unseres Programms (siehe Abb. 7.9): Durch die gleichzeitige Aktivität der Zugobjekte ist es an vielen Stellen dem Zufall überlassen, welcher Thread zuerst eine für ihn mögliche Botschaft absetzt. Bei mehreren Abläufen kommt es daher zufallsbedingt zu unterschiedlichen Abfolgen der Aktionen.

Zug, Gleis, Pos.	Signale A0, A1, C, E0, E1	Weichen B, C
3 auf 2 vor D	FAHRT HALT! FAHRT HALT! HALT!	0 0
1 auf 0 vor B	HALT! HALT! FAHRT FAHRT HALT!	0 0
1 auf 2 vor C	HALT! FAHRT FAHRT FAHRT HALT!	0 0
2 auf 1 vor A	HALT! FAHRT HALT! FAHRT HALT!	0 0
2 auf 1 vor B	HALT! HALT! HALT! FAHRT FAHRT	1 0
3 auf 0 vor E	FAHRT HALT! HALT! FAHRT FAHRT	1 0
3 auf 0 vor A	FAHRT HALT! HALT! HALT! FAHRT	1 0
1 auf 2 vor D	HALT! HALT! FAHRT FAHRT	0 0
...		

Zug, Gleis, Pos.	Signale A0, A1, C, E0, E1	Weichen B, C
1 auf 0 vor B	HALT! HALT! FAHRT FAHRT HALT!	0 0
1 auf 2 vor C	HALT! FAHRT FAHRT FAHRT HALT!	0 0
1 auf 2 vor D	HALT! FAHRT HALT! FAHRT HALT!	0 0
1 auf 0 vor E	HALT! FAHRT FAHRT FAHRT HALT!	0 0
1 auf 0 vor A	HALT! FAHRT FAHRT HALT! HALT!	0 0
1 auf 0 vor A	HALT! FAHRT FAHRT HALT! FAHRT	1 1
2 auf 1 vor B	HALT! HALT! FAHRT HALT! FAHRT	1 1
...		

Abb. 7.8. Zwei verschiedene Abläufe unseres parallelisierten Systems (Ausschnitt)

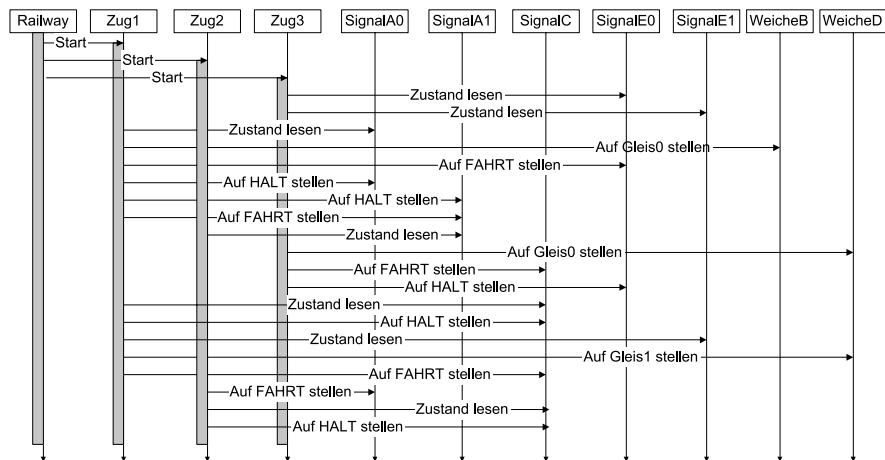


Abb. 7.9. Ein Ablauf des parallelen Systems

# 8 Rekursive Datenstrukturen

Nun sollen einige Unterrichtsvorschläge für Grund- und Leistungskurse der gymnasialen Oberstufe vorgestellt werden. Wegen des großen Umfangs des Lernstoffs, der in den beiden Jahrgängen der Oberstufe untergebracht werden kann, würde die Darstellung einer vollständigen Folge von Unterrichtsprojekten für die gesamte Oberstufe den Rahmen dieser Arbeit bei weitem sprengen. Wir wollen hier deshalb nur zu jedem Kurshalbjahr ein mehr oder weniger detailliertes Schlaglicht auf das Unterrichtsgeschehen werfen, um dem Leser eine gewisse Vorstellung von den Möglichkeiten der Oberstufeninformatik zu vermitteln.

In der 10. und 11. Jahrgangsstufe haben sich die Schüler ausgiebig mit diversen Modellierungstechniken beschäftigt. Dabei wurden vor allem Diagramme (Datentransfuss-, Zustands-Übergangs-, Objekt-, Interaktionsdiagramme) zur Veranschaulichung der Modelle eingesetzt, bei denen es sich meist um Graphen mit speziellen Formen und Markierungen für Ecken (Knoten) und Kanten handelte. Daher liegt es nun nahe, sich allgemein mit Graphen zu beschäftigen.

Allerdings sollte auch hier das Prinzip der Informationsorientierung eingehalten werden: Die Graphen dürfen niemals um ihrer selbst oder gewisser mathematischer Aspekte willen, sondern immer nur als Mittel zur Repräsentation von Information betrachtet werden.

Anfangs wird man sich mit einfachen Graphen beschäftigen, am besten wohl mit linearen Listen. Bei deren Implementierung wird an auf natürliche Weise zu rekursiven Datenstrukturen und damit zu rekursiven Algorithmen geführt.

Ein geeignetes Implementierungsmittel für wäre eine funktionale Programmiersprache wie etwa *gofer*, deren Feinheiten dabei allerdings unbedingt ausser acht gelassen werden sollten. Darüber hinaus sind die Schüler bereits mit einer objekt-orientierten Sprache, z.B. *Java*, vertraut (siehe Kapitel 7), die deshalb ebenfalls eingesetzt werden kann.

## 8.1 Aufgabenstellung und Lernziele

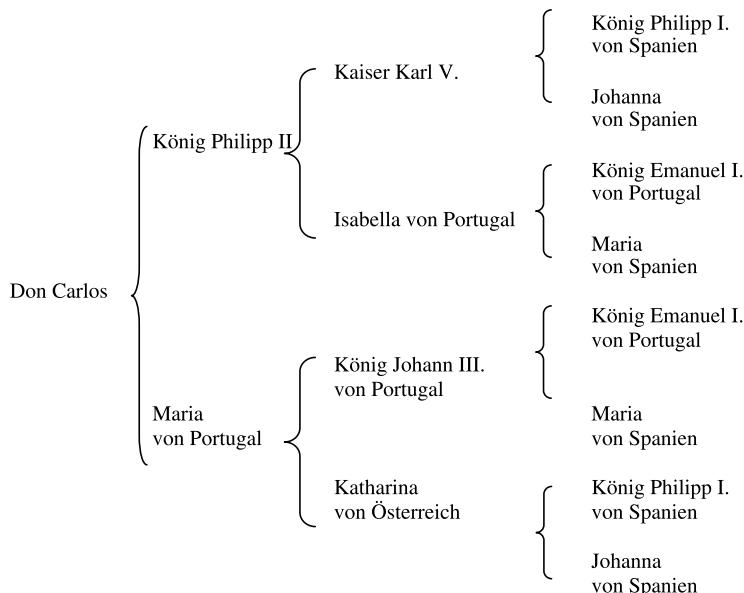
Wir wollen nun eine Unterrichtssequenz vorstellen, die sich, aufbauend auf den Kenntnissen über Listen, mit binären Bäumen beschäftigt.

Dafür suchen wir uns eine geeignete Informationsstruktur, etwa den Stammbaum des Don Carlos aus Bauer u. Goos (1984), wie er in Abb. 8.1 skizziert ist.

Geeignete Aufgabenstellungen für die folgende Unterrichtssequenz wären etwa:

1. Zähle alle Vorfahren von Don Carlos auf!
2. Befindet sich eine gewisse Person x unter den Vorfahren von Don Carlos?

3. Gibt es Personen, die mehrfach im Stammbaum auftauchen?
4. Nenne für jede genannte Person jeweils ihre Eltern



**Abb. 8.1.** Der Stammbaum des Don Carlos (aus Bauer u. Goos (1984)).

Mit diesem Projekt verfolgen wir eine ganze Reihe von Lernzielen. So sollen die Schülerinnen und Schüler:

- Rekursionen in Datenstrukturen erkennen, mit graphischen Mitteln beschreiben und deren Konsequenzen verstehen,
- eine rekursive Datenstruktur zur Darstellung von binären Bäumen kennen, anwenden und implementieren können,
- den Zusammenhang zwischen Relationen und Graphen kennen,
- in der Lage sein, Algorithmen zum Aufzählen der Knoten eines Baumes in verschiedenen Reihenfolgen entwerfen und implementieren können.

## 8.2 Problembeschreibung

Für die Lösung jeder der o.g. Aufgabenstellungen müssen die Knoten der Bäume vollständig (in einer beliebigen Reihenfolge) aufgezählt werden. Dabei soll die Tiefe des Baumes nicht von vornherein auf einen festen Wert begrenzt sein. Bei der umgangssprachlichen Formulierung der Wege durch den Baum stellen wir

fest, dass wir uns bei der Abarbeitung jedes Knotens für eine der drei möglichen Reihenfolgen entscheiden müssen:

1. Knoten, linker Teilbaum, rechter Teilbaum (Vorordnung des Knotens),
2. linker Teilbaum, Knoten, rechter Teilbaum (Inordnung),
3. linker Teilbaum, rechter Teilbaum, Knoten (Nachordnung).

Die informelle Formulierung der benötigten Algorithmen bereitet uns ungewohnte Schwierigkeiten, weshalb wir an dieser Stelle darauf verzichten. Offensichtlich fehlt uns dazu noch ein gedankliches Konzept, das wir über die folgende Formalisierung zu gewinnen hoffen.

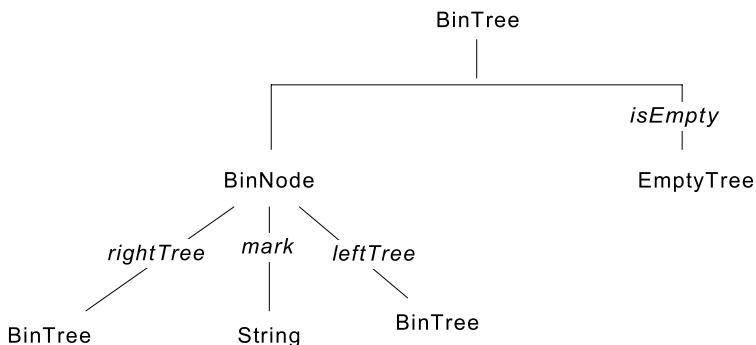
## 8.3 Formale Beschreibung

Zunächst wollen wir die vorliegende Struktur mit mathematischen Mitteln beschreiben: Der Baum wird offensichtlich durch eine zweistellige Relation (Relationen sind den Schülern bereits aus der Mittelstufe bekannt, siehe Kapitel 3)

$R \equiv \text{„ist Elternteil von“}$  mit  $R \subset \mathbf{Personenname} \times \mathbf{Personenname}$

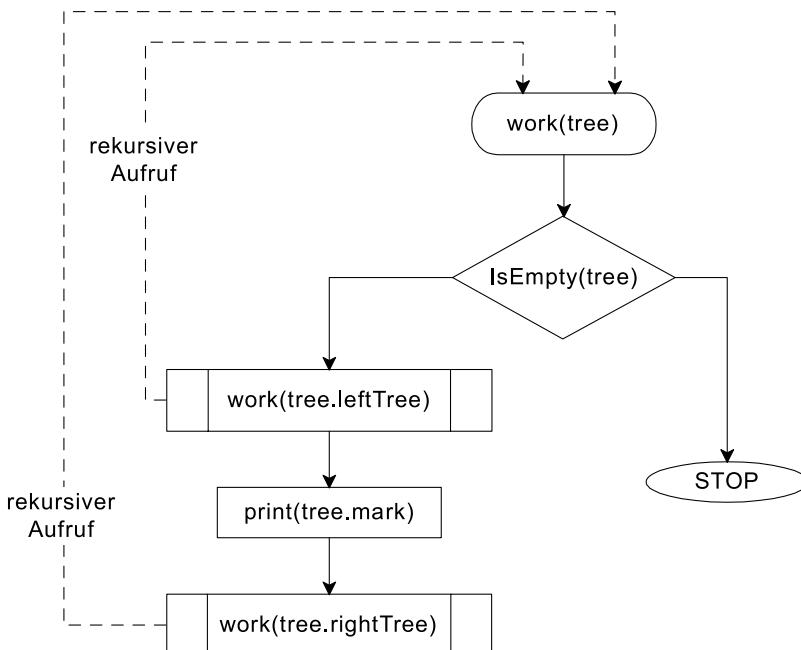
erzeugt, wobei wir, ausgehend von der Wurzel „Don Carlos“ nur die Personen aufnehmen, von denen er (direkt oder indirekt) abstammt und nach einer bestimmten Anzahl von Generationen abbrechen. Aus einer zweistelligen Relation kann man i.a. einen Graphen aufbauen, indem man die Komponenten der Relationspaare als Knoten zeichnet und alle Paare von Knoten miteinander verbindet, die in  $R$  enthalten sind.

Aus der Tatsache, dass jeder Mensch (zumindest biologisch) genau einen Vater und genau eine Mutter hat, ergibt sich die in unserem Fall vorliegende (vollständige) binäre Baumstruktur, bei der an jedem Knoten (Person) genau zwei Teilbäume hängen. Dies gilt allerdings nur, wenn man mehrfach vorkommende Personen auch mehrfach (d.h. an verschiedenen Knoten) einzeichnet, was man als Merkmal für Inzucht benutzen könnte. Verzichtet man dagegen auf diese Multiplizität, so ist der Graph nicht mehr minimal zusammenhängend und damit kein Baum mehr. Nun beschreiben wir die betrachtete Datenstruktur mit Hilfe der aus Abschnitt 2.4.1. bekannten Notationstechnik für Datenstrukturen (siehe Abb. 8.2). Charakteristisch ist die rekursive Verwendung des Typs *BinTree*, die wiederum zur Verwaltung einer Variante für leere Bäume (*EmptyTree* mit der Diskriminatorenfunktion *isEmpty*) zwingt, um ein Abbruchkriterium für die Rekursion zu erhalten.

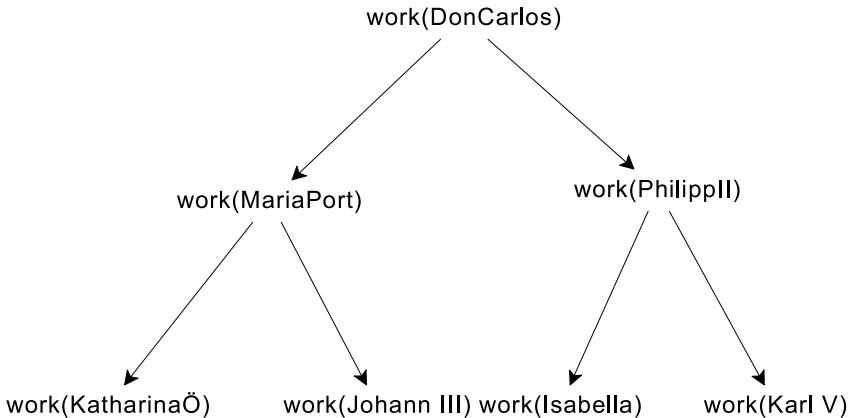
**Abb. 8.2.** Datenstruktur für binäre Bäume

Mit der Rekursivität dieser Struktur können wir uns jetzt auch die Probleme bei der informellen Beschreibung eines Verarbeitungsverfahrens erklären.

Nun erarbeiten wir einen rekursiven Algorithmus zur Abarbeitung der Baumstruktur, z.B. unter Verwendung von Inordnung. Dafür könnte man einen (um eine Notationsform für Rekursion erweiterten) Programmablaufplan (Kontrollflussdiagramm) entwerfen, der die Struktur des verwendeten Datentyps `BinTree` wieder spiegelt.

**Abb. 8.3.** Struktur des Inorder-Algorithmus

Die Ablaufsicht des Algorithmus kann mit einem Aufrufbaum modelliert werden, der uns die kaskadenartige Rekursion verdeutlicht, die als Darstellung von Instanzen der Funktion *work* genau der Struktur des Stammbaums (als Darstellung von Instanzen des Type *BinTree*) entspricht (siehe Abb. 8.4).



**Abb. 8.4.** Aufrufbaum des Inorder-Algorithmus

## 8.4 Implementierung

**Funktionale Variante.** Wegen der kompakten Darstellungsmöglichkeiten und der Nähe zur logischen Struktur (siehe Abb. 8.2) wählen wir zunächst eine funktionale Sprache, etwa *gofer*. Wir realisieren hier nur die erste Aufgabenstellung aus Abschnitt 8.1, nämlich die Aufzählung aller Vorfahren von Don Carlos in verschiedenen Ordnungen (Inorder, Preorder, Postorder). Für den Inorder-Fall erhält man das folgende Programm:

```

{----- Datentypdeklaration -----}
data BinTree = Empty | Node (String,BinTree,BinTree)

{----- Diskriminatoren -----}
isEmpty::BinTree -> Bool
isEmpty Empty = True
isEmpty tree = False

{----- Selektoren -----}
mark::BinTree->String
mark (Node (nodemark,ltree, rtree)) = nodemark

leftTree::BinTree->BinTree
leftTree (Node (nodemark,ltree, rtree)) = ltree
  
```

```

rightTree::BinTree->BinTree
rightTree (Node (nodemark,ltree, rtree)) = rtree

{----- Baumtraversierung -----}
work::BinTree -> String
work tree = if isEmpty(tree) then "|"
else
work(leftTree(tree))++mark(tree)++work(rightTree(tree))

{----- Aufbau der Datenstruktur -----}
do = work(Node ("Don Carlos",
    Node ( "Koenig Philipp II",
        Node ( "Kaiser Karl V",
            Node ("Koenig Philipp I von Spanien",Empty,Empty),
            Node ("Johanna von Spanien",Empty,Empty)),
        Node ( "Isabella von Portugal",
            Node ("Koenig Emanuel I von Portugal",Empty,Empty),
            Node ("Maria von Spanien",Empty,Empty))),
    Node ("Maria von Portugal",
        Node ( "Koenig Johann III.von Portugal",
            Node ("Koenig Emanuel I von Portugal",Empty,Empty),
            Node ("Katharina von Oesterreich",
                Node ("Koenig Philipp I von Spanien",Empty,Empty),
                Node ("Johanna von Spanien",Empty,Empty)))))))
Portugal",Empty,Empty),
    Node ("Maria von Spanien",Empty,Empty))),
```

Die Ausführung sieht am Bildschirm dann so aus:

```
E:\Lerninh\Beispiele\Stammbaum>gofer
Gofer Version 2.28b Copyright (c) Mark P Jones 1991-1993
```

```
Reading script file "d:\progspr\pcgofer\standard.pre":
```

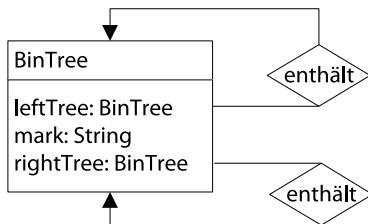
```
Gofer session for:
d:\progspr\pcgofer\standard.pre
Type ?: for help
?:l baum.gs
Reading script file "baum.gs":
```

```
Gofer session for:
d:\progspr\pcgofer\standard.pre
baum.gs
? do
|Koenig Philipp I von Spanien|Kaiser Karl V|Johanna von Spanien|Koenig
Philipp II|Koenig Emanuel I von Portugal|Isabella von Portugal|Maria von
Spanien|Don Carlos|Koenig Emanuel I von Portugal|Koenig Johann III.von
```

Portugal|Maria von Spanien|Maria von Portugal|Koenig Philipp I von Spanien|Katharina von Oesterreich|Johanna von Spanien|

(884 reductions, 2817 cells, 1 garbage collection)

**Imperative Variante.** Wagt man sich an die weiteren Aufgabenstellungen aus Abschnitt 8.1, so steht man vor dem Problem, dass das fehlende Zustandskonzept funktionaler Sprachen keine lokalen Variablen zulässt. Diese Schwierigkeiten kann man mit der Verwendung einer imperativen Sprache beheben. Zudem kann auch hier die objektorientierte Sichtweise zum besseren Verständnis beitragen, weshalb wir bevorzugt objektorientiert modellieren und (mit Java) implementieren. Wir entwerfen zunächst ein Klassendiagramm (siehe Abb. 8.5), in dem wir die doppelte Rekursion explizit sichtbar machen.



**Abb. 8.5.** Die Klasse BinTree

Auch die Implementierung der Variante *EmptyTree* (siehe Abb. 8.2) in Java sollte zur Sprache kommen: Wir behelfen uns hier mit einer „leeren“ Referenz auf ein nicht vorhandenes Objekt, so dass die Diskriminatorfunktion *tree.isEmptyTree()* nun mit dem Ausdruck *tree == null* gleichwertig ist.

Wir formulieren (zwecks Vergleich mit der funktionalen Programmvariante) zunächst die Funktion *work*, die eine Inorder-Abarbeitung am Bildschirm ausgibt. Dazu implementieren wir noch die Produktion je einer Liste der Knotenmarkierungen in Inordnung, Vorordnung und Nachordnung sowie einer Liste, die jedem Kind seine jeweiligen Eltern zuweist. Die Aufgaben 1 und 4 aus 8.1. sind damit direkt gelöst, die restlichen beiden reduzieren sich auf simple Operationen auf diesen Listen. Als Hilfsdatenstruktur verwenden wir eine verkettete Liste von Texten (Klasse *StringList*) mit den benötigten Methoden. Es folgen die Programmtexte:

### 1. Die Klasse *StringList*:

```

public class StringList {

    StringList next;
    String mark;

    public StringList(String m, StringList n) {
        mark = m;
        next = n;
    }
}
  
```

```
public void printList (){  
    StringList n = this;  
    while (n.next != null){  
        System.out.print(n.mark);  
        n = n.next;  
    };  
    System.out.println(n.mark);  
}  
  
public StringList getLast(){  
    StringList n = this;  
    while (n.next != null) {  
        n = n.next;  
    }  
    return n;  
}  
  
public void append(String m){  
    StringList lnew, llast;  
    lnew = new StringList (m, null);  
    llast = this.getLast();  
    llast.next = lnew;  
    lnew = new StringList (" ", null);  
    llast = this.getLast();  
    llast.next = lnew;  
}  
}
```

## 2. Die Klasse BinTree

```
public class BinTree {  
  
    public BinTree leftTree;  
    public String mark;  
    public BinTree rightTree;  
  
    public BinTree(BinTree ltree, String m, BinTree rtree){  
        leftTree = ltree;  
        mark = m;  
        rightTree = rtree;  
    }  
  
    public void work(){  
        if (leftTree != null) {leftTree.work();}  
        System.out.println(mark);  
        if (rightTree != null) {rightTree.work();}  
    }  
}
```

```

public void pre_order_List(StringList l){
l.append(mark);
if (leftTree != null) {leftTree.pre_order_List(l);}
if (rightTree != null) {rightTree.pre_order_List(l);}
}

public void in_order_List(StringList l){
if (leftTree != null) {leftTree.in_order_List(l);}
l.append(mark);
if (rightTree != null) {rightTree.in_order_List(l);}
}

public void post_order_List(StringList l){
if (leftTree != null) {leftTree.post_order_List(l);}
if (rightTree != null) {rightTree.post_order_List(l);}
l.append(mark);
}

public void parent_List(StringList l){
if (leftTree != null && rightTree != null) {
    l.append(mark);
    l.append (" hat Eltern: ");
    l.append (leftTree.mark);
    l.append (" und ");
    l.append (rightTree.mark);
    l.append (" \n ");
    leftTree.parent_List(l);
    rightTree.parent_List(l);
}
}
}
}

```

### 3. Die Klasse WorkTree mit dem Hauptprogramm

```

public class WorkTree {

    public WorkTree() {
    }

    public static void main(String[] args) {

        // Die Blätter des Baumes (z.T. mit redundanter Information)
        BinTree w11tree = new BinTree (null, "Philipp I", null);
        BinTree w12tree = new BinTree (null, "Johanna", null);
        BinTree w13tree = new BinTree (null, "Emanuel I", null);
        BinTree w14tree = new BinTree (null, "Maria v.S.", null);
        BinTree w15tree = new BinTree (null, "Emanuel I", null);
        BinTree w16tree = new BinTree (null, "Maria v.S.", null);
    }
}

```

```

BinTree w17tree = new BinTree (null, "Philipp I", null);
BinTree w18tree = new BinTree (null, "Johanna", null);

// Die Knoten der 2. Ebene (von unten)
BinTree w21tree = new BinTree (w11tree, "Karl V.",w12tree);
BinTree w22tree = new BinTree (w13tree, "Isabella", w14tree);
BinTree w23tree = new BinTree (w15tree, "Johann III", w16tree);
BinTree w24tree = new BinTree (w17tree, "Katharina", w18tree);

// Die Knoten der 3. Ebene (von unten)
BinTree w31tree = new BinTree (w21tree, "Philipp II", w22tree);
BinTree w32tree = new BinTree (w23tree, "Maria v.P.", w24tree);

// Die Wurzel
BinTree wroottree = new BinTree (w31tree, "Carlos", w32tree);

System.out.println("WORK:");
wroottree.work();

StringList prelist = new StringList("PREORDER: ", null);
wroottree.pre_order_List(prelist);
prelist.printList();

StringList inlist = new StringList("INORDER: ", null);
wroottree.in_order_List(inlist);
inlist.printList();

StringList postlist = new StringList("POSTORDER: ", null);
wroottree.post_order_List(postlist);
postlist.printList();

StringList parentlist = new StringList("ELTERNLISTE:\n ", null);
wroottree.parent_List(parentlist);
parentlist.printList();
}

}

```

Der Ablauf des Programms stellt sich am Bildschirm folgendermaßen dar:

```

C:\PROGSPR\JBUILDER\java\bin\javaw.exe -classpath
"C:\ProgSpr\JBUILDER\myclasses;C:\PROGSPR\JBUILDER\lib\swingall.jar;C:\PROGSPR\JBUILDER\lib\jbcl2.0-res.jar;C:\PROGSPR\JBUILDER\lib\jgl3.1.0.jar;C:\PROGSPR\JBUILDER\java\lib\classes.zip" BinTreeIJB.WorkTree
AppAccelerator(tm) 1.1.034 for Java (JDK 1.1), x86 version.
Copyright (c) 1998 Borland International. All Rights Reserved.

```

**WORK:**

Philipp I Karl V. Johanna Philipp II Emanuel I Isabella Maria v.S. Carlos Emanuel  
 I Johann III Maria v.S. Maria v.P. Philipp I Katharina Johanna

**PREORDER:** Carlos Philipp II Karl V. Philipp I Johanna Isabella Emanuel I  
 Maria v.S. Maria v.P. Johann III Emanuel I Maria v.S. Katharina Philipp I  
 Johanna

**INORDER:** Philipp I Karl V. Johanna Philipp II Emanuel I Isabella Maria v.S.  
 Carlos Emanuel I Johann III Maria v.S. Maria v.P. Philipp I Katharina Johanna

**POSTORDER:** Philipp I Johanna Karl V. Emanuel I Maria v.S. Isabella Philipp II  
 Emanuel I Maria v.S. Johann III Philipp I Johanna Katharina Maria v.P. Carlos

**ELTERNLISTE:**

Carlos hat Eltern: Philipp II und Maria v.P.  
 Philipp II hat Eltern: Karl V. und Isabella  
 Karl V. hat Eltern: Philipp I und Johanna  
 Isabella hat Eltern: Emanuel I und Maria v.S.  
 Maria v.P. hat Eltern: Johann III und Katharina  
 Johann III hat Eltern: Emanuel I und Maria v.S.  
 Katharina hat Eltern: Philipp I und Johanna

## 8.5 Wertung und Ausblick

In der Wertung kann man abschließend Vor- und Nachteile einer hierarchischen Datenorganisation besprechen. Der strukturellen Mehrinformation steht ein erhöhter Verwaltungsaufwand für die Datenstruktur gegenüber. Mit alternativen Darstellungsmethoden für vollständige Binäräbäume (z.B. mit Hilfe von Feldern) könnte man dies deutlich machen.

Auch der Vergleich der beiden Programmvarianten gibt Anlass zu Diskussionen. Wo liegen die Grenzen der funktionalen Variante? Was sind die Gründe für diese Einschränkungen?

Schließlich könnte man noch diskutieren, ob man zur Arbeitsersparnis gleichartige Blätter des Binärbaumes nur jeweils einmal realisieren sollte. Technisch gesehen zerstört dies die Baumstruktur (da es dann jeweils zwei Referenzen auf jedes dieser Blätter gibt), der logischen Struktur tut es jedoch keinen Abbruch. Dennoch wäre dies ein Anlass, über die per Zuweisung erzeugte „Gleichheit“ von Objekten (im Vergleich zu der von einfachen Variablenwerten) zu reden.

# 9 Formale Sprachen

Dieses Oberstufenprojekt führt die Schülerinnen und Schüler erstmals auf eine systematische Beschäftigung mit der Struktur formaler Sprachen. Im Laufe des Informatikunterrichtes der vergangenen Jahre haben sie bereits eingesehen, dass es notwendig ist, die bei der Interaktion mit Rechnern verwendeten Sprachen exakt zu definieren. Dies wurde am Beispiel von Programmiersprachen ebenso deutlich wie bei der Manipulation der Objekte typischer Informatiksysteme.

Nun begegnen wir am Beispiel der Struktur einer E-Mail-Adresse erstmals den Begriffen „Syntax“ und „Semantik“ und ihrer Bedeutung. Außerdem lernen wir verschiedene Verfahren kennen, mit denen die Syntax einer formalen Sprache festgelegt und Ausdrücke der Sprache auf syntaktische Korrektheit geprüft werden können.

Eine besondere Bedeutung gewinnt bei diesem Projekt die Veranschaulichung: Um nicht in einen Kurzkurs in Theoretischer Informatik abzugleiten, stehen immer die Anwendung und der Nutzen der vermittelten Konzepte im Mittelpunkt des Interesses. Mathematische Formulierungen sollten daher soweit möglich vermieden, Implementierungen jeder Art dagegen ausgiebig betrieben werden.

## 9.1 Aufgabenstellung und Lernziele

Bereits vor langer Zeit, in der 6. Jahrgangsstufe, haben die Schüler elektronische Post kennen gelernt und inzwischen in anderen Unterrichtsfächern und privat ausgiebig benutzt. Nun wollen wir ein System entwerfen, das einerseits die syntaktische Korrektheit der Adressangaben von E-Mail-Nachrichten prüft und andererseits die relevanten, sinntragenden Einheiten (hier lokale Adresse und DNS-Name des Rechners) zur weiteren Verarbeitung bereitstellt.

Anstatt nun eine der Notationen für formale Sprachen deduktiv einzuführen, gehen wir exemplarisch vor und betrachten eine vorgegebene Definition, anhand derer wir sowohl die Beschreibungsmittel, in diesem Fall die der Backus-Naur-Form (BNF), als auch (unter deren Verwendung) die Syntax der betrachteten Sprache einführen. Im Sinne des konstruktivistischen Ansatzes der *Cognitive Apprenticeship* (siehe z.B. Reinmann-Rothmeier u. Mandl (1996)) ist dafür eine tatsächlich für die Praxis relevante Definition besser geeignet als eine „Spieldefinition“. Auf der anderen Seite wird erstere oft naturgemäß sehr detailliert und daher relativ schwer verständlich und zu aufwendig für unser Vorhaben sein. Wir schlagen deshalb einen Mittelweg ein, indem wir zwar mit der Betrachtung einer „echten“ Definition zur Einführung der BNF-Notation

beginnen, diese allerdings vor der weiteren Verarbeitung soweit nötig und vertretbar vereinfachen.

Für die Definition des Adressformates der elektronischen Post im Internet gilt der *Request for Command* Nummer 822 (kurz RFC 822). Allerdings wird darin eine relativ schwer verständliche Syntaxnotation verwendet, weshalb wir für den Einstieg die „saubere“ BNF-Adressdefinition aus RFC 821 vorziehen, obwohl dieser eigentlich nur für die Beschreibung des Übertragungsprotokolls *Simple Mail Transfer Protocol* (SMTP) zuständig ist (siehe Tanenbaum (1996)).

Wir besorgen uns also die beiden genannten RFC's aus dem Internet (z.B. unter der URL <http://www.rfc-editor.org/rfc/rfc821.txt> bzw. [.../rfc822.txt](http://www.rfc-editor.org/rfc/rfc822.txt)) und betrachten zunächst die Vorgaben in RFC 821 auf S. 29ff. Am Beispiel einer konkreten E-Mail-Adresse besprechen wir einerseits kurz die Bedeutung der einzelnen Sprachmittel (Definition „::=“, Alternative „|“ und Sequenz), andererseits die Struktur der durch die obigen BNF-Regeln vorgegebenen Sprache, wobei insbesondere die Rekursionen einer näheren Betrachtung wert sind. Die Schüler erkennen dabei die Analogie zu rekursiven Datenstrukturen aus dem vorigen Kapitel.

Bei der Arbeit mit RFC 821 fällt uns auf, dass hier nur Domänenbezeichner zugelassen werden, die mindestens aus drei Zeichen bestehen. Wie die zulässige Adresse *Peter.Hubwieser@in.tum.de* zeigt, kann diese Einschränkung i.a. nicht (mehr) gültig sein. Wir suchen also in RFC 822 nach Abweichungen und werden auf S.26 und S. 43ff fündig.

Anschließend reduzieren die offizielle Definition auf die folgende stark vereinfachte Form und verbessern dabei auch die Aussagekraft einiger Bezeichner durch Umbenennung.

```

<mailbox> ::= <dot-string> "@" <dot-string>
<dot-string> ::= <string> | <string> "." <dot-string>
<string> ::= <char> | <char> <string>
<special> ::= "<" | ">" | "(" | ")" | "[" | "]" | "\\" | "/" | "." | "," | ";" | ":" | "@" | """" | the
control characters (ASCII codes 0 through 31 inclusive and 127)

<char> ::= any one of the 128 ASCII characters, but not any <special> or the
space character (ASCII code 32)

```

Am Ende dieser ersten Phase vereinbaren wir als Zielsetzung den Entwurf und die Implementierung eines Erkennungsprogramms, das die am Beginn dieses Abschnittes beschriebenen Leistungen erbringt.

Am Ende der Unterrichtssequenz sollten die Schüler in der Lage sein:

- die Notwendigkeit der Formalisierung von Sprachen einzusehen,
- BNF, reguläre Ausdrücke und Syntaxdiagramme zur Definition künstlicher Sprachen einzusetzen und zu interpretieren,
- Terminale und Nichtterminale (syntaktische Variablen) zu unterscheiden,
- mit Hilfe von BNF-Regeln, regulären Ausdrücken und Syntaxdiagrammen die Zugehörigkeit eines Ausdrucks zur beschriebenen Sprache zu entscheiden,

- zu einfachen regulären Sprachen auf intuitivem Wege erkennende Automaten zu konstruieren und diese Automaten in einer geeigneten Programmiersprache zu implementieren.

## 9.2 Beschreibung formaler Sprachen

Zur Veranschaulichung der Syntax formaler Sprachen auf schulischem Niveau eignen sich wohl am besten Syntaxdiagramme (siehe z.B. Wirth (1986)). Wir übersetzen also die (reduzierte) „offizielle“ Syntax in entsprechende Diagramme mit rechteckigen Knoten für Nichtterminale und runden für Terminate (siehe Abb. 9.1). Auf die Darstellung `<special>` verzichten wir dabei ganz, `<char>` stellen wir nur angedeutet dar.

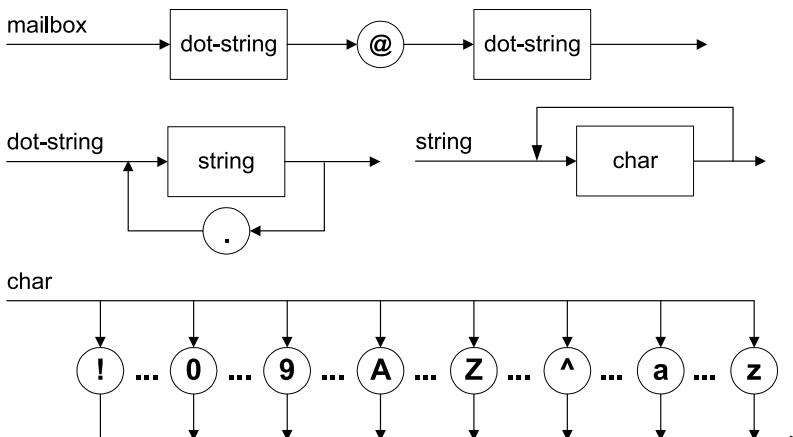


Abb. 9.1. Syntaxdiagramme für E-Mail-Adressen

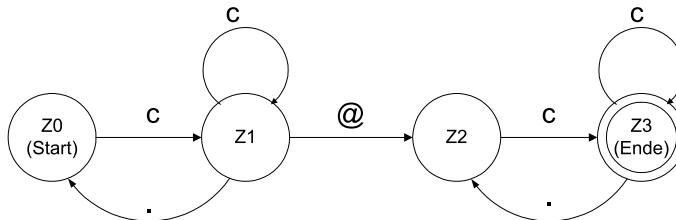
Bevor wir uns dem Problem der Syntaxüberprüfung zuwenden, wenden wir noch eine dritte Art der Beschreibung (regulärer) formaler Sprachen an, nämlich reguläre Ausdrücke. Dazu übernehmen wir von der BNF-Notation die Schreibweisen für Alternativen ("|") und Sequenzen und fügen als neue Sprachmittel die Möglichkeit der Wiederholung eckig geklammter Teilausdrücke ("\*" für 0 .. n bzw. "+" für 1 .. n Wiederholungen) hinzu. Die Übersetzung unserer Syntaxdiagramme in einen regulären Ausdruck beginnen wir bei der Regel für „mailbox“ und ersetzen dann sukzessive alle Nichtterminale durch die entsprechenden Ausdrücke. Schließlich tauschen wir noch `<char>` durch das (als *Variable* mit Belegung durch Terminate aus einer speziellen Zeichenmenge) aufzufassenden Symbol c. Damit erhalten wir den folgenden sehr kompakten Ausdruck:

$$c^+ [c^+]^* @ c^+ [c^+]^* \quad \text{mit} \quad c = ["!" | ... | "z" | "\sim"]$$

### 9.3 Erkennung formaler Sprachen durch Automaten

Zwar haben die Schüler in der Mittelstufe bereits Zustands-Übergangsdiagramme kennen gelernt, jedoch wurden diese dort vor allem zur Modellierung von Abläufen eingesetzt. Daher stellt die Anwendung von endlichen Automaten zur Prüfung der Struktur von Zeichenketten an dieser Stelle eine neue Erfahrung dar, weshalb man an dieser Stelle sehr behutsam vorgehen und sich auf deterministische endliche Automaten (DEA) ohne  $\epsilon$ -Übergänge beschränken sollte. Das übliche Verfahren zur Konstruktion eines akzeptierenden Automaten zu einem regulären Ausdruck (siehe etwa Hopcroft u. Ullman (1994), Broy (1997) oder Schöning (1995)) scheidet damit aus (da damit nichtdeterministische Automaten mit  $\epsilon$ -Übergängen produziert werden). U.U. wäre im Leistungskurs eine spätere Behandlung nichtdeterministischer Automaten als Vertiefung angebracht. Wir konstruieren also hier intuitiv einen DEA für die oben beschriebene reguläre Sprache (siehe Abb. 9.2). Dabei ist vorab zu klären, wie die Abarbeitung erfolgt und wann eine Adresse durch diesen Automaten als akzeptiert zu betrachten ist:

1. Die an den Übergängen notierten Zeichen sind im Sinne einer die Transition auslösenden Aktion „als nächstes Zeichen wird c eingegeben“ zu verstehen.
2. Wird in einem der Zustände ein Zeichen eingegeben, das an keiner von diesem Zustand ausgehenden Transition notiert ist, so wird die Bearbeitung als „nicht akzeptiert“ abgebrochen.
3. Die gesamte eingegebene Zeichenkette gilt als akzeptiert, wenn sich der Automat nach ihrem letzten Zeichen in einem Endzustand befindet.



**Abb.9.2.** Akzeptierender endlicher Automat für unsere reguläre Sprache.

Die zweite Regel könnte auch durch einen „Fehlerzustand“ symbolisiert werden zu dem von jedem anderen Zustand aus eine Kante mit der Markierung „alle anderen Zeichen“ führt.

### 9.4 Implementierung

Jetzt ist es an der Zeit, ein lauffähiges Simulationsprogramm für unseren Automaten zu erstellen. Aus logischer Sicht stellt das die Schüler vor keine größeren Probleme, da sie in der Mittelstufe bereits zahlreiche Zustands-

Übergangsdiagramme simuliert haben. Wir wiederholen die dort (siehe Kapitel 4) gewonnenen Erkenntnisse und wenden sie auf unseren Akzeptor an, wobei wir wegen der bequemen Interaktionsmöglichkeiten eine MS-Excel®-Tabelle mit den benötigten Visual-Basic®-Funktionen hinterlegen. Der verwendete Quellcode lautet:

1. Akzeptor-Funktion zur Überprüfung der Syntax:

```
Public Function Akzeptor(Adresse As String) As Boolean
    State = "z0"
    i = 0
    Do While i < Len(Adresse) And State <> "error"
        element = Mid(Adresse, i + 1, 1)
        elementType = CharType(element)
        Select Case State
            Case "z0"
                Select Case elementType
                    Case "char"
                        State = "z1"
                    Case Else
                        State = "error"
                End Select
            Case "z1"
                Select Case elementType
                    Case "char"
                        State = "z1"
                    Case "point"
                        State = "z0"
                    Case "at"
                        State = "z2"
                    Case Else
                        State = "error"
                End Select
            Case "z2"
                Select Case elementType
                    Case "char"
                        State = "z3"
                    Case Else
                        State = "error"
                End Select
            Case "z3"
                Select Case elementType
                    Case "char"
                        State = "z3"
                    Case "point"
                        State = "z2"
                    Case Else
                        State = "error"
                End Select
        End Select
    Loop
    If State = "error" Then
        Akzeptor = False
    Else
        Akzeptor = True
    End If
End Function
```

```

        End Select
    End Select
    i = i + 1
Loop
If State = "z3" Then
    Akzeptor = True
    Else
        Akzeptor = False
    End If
End Function

```

2. Die Funktion CharType zur Überprüfung der für <char> zulässigen Zeichen

```

Public Function CharType(ByVal z As String) As String
Select Case Asc(z)
Case 33, 35 To 39, 42 To 45, 65 To 90, 94 To 122, 126
    CharType = "char"
Case 46
    CharType = "point"
Case "64"
    CharType = "at"
Case Else
    CharType = "special"
End Select
End Function

```

3. Die Funktionen LocalPart und Domain zur Aufgliederung der Adresse

```

Public Function LocalPart(Adresse As String) As String
If Akzeptor(Adresse) Then
    LocalPart = Left(Adresse, InStr(Adresse, "@") - 1)
Else
    LocalPart = "Fehler!"
End If
End Function

```

```

Public Function Domain(Adresse As String) As String
If Akzeptor(Adresse) Then
    Domain = Right(Adresse, Len(Adresse) - InStr(Adresse, "@"))
Else
    Domain = "Fehler!"
End If
End Function

```

The screenshot shows a Microsoft Excel spreadsheet window titled "Email-Adressen.xls". The spreadsheet contains two columns, A and B. Row 1 contains the text "Projekt E-Mail-Adressen 12/2". Row 2 contains "P. Hubwieser". Row 3 is empty. Row 4 contains the entry "4 Adresse: Peter.Hubwieser@in.tum.de", where "Peter.Hubwieser@in.tum.de" is highlighted with a red border. Row 5 is empty. Row 6 contains the entry "6 Akzeptiert: WAHR". Row 7 is empty. Row 8 contains the entry "8 Lokaler Teil: Peter.Hubwieser". Row 9 contains the entry "9 Domäne: in.tum.de". Row 10 is empty. At the bottom of the spreadsheet, there are navigation buttons for "Tabelle1" and "Tabelle2", along with other standard Excel toolbar icons.

	A	B
1	Projekt E-Mail-Adressen 12/2	
2	P. Hubwieser	
3		
4	Adresse:	Peter.Hubwieser@in.tum.de
5		
6	Akzeptiert:	WAHR
7		
8	Lokaler Teil:	Peter.Hubwieser
9	Domäne:	in.tum.de
10		

Abb. 9.3. Ablauf der Syntaxüberprüfung am Bildschirm

Abschließend führen wir natürlich auch in diesem Projekt eine ausgiebige Reflexionsphase durch, auf deren Darstellung hier jedoch verzichtet werden soll.

# 10 Rechnerkommunikation

Nachdem die Schülerinnen und Schüler im Laufe des vorausgegangenen Informatikunterrichtes ebenso wie im Zusammenhang mit dem Medieneinsatz von Informatiksystemen in anderen Fächern intensiven Kontakt mit den Diensten globaler und lokaler Rechnernetze hatten, ist es nun an der Zeit, die dabei beobachteten Phänomene systematisch zu untersuchen und zu erklären. Insbesondere zwei Fragestellungen von hoher allgemein bildender Bedeutung sind zu klären:

- (1) Wie kommunizieren zwei Rechner miteinander?
- (2) Wie geht man mit der Nebenläufigkeit der an der Rechnerkommunikation beteiligten Prozesse um?

Während die Antworten auf die erste Fragestellung vor allem im Hinblick auf Verständnis und Beurteilung verteilter Informatiksysteme relevant sind, haben Erkenntnisse jene zur zweiten darüber hinaus auch eine große Bedeutung in Bezug auf allgemeine nebenläufige Prozesse, wie sie in unserer immer mehr auf Parallelisierung ausgerichteten Arbeitswelt sehr häufig zu finden sind (siehe auch Brauer (1989)). Man denke dabei nur an die Synchronisation der Arbeit parallel aktiver Mitarbeiterteams innerhalb größerer Projekte oder Probleme der Koordination von Robotern an einem Fließband. Ähnlich wie man die Schüler nicht aus dem gymnasialen Physikunterricht entlassen sollte, ohne ihnen über das deterministische Weltbild hinausgehend zumindest eine Ahnung von statistischen Phänomenen vermittelt zu haben, darf sich der Informatikunterricht nicht auf sequentielle Datenverarbeitung beschränken.

## 10.1 Aufgabenstellung

Da wir uns im vorausgegangenen Kapitel mit dem Adressformat elektronischer Post beschäftigt haben, liegt es nahe, ein kleines Simulationssystem für den Transport elektronischer Nachrichten zu entwerfen und zu implementieren. Dazu beginnen wir mit der Modellierung einer einfachen (E-Mail-)Erzeuger-Verbraucher-Situation mit Hilfe von Petrinetzen und bauen diese Situation stufenweise zu einem Kommunikationsnetz aus, in dem mehrere Sender mit mehreren Empfängern über einen gemeinsamen Server mit beschränktem Nachrichtenpuffer kommunizieren.

Die Implementierung stellen wir in *Java* vor, da diese Sprache einerseits ein sehr einfaches Konzept für nebenläufige Prozesse anbietet, andererseits über komfortable Mechanismen zur Kommunikation über Netzwerke (Sockets) verfügt, so dass eine (hier nicht gezeigte) Verteilung des Systems auf mehrere Rechner

durchaus im Bereich des Möglichen liegt. Anleitungen dazu findet man z.B. in Niemeyer u. Peck (1997).

Nach der Arbeit diesem Projekt sollen die Schüler in der Lage sein,

- einfache Protokolle zur Kommunikation von Rechnern zu verstehen und zu entwerfen,
- die grundsätzlichen Aspekte solcher Protokolle zu beschreiben,
- die prinzipiellen Unterschiede in der Funktionsweise von Zustellungs- bzw. Abholprotokollen wiederzugeben,
- einfache nebenläufige Prozesse mit Hilfe von Petrinetzen zu modellieren und zu implementieren,
- die wichtigsten Konzepte zur Synchronisation des Zugriffs paralleler Prozesse auf gemeinsame Betriebsmittel (Semaphore- und Monitorkonzept) in typischen Situationen (z.B. Erzeuger-Verbraucher) zu verstehen und zu erklären,
- einfache Verklemmungssituationen zu erkennen und aufzulösen.

Die folgende Unterrichtsskizze beschreibt allerdings nicht alle zur Erreichung dieser Lernziele notwendigen Unterrichtsaktionen. Sie soll lediglich anhand eines konkreten Beispiels eine schülergemäße Behandlung des Themenkomplexes andeuten.

## 10.2 Postprotokolle

Zur Klärung der Frage, wie sich Rechner überhaupt miteinander verständigen, versetzen wir uns zunächst in die Lage eines Rechners, der für einen seiner Benutzer elektronische Post über einen Mailserver absenden will. Dazu öffnen wir in einer möglichst originären Umgebung (im Sinne von *Cognitive Apprenticeship*, siehe Reinmann-Rothmeier u. Mandl (1996), Dubs (1995)) eine *telnet*-Verbindung mit einem Mailserver (auf Port 25) und führen „von Hand“ eine SMTP-Sitzung durch. Der Ablauf gestaltet sich folgendermaßen:

c:> telnet outof.bnro.de 25

*Eingabe:*

*Antwort von outof.bnro.de:*

220- OutOf.Rosenheim.Baynet.De Sendmail  
50413.SGI.8.6.12/950213.SGI.AUTOCF read y at  
Fri, 19 Jun 1999 10:19:10 +0100  
220 ESMTSP spoken here

he1o OutOf

250 OutOf.Rosenheim.Baynet.De Hello line-41  
[194.95.220.185], pleased to meet you

mail from: hub-  
wiese@bnro.de

250 hubwiese@bnro.de ... Sender ok

rcpt to: meier@in.tum.de

250 meier@in.tum.de ... Recipient ok

Data

354 Enter mail, end with "." on a line by itself

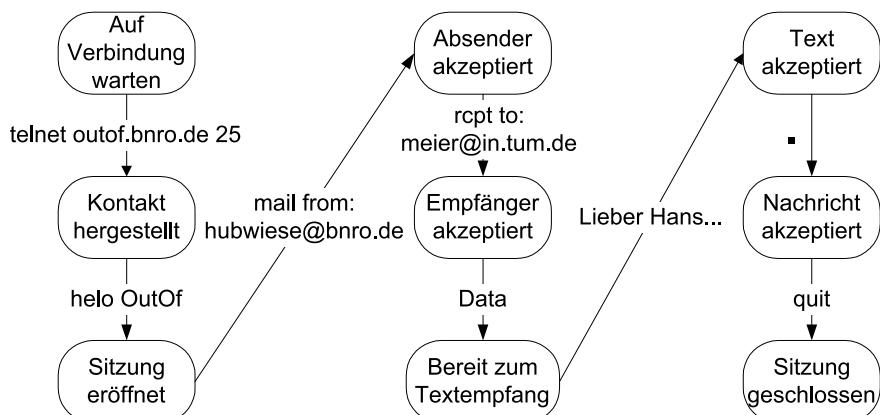
Lieber Hans,  
bitte ruf mich bald mal  
an!

250 KAA13937 Message accepted for delivery

quit

Am besten geben wir den Schülern die (wenigen) dazu benötigten SMTP-Kommandos auf einem Arbeitsblatt in die Hand. Diese technischen Einzelheiten dienen natürlich ausschließlich der Veranschaulichung und der Motivierung und sind *keinesfalls* Lernstoff! Nach Abschluss der (etwas mühsamen) Unterhaltung auf Rechnerebene stellen wir fest:

- Die Kommunikation verläuft über eine Folge von genau festgelegten Kommandos, die auf den beteiligten Rechnern jeweils einen Zustandsübergang auslösen (in vereinfachter Form siehe Abb. 10.1).
- Es existiert ein (natürlich mittlerweile allgemein bekanntes) Sicherheitsloch: Offensichtlich kann jeder, der eine solche Sitzung eröffnen kann, mittels dieses Protokolls eine Nachricht unter der angegebenen Absenderadresse absetzen.



**Abb. 10.1.** Zustände des Servers während unserer SMTP-Sitzung

Nachdem wir den Sendeprozess an einem Beispiel so genau untersucht haben, werfen wir noch einen kurzen Blick auf den Empfangsmechanismus, genauer gesagt auf die Abholung elektronischer Nachrichten vom Mailserver, wobei im Gegensatz zum Versand das Protokoll POP3 (Post Office Protokoll) „gesprochen“ wird (der zuständige Rechner ist jetzt der Mailserver des Empfängers, der verantwortliche Port 110):

c:>telnet sunbroy49.informatik.tu-muenchen.de 110

<i>Eingabe</i>	<i>Antwort von sunbroy49</i>
	+OK QUALCOMM Pop server derived from UCB (version 2.1.4-R3) at sunbroy49 starting.
user meier	+OK Password required for meier.
pass	
meinpasswort	+OK meier has 2 message(s) (2151 octets).
retr 1	+OK 1073 octets Received: from OutOf.bnro.de ([194.95.220.131] HELO OutOf.bnro.de ident: NO-IDENT-SERVICE [port 29361]) by tuminfo2.informatik.tu-muench en.de with SMTP id <110968-223>; Fri, 19 Jun 1998 11:20:36 +0000 Received: from outof (line-41 [194.95.220.185]) by OutOf.bnro.De (95 0413.SGI.8.6.12/950213.SGI.AUTOCF) via SMTP id KAA13919 for meier@in.tum.de; Fri, 19 Jun 1998 10:17:39 +0100 Date: Fri, 19 Jun 1998 10:17:39 +0100 From: hubwiese@bnro.de Message-Id: <199806190917.KAA13919@OutOf.bnro.de> To: meier@in.tum.de Lieber Hans, ...  .
quit	

Diesmal untersuchen wir den Vorgang nur noch auf Unterschiede zum Sendevorgang. Dabei stellt sich vor allem heraus, dass hier darauf zu achten ist, ob überhaupt eine Nachricht zum Abholen vorhanden ist. Das führt uns unmittelbar auf ein Erzeuger-Verbraucher-Problem, das wir uns folgenden Abschnitt vornehmen wollen.

### 10.3 Erzeuger, Verbraucher und Petrinetze

Als erstes System wollen wir eine einfache Erzeuger-Verbraucher-Situation, bezogen auf elektronische Nachrichten, modellieren und mit Hilfe eines Java-Programms simulieren. Zunächst überlegen wir uns dazu informell den Ablauf der einzelnen Arbeitszyklen, dann führen wir *Petri-Netze* ein, insbesondere die Regeln zum Feuern der Transitionen und des damit verbundenen Markenspiels, wie sie in Walter (1995) dargestellt sind, wobei wir uns zunächst auf Stellen mit der Kapazität 1 beschränken. Da wir diese Kapazität zur Modellierung eines Puffers später

erhöhen wollen, verwenden wir von Anfang an die Bezeichnung „Stelle“ und „Transition“ anstatt „Bedingung“ und „Ereignis“ (siehe Reisig 1985). Unser Erzeuger-Verbraucher-Problem (mit einem Puffer im Erzeugersystem) könnte dann in etwa so aussehen, wie in Abb. 10.2. gezeigt.

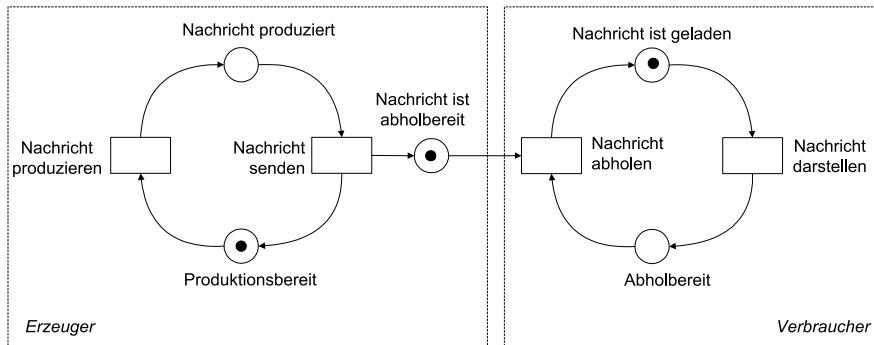


Abb. 10.2. Petrinetzdarstellung des Erzeuger-Verbraucher-Problems

Bevor wir die Implementierung dieses System angehen, müssen wir noch die Problematik des gleichzeitigen Zugriffs paralleler Prozesse auf kritische Ressourcen besprechen. In unserem Fall müssen Erzeuger und Verbraucher auf den gemeinsamen Ablageplatz der Nachricht zugreifen. Was geschieht, wenn der Verbraucher beginnt, die Nachricht abzuholen, bevor sie vollständig abgelegt ist? Wir werden also dafür sorgen müssen, dass beide Prozesse auf der gemeinsamen Ablage gewissermaßen „in Ruhe arbeiten“ können, dass also der Produzent erst störungsfrei seine Schreibarbeit vollenden kann, bevor der Empfänger mit der Abholung beginnt. Dazu gibt es das Hilfsmittel der Zusicherung *exklusiven Zugriffs* auf bestimmte Programm- und Datenbereiche (in Java intern durch ein Monitorkonzept realisiert).

Nun implementieren wir das System aus Abb. 10.2, wobei wir unser Augenmerk besonders auf die Aspekte der Nebenläufigkeit richten. Nach einigen ersten Abläufen ergänzen wir unser System um zwei weitere Konsumenten (mit unterschiedlicher Verbrauchsrate), was im Petrinetz keinen großen Aufwand verursacht: Wir schließen an der Stelle „Nachricht ist abholbereit“ einfach zwei weitere Verbraucher an, s.a. Reisig (1985). Dann erweitern wir die Kapazität der genannten Stelle, z.B. auf 5 Marken und erhalten somit einen „echten“ Puffer für unsere Nachrichten. Die folgende Implementierung dieser Ausbaustufe Systems orientiert sich stark an der Vorlage von Niemeyer u. Peck (1997), die als Inhalt der Nachricht einfach einen Zeitstempel und für den Puffer ein Objekt der Klasse *Vector* (heterogene Liste) verwenden. Wir konzentrieren uns bei der Implementierung auf den Übergabevorgang und lassen die eigentliche Produktion und die Darstellung der Nachrichten (siehe Abb. 10.2.) bis auf weiteres einfach weg. Die Mechanismen zur Behandlung von Ausnahmesituationen betrachten wir zudem ohne großes Aufheben als „black-box“-Systeme.

## 1. Die Klasse der Erzeuger

```
package Kommunikation;

import java.util.Vector;

class Producer extends Thread {
    static final int MAXQUEUE = 5;
    private Vector messages = new Vector();

    public void run() {
        try {
            while ( true ) {
                putMessage();
                sleep( 1000 );
            }
        } catch( InterruptedException e ) {}
    }

    private synchronized void putMessage()
        throws InterruptedException {

        while ( messages.size() == MAXQUEUE )
            wait();
        messages.addElement( new java.util.Date().toString() );
        notifyAll();
    }

    // wird vom Konsumenten aufgerufen
    public synchronized String getMessage()
        throws InterruptedException {
        notifyAll();
        while ( messages.size() == 0 )
            wait();
        String message = (String)messages.firstElement(); // Datenkonvertierung
        messages.removeElement( message );
        return message;
    }
}
```

## 2. Die Klasse der Verbraucher

```
class Consumer extends Thread {
    Producer producer;
    String name;
    int time;
```

```
Consumer(String n, Producer p, int t) {  
    this.producer = p;  
    this.name = n;  
    this.time = t;  
}  
  
public void run() {  
    try {  
        while ( true ) {  
            String message = producer.getMessage();  
            System.out.println(name+" got message: " + message);  
            sleep( time );  
        }  
    }  
    catch( InterruptedException e ) {}  
}
```

### 3. Die Hauptklasse zur Steuerung des Gesamtsystems

```
public class Communication {  
  
    public static Producer producer = new Producer();  
    public static Consumer cons1 = new Consumer( "Cons1", producer, 1000 );  
    public static Consumer cons2 = new Consumer( "Cons2", producer, 2000 );  
    public static Consumer cons3 = new Consumer( "Cons3", producer, 3000 );  
  
    public static void main(String args[]) {  
        producer.start();  
        cons1.start();  
        cons2.start();  
        cons3.start();  
    }  
}
```

Wie schon anlässlich der Parallelisierung unserer Zugsteuerung in Kapitel 7 beobachtet, verläuft das Zusammenspiel unserer nebenläufigen Prozesse wiederum nichtdeterministisch. Zur Demonstration folgen zwei aufeinander folgende Abläufe.

#### 1. Ablauf

```
Cons1 got message: Fri Mar 24 15:54:48 CET 2000  
Cons2 got message: Fri Mar 24 15:54:49 CET 2000  
Cons3 got message: Fri Mar 24 15:54:50 CET 2000  
Cons1 got message: Fri Mar 24 15:54:51 CET 2000  
Cons2 got message: Fri Mar 24 15:54:52 CET 2000  
Cons3 got message: Fri Mar 24 15:54:53 CET 2000
```

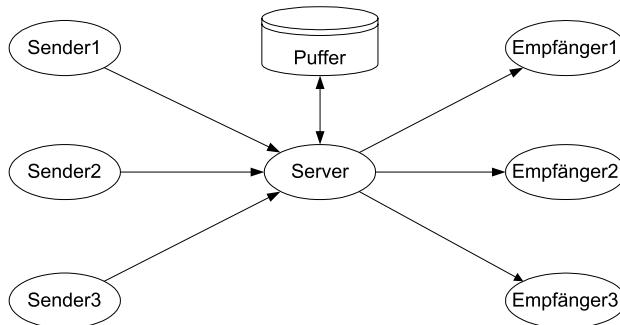
```
Cons2 got message: Fri Mar 24 15:54:54 CET 2000
Cons1 got message: Fri Mar 24 15:54:55 CET 2000
Cons3 got message: Fri Mar 24 15:54:56 CET 2000
Cons2 got message: Fri Mar 24 15:54:57 CET 2000
```

## 2. Ablauf

```
Cons1 got message: Fri Mar 24 15:56:10 CET 2000
Cons3 got message: Fri Mar 24 15:56:11 CET 2000
Cons2 got message: Fri Mar 24 15:56:12 CET 2000
Cons1 got message: Fri Mar 24 15:56:13 CET 2000
Cons2 got message: Fri Mar 24 15:56:14 CET 2000
Cons1 got message: Fri Mar 24 15:56:15 CET 2000
Cons3 got message: Fri Mar 24 15:56:16 CET 2000
Cons1 got message: Fri Mar 24 15:56:17 CET 2000
Cons2 got message: Fri Mar 24 15:56:18 CET 2000
Cons1 got message: Fri Mar 24 15:56:19 CET 2000
```

## 10.4 Simulation elektronischer Post

Nach diesen Vorüberlegungen an vereinfachten Systemen wagen wir uns an die Simulation unseres Postsystems. Dazu modellieren wir zunächst die Datenflüsse in unserem System (siehe Abb. 10.3). Wir planen drei Absender und drei Empfänger ein, von denen immer nur insgesamt einer mit dem Server verbunden sein darf. Letzterer regelt wiederum in exklusivem Zugriff den Datenverkehr mit dem Nachrichtenpuffer.



**Abb. 10.3.** Datenflüsse in unserem geplanten E-Mail-System

### 10.4.1 Das Protokoll

Die Sender sollen ihre Nachrichten an die unterschiedlichen Empfänger adressieren und sich als Absender eintragen können. Wir müssen uns also auf ein Daten-

format für die Nachrichten einigen. Die Schüler erkennen dabei, dass ein Protokoll immer mindestens zwei Arten von Vorschriften enthalten muss:

- (1) Regeln für den Ablauf der Datenübertragung und
- (2) Regeln für das Format der ausgetauschten Nachrichten.

Analog zu den Vorbildern aus dem Internet wollen wir für den Versand zum Server bzw. für das Abholen vom Server zwei verschiedene (Spiel-)Protokolle entwerfen: *Send Mail Protocol* (SMP) bzw. *Get Mail Protocol* (GMP). Beide verwenden dasselbe Datenformat, enthalten jedoch verschiedene Ablaufspezifikationen.

Falls die Zeit vorhanden ist, würde es den Schülern an dieser Stelle sicher viel Spaß machen, die Regeln der Protokolle nach dem Vorbild eines *Request for Comment* (RFC, siehe Kapitel 8) auszudiskutieren. Dabei gehört vor allem die Einsicht, dass diese Vereinbarungen bis ins letzte Detail vollständig und eindeutig sein müssen, zu den angestrebten Lernzielen.

### **Datenformat**

Die Nachrichten beider Protokolle sollen aus den folgenden vier Komponenten bestehen:

- *sender*: Absender, z.B. „SENDER1“ (7 Zeichen),
- *receiver*: Empfänger, z.B. „RECEIV1“ (7 Zeichen),
- *date*: Zeitstempel, z.B. „Fri Mar 24 15:56:10 CET 2000“ (30 Zeichen),
- *text*: Nachricht, z.B. „Lieber Hans, ...“

### **Ablauf**

**SMP:** Nach dem Datenformat regeln wir den Ablauf der Kommunikationssitzungen, zunächst für den Versand zum Server:

- (1) Sobald der Sender die synchronisierte Empfangsmethode des Servers (*login*) aufrufen darf, übergibt er mit diesem Aufruf seinen Namen (z.B. „SENDER1“) und erhält vom Server als den Rückgabewert dieser Methode den Text „WELCOME“.
- (2) Dann ruft er die Methode *putMail* mit dem Empfänger und dem Text der Nachricht als Parameterwerte auf, was vom Server mit „ACCEPTED“ quittiert wird, falls auf dem Puffer noch Platz für die Nachricht ist, ansonsten mit der Meldung „FAILED“.
- (3) Schließlich bricht der Sender die Verbindung mit dem Aufruf von *close* wieder ab.
- (4) Falls der Versand fehlgeschlagen, versucht der Sender erneut, eine Verbindung zum Server aufzubauen.

**GMP:** Die Sitzung soll in den Punkten (1), (3), (4) analog zum SMP verlaufen. Der verbliebene Schritt (2) muss natürlich geändert werden:

- (2) Der Empfänger ruft die Methode *hasMail* ohne Parameter auf und erhält als Rückgabewert *true*, falls (mindestens) eine Nachricht für ihn im Puffer vorliegt, ansonsten *false*. Im ersten Fall ruft er, wiederum ohne Parameter, die Methode *getMail* auf und erhält als deren Rückgabewert die (oberste) für ihn bestimmte Nachricht, die anschließend im Puffer gelöscht wird.

Für die Modellierung dieser Abläufe mittels Petrinetzen würden wir wegen der unterschiedlichen Empfängeradressen individuelle Marken benötigen. Da diese dritte Variante derselben Modellierungstechnik innerhalb *eines* Projektes die Schüler überfordern könnte, verzichten wir hier auf den Einsatz von Petrinetzen und modellieren die Abläufe mit Interaktionsdiagrammen (siehe Abb. 10.4). Dabei beschränken wir uns auf je einen Sender bzw. Empfänger.

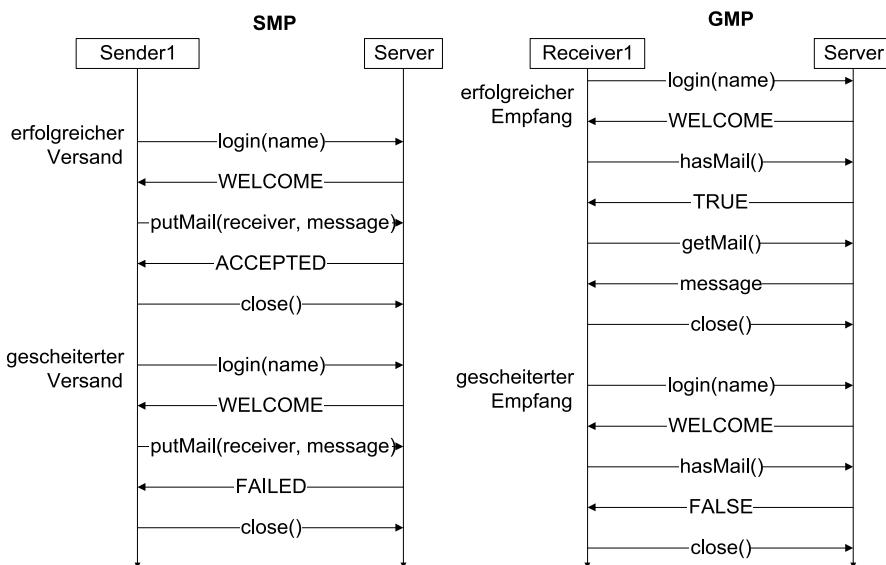


Abb. 10.4. Interaktionsdiagramme zu GMP und SMP

### 10.4.2 Die Implementierung

Abschließend werfen wir einen Blick auf die Implementierung, wobei wir als Text der Nachrichten der Einfachheit halber wie oben einfach einen Zeitstempel verwenden. Der Empfänger der Nachrichten ist für jeden Sender fest eingestellt. Außerdem arbeiten die Sender und Empfänger mit unterschiedlichen Geschwindigkeiten (implementiert durch unterschiedliche Argumente der Funktion *sleep*)

1. Klasse EmailMessage für das Datenformat der Nachrichten:

```
package EMail;
```

```
public class EmailMessage {  
    public String receiver;  
    public String sender;  
    public String text;  
  
    public EmailMessage(String r, String s, String t) {  
        receiver = r;  
        sender = s;  
        text = t;  
    }  
}
```

## 2. Klasse *Server* für unseren „Mailserver“

```
import java.util.Vector;  
  
public class Server {  
    static final int MAXQUEUE = 5;  
    private Vector receivers = new Vector();  
    private Vector senders = new Vector();  
    private Vector texts = new Vector();  
    private String sessionPartner = new String("NONE");  
  
    public String login (String name){  
        sessionPartner = name;  
        System.out.println("Start Server Connection: "+ name);  
        return "WELCOME";  
    }  
  
    public String putMail(EmailMessage mess){  
        if(texts.size() == MAXQUEUE) return "FAILED";  
        else {  
            receivers.addElement(mess.receiver);  
            senders.addElement(mess.sender);  
            texts.addElement(mess.text);  
            return "ACCEPTED";  
        }  
    }  
  
    public boolean hasMail(String rec){  
        if(receivers.contains(rec)) return true;  
        else return false;  
    }  
  
    public EmailMessage getMail(String rec){  
        int i = receivers.indexOf(rec);  
        receivers.removeElementAt(i);  
        String text = (String)texts.elementAt(i);  
        texts.removeElementAt(i);  
        String sender = (String)senders.elementAt(i);  
    }
```

```
senders.removeElementAt(i);
EmailMessage message = new EmailMessage (rec, sender, text);
return message;
}

public void close(){
System.out.println("Stop Server Connection: "+ sessionPartner);
sessionPartner = "NONE";
}
}
```

### 3. Die Klasse *Sender* für die Absender der Nachrichten

```
public class Sender extends Thread{
Server server;
String name;
String receiver;
String text;
int time;
boolean sent;

public Sender (String n, Server s, String r, int t) {
    this.server = s;
    this.name = n;
    this.receiver = r;
    this.time = t;
}

public void run() {
    while ( true ) {

        // Neue Nachricht produzieren
        String text = new java.util.Date().toString();
        EmailMessage mess = new EmailMessage(receiver, name, text);
        sent = false;

        // Nachricht senden
        while (!sent) {
            synchronized (server) {
                if (server.login(name) == "WELCOME") {
                    if (server.putMail(mess) == "ACCEPTED") {
                        sent = true;
                        System.out.println(name+" sent message to " + mess.receiver);
                    }
                }
                server.close();
            }
        }      // Ende der Synchronisation
    try {
```

```
        sleep( time );
    }
    catch( InterruptedException e ) {
    }
}      // Ende Nachricht senden
}      // Ende while(true)
}      // Ende run()
}
```

#### 4. Die Klasse Receiver für die Empfänger

```
public class Receiver extends Thread {
    Server server;
    String name;
    int time;

    public Receiver (String n, Server s, int t) {
        this.server = s;
        this.name = n;
        this.time = t;
    }

    public void run() {
        while ( true ) {
            synchronized (server) {
                if (server.login(name) == "WELCOME") {
                    if (server.hasMail(name)) {
                        EmailMessage mess = server.getMail(name);
                        System.out.println(name+" got message from: " + mess.sender+ ".");
                        System.out.println(mess.text);
                    }
                    server.close();
                }
            } // Ende der Synchronisation
            try {
                sleep( time );
            }
            catch( InterruptedException e ) {
            }
        }
    }
}
```

#### 5. Schließlich die Klasse *MainProcess* für das Hauptprogramm

```
public class MainProcess {

    public static Server server = new Server();
```

```
public static Sender send1 = new Sender( "SENDER1", server, "RE-
CEIV1",1000 );
public static Sender send2 = new Sender( "SENDER2", server, "RE-
CEIV2",2000 );
public static Sender send3 = new Sender( "SENDER3", server, "RE-
CEIV3",3000 );
public static Receiver rec1 = new Receiver( "RECEIV1", server, 3000 );
public static Receiver rec2 = new Receiver( "RECEIV2", server, 2000 );
public static Receiver rec3 = new Receiver( "RECEIV3", server, 1000 );

public static void main(String args[]) {

    send1.start();
    send2.start();
    send3.start();
    rec1.start();
    rec2.start();
    rec3.start();
}
```

Damit können wir nun endlich unser E-Mail-System simulieren. Anfangs betrachten wir uns die Sache etwas genauer, insbesondere den Synchronisationsaspekt, und lassen uns vom Server jeweils mitteilen, wer die Verbindung mit ihm aufgenommen hat:

```
Start Server Connection: RECEIV2
Stop Server Connection: RECEIV2
Start Server Connection: RECEIV3
Stop Server Connection: RECEIV3
Start Server Connection: RECEIV1
Stop Server Connection: RECEIV1
Start Server Connection: SENDER3
SENDER3 sent message to RECEIV3
Stop Server Connection: SENDER3
Start Server Connection: SENDER2
SENDER2 sent message to RECEIV2
Stop Server Connection: SENDER2
Start Server Connection: SENDER1
SENDER1 sent message to RECEIV1
Stop Server Connection: SENDER1
Start Server Connection: RECEIV3
RECEIV3 got message from: SENDER3.
Sat Mar 25 12:02:41 CET 2000
Stop Server Connection: RECEIV3
Start Server Connection: SENDER1
SENDER1 sent message to RECEIV1
Stop Server Connection: SENDER1
```

Schließlich wollen wir uns noch eine zeitlich etwas ausgedehntere Darstellung der E-Mail-Transaktionen ansehen und kommentieren zu diesem Zeck die Server-Meldungen aus:

```
AppAccelerator(tm) 1.1.034 for Java (JDK 1.1), x86 version.  
Copyright (c) 1998 Borland International. All Rights Reserved.  
SENDER3 sent message to RECEIV3  
SENDER1 sent message to RECEIV1  
SENDER2 sent message to RECEIV2  
RECEIV3 got message from: SENDER3.  
Sat Mar 25 12:04:56 CET 2000  
SENDER1 sent message to RECEIV1  
RECEIV2 got message from: SENDER2.  
Sat Mar 25 12:04:56 CET 2000  
SENDER1 sent message to RECEIV1  
SENDER2 sent message to RECEIV2  
RECEIV1 got message from: SENDER1.  
Sat Mar 25 12:04:56 CET 2000  
SENDER3 sent message to RECEIV3  
SENDER1 sent message to RECEIV1  
RECEIV2 got message from: SENDER2.  
Sat Mar 25 12:04:58 CET 2000  
RECEIV3 got message from: SENDER3.  
Sat Mar 25 12:04:59 CET 2000  
SENDER2 sent message to RECEIV2  
SENDER1 sent message to RECEIV1
```

## 10.5 Wertung und Diskussion

Dieses Projekt wirft auf jeden Fall genügend Fragen für eine abschließende Diskussion auf. Nur einige davon sollen nicht unerwähnt bleiben:

- Wie sieht es mit der Datensicherheit aus: Wer kann welche Nachrichten abrufen?
- Was würde sich bei asynchronem Datenverkehr über „echte“ Nachrichten mit relevanter Laufzeit ändern?
- Wie könnte man das System auf mehrere Rechner im Netz verteilen?
- Welche Alternativen gibt es zum Synchronisationskonzept? Wo liegen diesbezüglich besonders kritische Stellen?
- Wie wird den einzelnen Teilprozessen (*threads*) die Rechenzeit zugeteilt (*round-robin*, Zeitscheibe)?

# 11 Das Halteproblem

Als „krönenden Abschluss“ des Informatikkurses der Kollegstufe wollen wir uns daran wagen, den Schülerinnen und Schülern wenigstens eine grobe Ahnung davon zu vermitteln, dass es grundsätzliche (logische) Beschränkungen der Berechenbarkeit gibt, die von keiner Rechenanlage, sei sie noch so leistungsfähig, jemals durchbrochen werden können. In dieser Erkenntnis steckt wohl einer der wesentlichsten Beiträge der Informatik zur Allgemeinbildung, denn gerade das Bewusstsein dieser Grenzen unterscheidet den Menschen auf Dauer von jeder noch so intelligenten Maschine.

Unter diesen Grenzen ist m.E. das Halteproblem für die Schüler gedanklich am leichtesten zugänglich, wobei ein strenger Beweis seiner Nichtentscheidbarkeit (etwa über die Gödelnummern von Turingmaschinen, siehe Wegener (1999)) sicher an (oder über) der Grenze der Möglichkeiten der Schulinformatik liegt. Wir wollen deshalb hier nur beweisen, dass es kein *imperatives Programm* geben kann, welches entscheidet, ob ein anderes imperatives Programm bei einer bestimmten Eingabe anhält. Wir schlagen eine Behandlung nach der Darstellung von Smith (1996) vor, die für Schüler ohne größere Schwierigkeiten bereits in der dortigen Form nachzuvollziehen sein dürfte, weshalb wir uns hier relativ kurz fassen können.

Am Ende des Projektes sollten die Schüler zumindest wissen, dass es grundsätzliche Grenzen maschineller (d.h. algorithmischer) Berechenbarkeit gibt sowie das Halteproblem als ein Beispiel für solche Einschränkungen kennen, formulieren und seine Unlösbarkeit begründen können

Ein alternativer, wesentlich breiter und tiefer angelegter Einstieg in diesen Problemkreis wäre z.B. in der von Brauer (1990) beschriebenen Weise über das *busy-beaver-Spiel* denkbar.

## 11.1 Aufgabenstellung

Nachdem wir in den vorausgegangenen Jahren zahlreiche Programme und Anwendungen erstellt und getestet haben, wobei uns immer wieder nichtterminierende Programme begegnet sind, könnten wir ja auf die nahe liegende Idee kommen, einen maschinellen Test auf Terminierung zu entwerfen. Zunächst schreiben wir eine informelle Funktion *Halt* dafür auf, von der wir annehmen, dass sie das gewünschte leistet. Wir lehnen uns dabei eng an Smith (1996) an, verwenden allerdings die Programmiersprache *Pascal*:

```

function Halt(prg, inp: array[1..max] of char): boolean;
(* Falls prg
  (1) ein syntaktisch korrektes MODULA-Programm ist UND
  (2) genau einen Eingabe-Befehle enthält UND
  (3) bei Eingabe von inp terminiert.
dann Halt := true,
sonst Halt := false *)

```

## 11.2 Unlösbarkeit des allgemeinen Halteproblems

Bevor wir uns an die Implementierung machen, wollen wir uns aber doch noch Gedanken darüber machen, ob es eine solche Prozedur überhaupt geben kann. Dazu nehmen wir an, wir hätten eine korrekte Implementierung von *Halt*, die wir in das folgende Programm *Test* einbauen können:

```

program Test;
const max = 65535; (* Technische Grenze, spielt logisch keine Rolle *)
var str: array[1 .. max] of char;
begin
  readln(str);
  if Halt(str, str) then
    while true do end;      (* Endlosschleife *)
    end
  end.

```

Nun verwenden wir als Eingabe für dieses Programm *Test* seinen eigenen Programmtext (kurz *TestText*). Nach Ausführung der *readln*-Anweisung ist die Variable *str* also folgendermaßen belegt:

```
str = „program Test; const max = 65535; var str: array[1 .. max] of char; begin readln(str); if Halt(str, str) then while true do end; end end.“.
```

Für das Verhalten von *Test* bei Eingabe von *TestText* gibt es nur zwei Möglichkeiten:

- Test* terminiert bei Eingabe von *TestText*: Dann kann die Funktion *Halt* bei Eingabe von *TestText* als Wert für die Parameter *prg* und *inp* nur *false* zurückgegeben haben, denn sonst wäre *Test* in die Endlosschleife geraten. Da die Bedingungen (1) und (2) aus *Halt* offensichtlich erfüllt sind, kann also Bedingung (3) nicht erfüllt sein, d.h. *Test* kann bei Eingabe von *TestText* nicht terminieren, offensichtlich im Widerspruch zur obigen Annahme.
- Test* terminiert *nicht* bei Eingabe von *TestText*: Offensichtlich kann dies nur durch die Endlosschleife verursacht worden sein. *Halt* hat also den Wert *true* geliefert. Dann muss nach (3) das Programm *Test* mit der Eingabe *TestText* terminiert haben, womit sich also auch in diesem Fall ein Widerspruch zur einleitenden Annahme ergibt.

Wie man sieht, führt also jeder mögliche Fall für das Verhalten von *Test* zu einem Widerspruch, der sich nur auflösen lässt, wenn wir die Annahme, dass es eine Funktion *Halt* mit dem obigen Verhalten gibt, aufgeben.

Da wir außer der Existenz von Funktionen sowie von Eingabe- und Wiederholungsanweisungen und der (rein technisch) begrenzten Länge der Eingabezeichenkette keine speziellen Eigenschaften von *Pascal* verwendet haben, lässt sich diese Argumentation auch auf jede andere imperativen Programmiersprache übertragen.

Die direkte Übertragung auf Sprachen ohne Wiederholungsanweisungen (z.B. streng funktionale oder logische) verursacht dagegen Probleme, da die Implementierung über eine nichtterminierende Rekursion in der Realisierung irgendwann einen Stapelüberlauf erzeugen würde. Vor einer Verallgemeinerung auf Algorithmen müsste man die Churchsche These in einer passenden Formulierung behandeln. Wir beschränken unsere Aussage also auf die Unentscheidbarkeit des allgemeinen Halteproblems in einer imperativen Sprache.

Folgerung: Es gibt kein imperatives Programm, das feststellen kann, ob ein anderes imperatives Programm bei Eingabe eines bestimmten Wertes terminiert oder nicht.

Abschließend könnte man die Entscheidbarkeit für einige andere Eigenschaften von Programmen besprechen und die Konsequenzen dieser Erkenntnisse auf maschinelle Programmüberprüfung besprechen.

# 12 Das Musterprojekt InfoBank

Den Abschluss der Folge von Unterrichtsbeispielen bildet ein Musterprojekt aus dem Anwendungsfeld „Bankwesen“, das die wichtigsten in diesem Buch besprochenen Modellierungstechnik in Form beispielhafter Aufgaben und Lösungen veranschaulicht. Die skizzierten Unterrichtsprojekte sollen dabei den kumulierten Lernerfolg der jeweiligen Ausbildungsstufe widerspiegeln. Auf eine ausführliche didaktische Darstellung der Einführung der jeweils neuen Konzepte wird hier nicht mehr eingegangen. Als roter Faden dienen dabei Aufgabenstellungen einer fiktiven Bank namens *InfoBank*. An exemplarischen Aufgabenstellungen soll gezeigt werden, welche Lernziele von den Schülerinnen und Schülern am Ende des jeweiligen Themenbereichs erreicht werden sollten.

## 12.1 Funktionen, Datenflüsse und Tabellenkalkulation

Die **InfoBank** überwacht die Girokonten ihrer Kunden. Wenn ein Kunde seinen persönlichen Dispositionskredit überschreitet, soll eine Warnung ausgegeben werden. Der Dispositionskredit ergibt sich aus dem Durchschnitt der Maximalkontostände, die für die letzten 5 Monate festgehalten wurden.

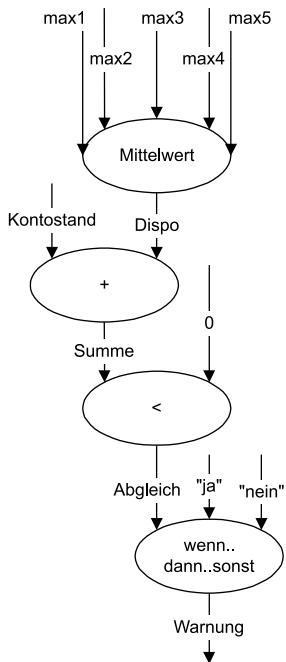
**Aufgabe 12.1:** Erstelle ein Datenflussdiagramm und einen dazu äquivalenten Term zur Entscheidung, ob ein Kunde sein Konto unzulässig (d.h. über seinen persönlichen Dispositionskredit hinaus) überzogen hat.

### Lösungen:

Der gesuchte Term lautet:

Überzogen := Wenn(Kontostand+Mittelwert(max1;max2;max3;max4;max5) < 0; „ja“; „nein“)

Das Datenflussdiagramm findet sich Abb. 12.1.



**Abb. 12.1.** Lösung zu Aufgabe 12.1

**Aufgabe 12.2:** Übertrage das Diagramm in ein Rechenblatt eines Tabellenkalkulationsprogramms.

**Lösung:** Der Term wird aufgeteilt und in eine Tabelle übertragen:

	A	B	C	D	E	F	G	H	I	J
4	Kunde	Warnung	Kto aktuell	Dispo	Maximalkontostände der letzten 5 Monate					
5					1. Monat	2. Monat	3. Monat	4. Monat	5. Monat	
6	Müller	Erna	ja	-7005,22	6069,34	6220,10	8233,22	2579,30	6211,10	7102,99
7	Huber	Hans	nein	-4522,00	5193,55	12223,98	3832,13	7987,12	1012,20	912,33
8	Hauser	Emily	ja	-12303,00	2101,01	986,23	777,43	9876,30	98,20	-1233,12
9										
10										

Die verwendeten Formeln lauten:

In Zelle C6: =WENN(D6+E6<0;"ja";"nein")

In Zelle D6: =MITTELWERT(F6:J6)

Erwähnung verdient noch der Bereichsoperator, der auch als zweistellige Funktion (Argumente sind die linke obere und die rechte untere Begrenzungszelle) mit dem entsprechenden Bereich als Rückgabewert geschrieben werden kann:

Bereich(F6;J6), kurz F6:J6

## 12.2 Datenmodelle

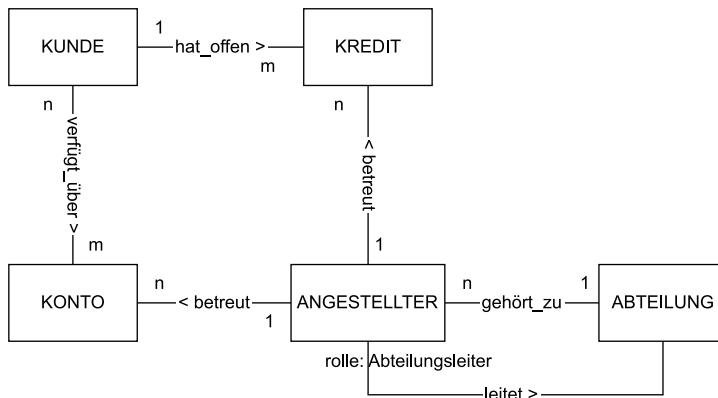
Die *InfoBank* will ihre Kunden, Konten, Angestellten Abteilungen und Kredite einander zuordnen. Damit sollen bestimmte Auswertungen, z.B. über die vergebenen Kredite, den Kontostand der Kunden möglich sein. Die Ausgabe soll für bestimmte Auswertungen auch Adresslisten der betreffenden Kunden liefern können, um postalische Rundschreiben (besondere Angebote, Warnungen etc.) an diese verschicken zu können.

**Aufgabe 12.2:** Erstelle ein Datenmodell für diese Aufgaben, setze dieses in eine relationale Datenbank um und führe zu den folgenden Fragestellungen je eine Abfrage durch:

- Durchschnittliche Höhe der Kredite eines Betreuers.
- Adresse und Betreuer von Kunden, deren Kredit den Betrag von 1 Mio € übersteigt.

### Lösungen:

Das Datenmodell findet sich in Abb. 12.2.



**Abb. 12.2.** Datenmodell zu Aufgabe 12.2

Die Umsetzung (mit dem Werkzeug MS-Access) wird in Abb. 12.3 lediglich durch die Schemata der Tabellen und Beziehungen dargestellt.

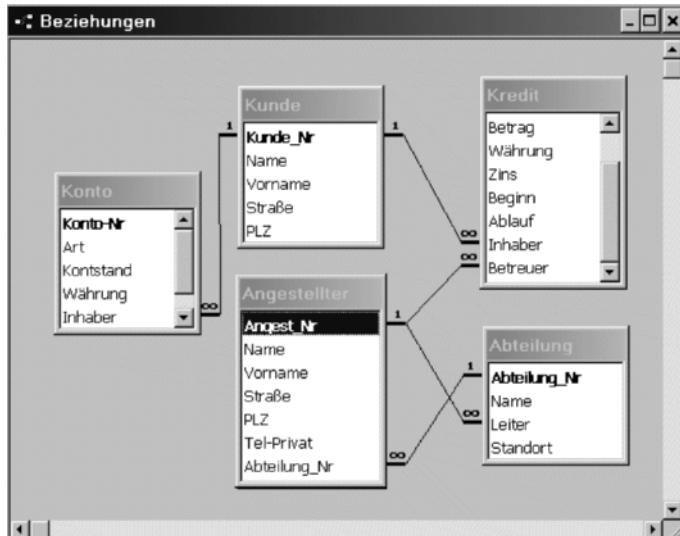


Fig 12.3. Umsetzung des Datenmodells in eine relationale Datenbank

Die interaktive Erstellung der Abfragen (siehe Abb. 12.4 und 12.5) generiert in MS-Access die jeweils darunter angegebenen SQL-Ausdrücke.

Kredit Abfrage : Auswahlabfrage			
Betreuer	Mittelwert von Betrag	Name	Vorname
A1002112	3594084,28	Müllermann	Heinz-Georg
A1201123	455170,37	Nörgelhuber	Petra

Abb. 12.4. Abfrage a) in MS-Access

```

SELECT DISTINCTROW Kredit.Betreuer, Avg(Kredit.Betrag) AS [Mittelwert von Betrag], Angestellter.Name, Angestellter.Vorname
FROM Angestellter INNER JOIN Kredit ON Angestellter.Angest_Nr = Kredit.Betreuer
GROUP BY Kredit.Betreuer, Angestellter.Name, Angestellter.Vorname;
    
```

Aufgabe-b : Auswahlabfrage							
Betrag	Kunde.Name	Kunde.V	Straße	PLZ	Angest_Nr	Angestellter.I	Angestellter.Vo
12231993,12	Meier	Alfons	Staatstr. 17	83099	A1002112	Müllermann	Heinz-Georg
2001222,00	Huber	Erna	Rilkestr. 22a	62400	A1002112	Müllermann	Heinz-Georg
1122211,00	Knödermann	Gustav	Kastenweg 19	55023	A1201123	Nörgelhuber	Petra

Abb. 12.5. Abfrage b) in MS-Access

```

SELECT Kredit.Betrag, Kunde.Name, Kunde.Vorname, Kunde.Straße, Kunde.PLZ, Angestellter.Angest_Nr, Angestellter.Name, Angestellter.Vorname, Angestellter.Abteilung_Nr
FROM Kunde INNER JOIN (Angestellter INNER JOIN Kredit ON Angestellter.Angest_Nr = Kredit.Betreuer) ON Kunde.Kunde_Nr = Kredit.Inhaber
WHERE (((Kredit.Betrag)>1000000));

```

## 12.3 Zustandsmodelle

Die *InfoBank* betreibt eine Reihe von Geldautomaten.

**Aufgabe 12.3:** Wir wollen den Ablauf eines dieser Geldautomaten simulieren. Erstellen Sie dazu zunächst ein Zustands-Übergangsdiagramm. Simulieren Sie

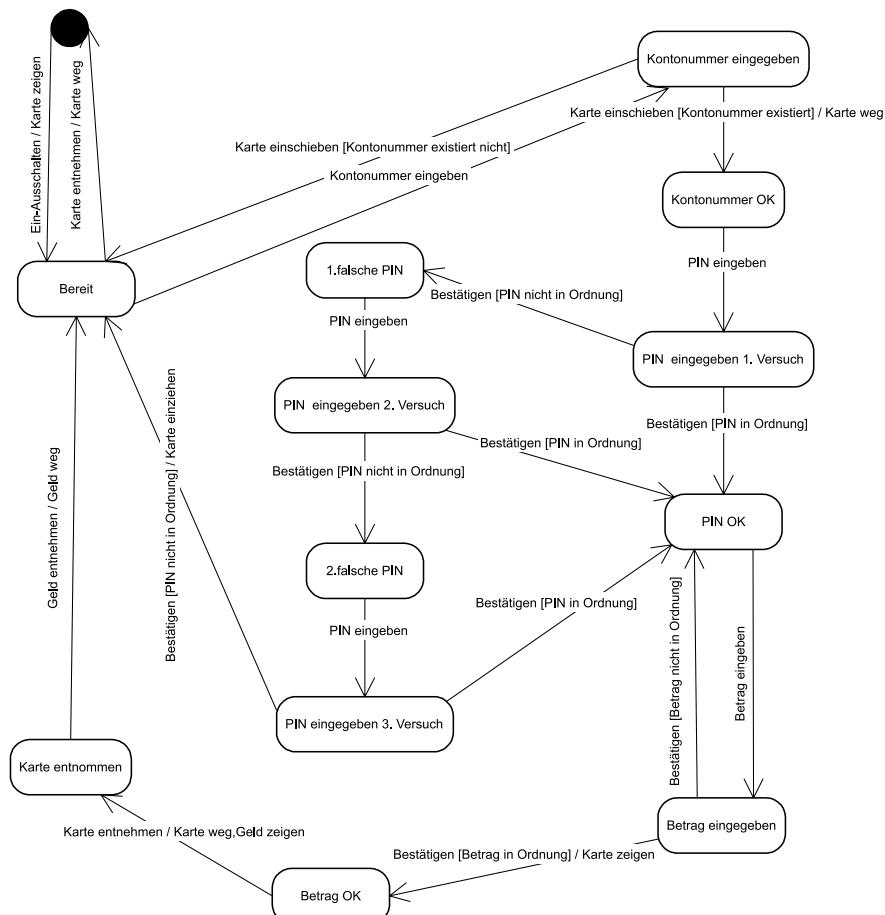


Abb. 12.6. Zustandsmodell des Geldautomaten

den Ablauf dann durch die Umsetzung Ihres Diagramms in eine geeignete Programmiersprache. Nutzen Sie dazu die Tabelle „Konto“ aus dem letzten Projekt der Infobank in der JGSt. 9

### Lösungen:

Das Zustandsdiagramm ist in Abb. 12.6 abgebildet. Die Benutzeroberflächen der gesuchten Programme (MS Visual Basic) in Abb. 12.7.



**Abb. 12.7.** Benutzerschnittstelle in zwei Zuständen

## 12.4 Objektmodelle

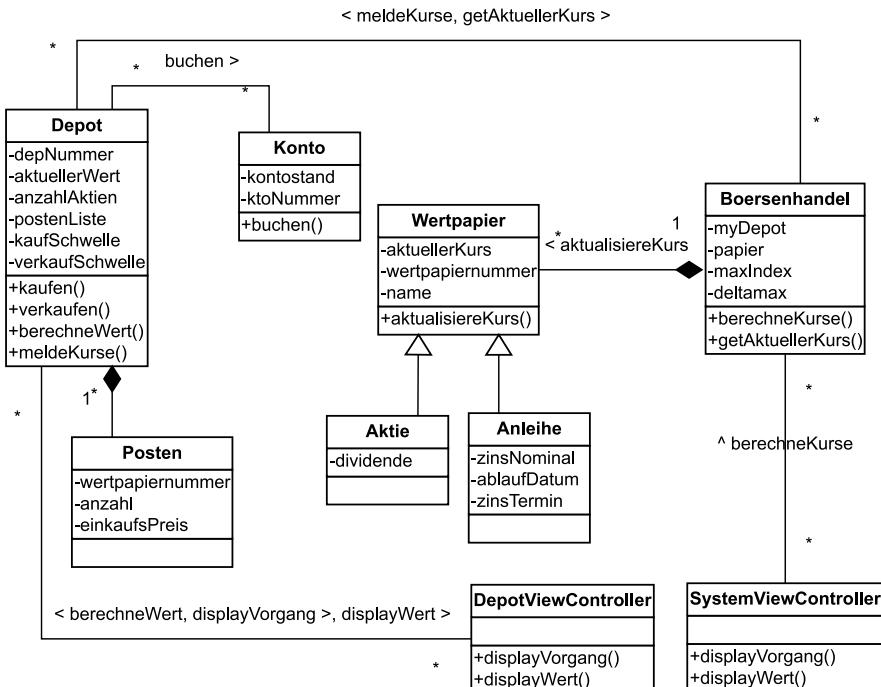
Die *InfoBank* will ihren Wertpapierservice automatisieren. Dabei sollen die folgenden Nutzungsfälle abgedeckt werden: Kunden kaufen und verkaufen Wertpapiere (Aktien oder Anleihen) je nach Kursentwicklung. Die Papiere werden als Posten (je Wertpapierart ein Posten) in einem Depot des Kunden verwaltet. Kosten und Erlöse werden über ein spezielles, dem Depot zugeordnetes Wertpapierkonto abgerechnet.

**Aufgabe 12.4:** Erstellen Sie zunächst ein geeignetes Klassenmodell. Simulieren Sie den Handel durch Implementierung Ihres Objektmodells in eine geeignete objektorientierte Programmiersprache. Vereinfachen Sie die Situation dazu wie folgt: Wertpapierposten werden gekauft, wenn der Kurs um einen bestimmten Prozentsatz steigt und verkauft, wenn der Kurs um einen bestimmten Prozentsatz fällt. Simulieren Sie die Kursentwicklung schrittweise (1 Schritt je Börsentag) durch Zufallszahlen. Entwerfen Sie das System zunächst ohne Benutzung einer Datenbank mit minimaler Benutzeroberfläche für zwei verschiedene Depots mit unterschiedlichen Kauf- Verkaufsschwellen.

Ergänzen Sie Ihr System dann um eine Datenbankanbindung, in der die Daten für Kunde, Konten, Depots und Wertpapiere persistent gespeichert werden können. Übernehmen Sie dabei soweit möglich Datenbankfragmente aus der bisherigen Arbeit an der *InfoBank*.

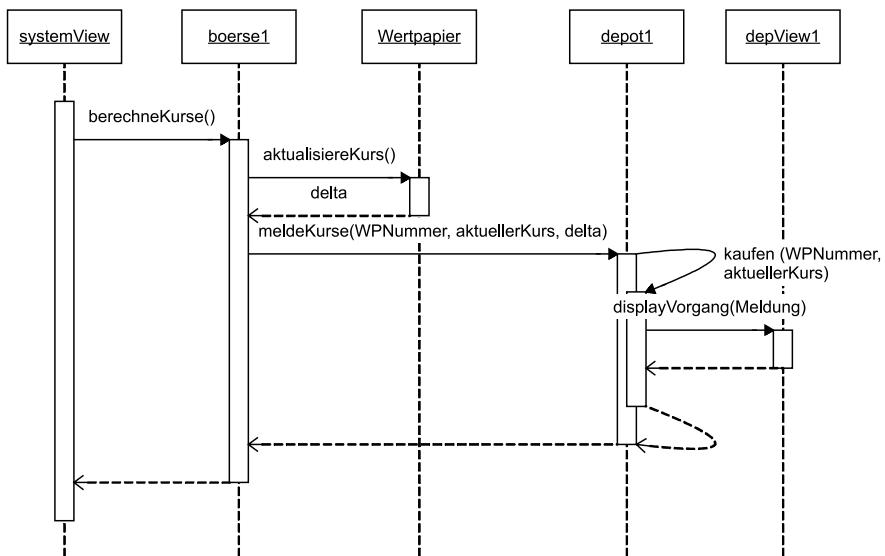
### Lösung:

Wir modellieren die Situation wie folgt: Die Börse enthält eine Reihe von Wertpapieren. Sie ist getaktet und berechnet pro Takt die Kurse ihrer Papiere. Die Depots werden über die Kursentwicklung benachrichtigt und entscheiden dann über Kauf oder Verkauf der einzelnen Papiere. Wertpapiere werden im Depot mit Hilfe von Posten verwaltet (ein Posten je Wertpapiernummer). Abb. 12.8 zeigt das Klassendiagramm.



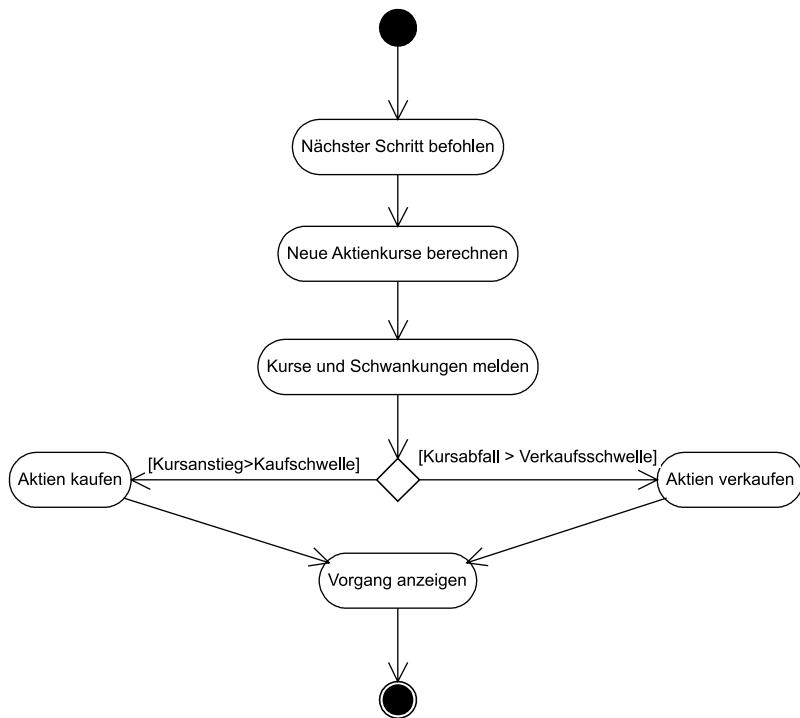
**Abb. 12.8.** Klassenmodell für das Wertpapiersystem

Die Interaktionen ergeben sich aus den Interaktionsdiagrammen diverser Szenarien, z.B. eines Kaufvorganges (siehe Abb. 12.9).



**Abb 12.9.** Interaktionsdiagramm des Kaufvorgangs

Für die Modellierung der Abläufe ist hier ein Zustandsdiagramm wenig hilfreich, weshalb wir hier ein Aktivitätsdiagramm vorziehen (siehe Abb. 12.10).



**Abb. 12.10.** Aktivitätsdiagramm des Wertpapiersystems

Für die Implementierung in Java nehmen wir die folgenden Vereinfachungen vor: Wir betrachten zwei Depots ohne Kunde und Konto, beschränken uns auf 10 Wertpapiere (5 Aktien und 5 Anleihen). Die Benutzerschnittstelle erlaubt folgende Eingaben: Ausführung des nächsten Schritts und Abbruch des Programms. Sie kann folgende Ausgaben erzeugen: Wert der Depots berechnen und Anzeige der Kauf- Verkaufsvorgänge. Abb. 12.11. zeigt die fertige Benutzerschnittstelle.

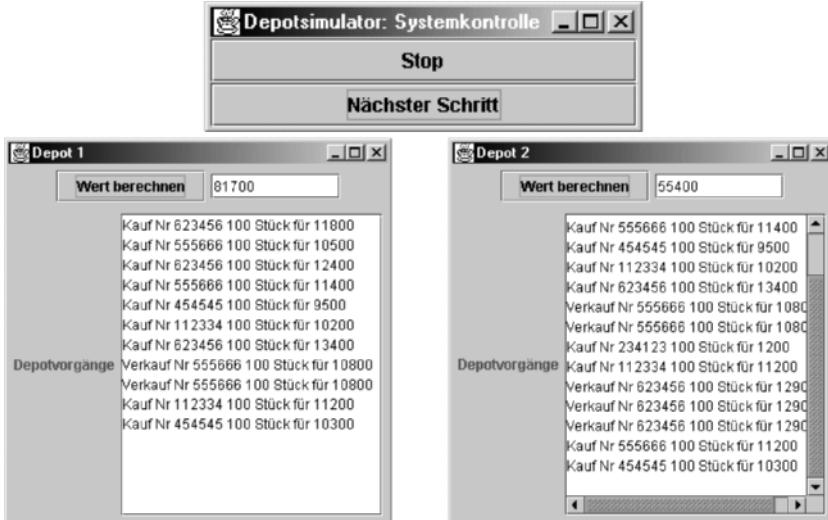


Abb. 12.11. Benutzerschnittstelle für das Wertpapiersystem

# Literatur

- Anderson J.R.: Cognitive Psychology and its Implications. Freeman & Co, N.Y. 1985
- Anderson J.R.: Kognitive Psychologie. Eine Einführung. Spektrum Akademischer Verlag, Heidelberg, 2. Aufl. 1989
- Anderson J.R.: Language, Memory and Thought. Erlbaum, Hillsdale, N.J. 1976
- Aschersleben K.: Einführung in die Unterrichtsmethodik, Kohlhammer, Stuttgart, 4. Aufl. 1984
- Bandura A., Walters R.: Social Learning and Personality Development. Holt, Rinehart & Winston, N.Y. 1963
- Bandura A.: Principles of Behaviour Modification. Holt, Rinehart & Winston, N.Y. 1969
- Bauer F.L.: Entzifferte Geheimnisse. Methoden und Maximen der Kryptologie. Springer, Berlin, 2. Aufl. 1997
- Bauer F.L., Goos G.: Informatik. Eine einführende Übersicht. Erster Teil. Springer, Berlin, 3. Aufl. 1982
- Bauer F.L., Goos G.: Informatik. Eine einführende Übersicht. Zweiter Teil. Springer, Berlin, 3. Aufl. 1984
- Bauer F.L.: Top down teaching im Informatikunterricht. In: Weinhart K. (Hrsg.): Informatik im Unterricht. Oldenbourg, München 1979
- Baues J., Hillebrand H.-P., Hüster E., Mersch B.: Informatik erleben. Lehr- und Übungsbuch für Sekundarstufe I. Teil 1,2. Dümmler, Bonn 1996
- Baumann R.: Didaktik der Informatik. Klett, Stuttgart 1990, 2. Aufl. 1996
- Baumann R.: Fundamentale Ideen der Informatik – gibt es das? In: Koerber B., Peters I.R. (Hrsg.): Informatische Bildung in Deutschland. Perspektiven für das 21. Jahrhundert. LOG IN Verlag, Berlin 1998, S. 89–107
- Bayerische Staatskanzlei: Softwaretechnik. Ein Bericht des Wissenschaftlich-Technischen Beirats der Staatsregierung, München 1996
- Beer U.: Wie werde ich schöpferisch? Zeitwende 4 (1970), S. 270
- Berger R.: Success Factors in Electronic Commerce. Studie der Roland Berger & Partner GmbH, Frankfurt/Main, Februar 1999
- Bloom, B.S. (ed.): Taxonomy of Educational Objectives: The Classification of Educational Goals. Handbook I: Cognitive Domain. David McKay, N.Y. 1956.
- Booch G., Rumbaugh J., Jacobson I.: Unified Modeling Language – Semantics and Notation Guide 1.0. Rational Rose Software Corp., San Jose, Cal. 1997
- Booch G.: Object-Oriented Analysis and Design. Benjamin/Cummings, Redwood City, Cal. 1994
- Brauer et al.: Zielsetzungen und Inhalte des Informatik-Unterrichtes. Zentralblatt für Didaktik der Mathematik 8 (1976), S. 35
- Brauer W.: Von der sequentiellen zur parallelen Verarbeitung. HMD – Theorie und Praxis der Wirtschaftsinformatik, 26:150 (1989), S. 15–25
- Brauer W.: Grenzen maschineller Berechenbarkeit. Informatik Spektrum 13 (1990). S. 61–70
- Brauer W., Brauer U.: Better Tools – Less Education? In Ritter G.X. (ed.): Information Processing 89. Elsevier Science Publishers, Amsterdam 1989, pp. 101–106

- Brauer W., Brauer U.: Informatik an der Schule – weshalb und wie? In: Rechnerkunde. Algorithmen und DVA-Strukturen im Schulunterricht. 4. Paderborner Werkstattgespräch 1972. Forschungs- und Entwicklungszentrum für objektive Lehr- und Lernverfahren. Schöning, Paderborn 1973, S. 34–35
- Breier N.: Informatische Bildung als Teil der Allgemeinbildung. LOG IN 13: 5/6 (1994), S. 90
- Brenner A., Gunzenhäuser R.: Informatik. Didaktische Materialien für Grund- u. Leistungskurse. Klett, Stuttgart 1982
- Broadbent D.E.: Perception and Communication. Pergamon Press, London 1958
- Broy M., Geisberger E., Grosu R., Huber F., Rumpe B., Schätz B., Schmidt A., Slotosch O., Spies K.: Das AUTOFOCUS-Bilderbuch. Institut für Informatik der TU München, Mai 1996
- Broy M.: Informatik. Eine grundlegende Einführung, Bd. 1: Programmierung und Rechnerstrukturen. Springer, Berlin, 2. Aufl. 1997
- Broy M.: Informatik. Eine grundlegende Einführung, Bd. 2: Systemstrukturen und Theoretische Informatik. Springer, Berlin, 2. Aufl. 1998
- Bruner J. S.: Contemporary Approaches to Cognition. Harvard University Press, Cambridge, Mass. 1957
- Bruner J.S.: The Process of Education. Harvard University Press, Cambridge, Mass. 1960
- Bruner J.S.: The Act of Discovery. Harvard Educational Review 31 (1961), pp. 123–152
- Bundesministerium für Bildung und Forschung, Bundesministerium für Wirtschaft und Technologie: Innovation und Arbeitsplätze in der Informationsgesellschaft des 21. Jahrhunderts. Aktionsprogramm der Bundesregierung, Berlin 1999
- Bund-Länderkommission für Bildungsplanung und Forschungsförderung: Medienerziehung in der Schule. Orientierungsrahmen. Materialien zur Bildungsplanung und Forschungsförderung, Heft 44, Bonn 1995
- Bund-Länderkommission für Bildungsplanung und Forschungsförderung: Rahmenkonzept Informationstechnische Bildung in Schule und Ausbildung, Bonn 1984
- Burkert J.: Informatikunterricht – Quo vadis? Thesen zu Stand und Entwicklung der Schul-Informatik. In: Schubert (1995), S. 49–52
- Burkert J.: Umorientierung des Informatikunterrichtes. LOG IN 14:5/6 (1994), S. 86–89
- Bussmann H., Heymann H.-W.: Computer und Allgemeinbildung. Neue Sammlung 1 (1987), S. 2–39
- Büttemeyer W.: Wissenschaftstheorie für Informatiker. Spektrum Akademischer Verlag, Heidelberg 1995
- Chen P.: The Entity-Relationship Model – toward a Unified View of Data. ACM Trans. on Database Systems 1:1 (1976), pp. 9–36
- Cube, F. von: Der kybernetische Ansatz in der Didaktik. Die Deutsche Schule 60 (1968), S. 391–400
- Cyranek G., Forneck H.J., Goorhuis H. (Hrsg.): Beiträge zur Didaktik der Informatik. Diesterweg/Sauerländer, Frankfurt/Main 1990
- Dale E., Audio-Visual Methods in Teaching. Dryden Press, N.Y. 1954
- Dave R.H.: Eine Taxonomie pädagogischer Ziele und ihre Beziehung zur Leistungsmessung. In: Ingenkamp K., Marsolek T. (Hrsg.): Möglichkeiten und Grenzen der Testanwendung in der Schule. Beltz, Weinheim 1968
- Derbolav J.: Versuch einer wissenschaftlichen Grundlegung der Didaktik. In: Didaktik in der Lehrerbildung. Zeitschrift für Pädagogik, 2. Beiheft (1960), S. 17–45
- Deutsch J.A., Deutsch D.: Attention: Some Theoretical Considerations. Psychological Review 70 (1963), pp. 80–90
- Deutscher Bildungsrat (Hrsg.): Strukturplan für das Bildungswesen, Stuttgart 1970
- Dewey J.: Demokratie und Erziehung. Westermann, Braunschweig 1964

- Dubs R.: Konstruktivismus: Einige Überlegungen aus der Sicht der Unterrichtsgestaltung. *Zeitschrift für Pädagogik* 6 (1995), S. 889–903
- Duden Fremdwörterbuch. Dudenverlag, Mannheim 1997
- Duden Informatik. Dudenverlag, Mannheim, 2. Aufl. 1993
- Edelmann W.: Lernpsychologie – Eine Einführung. Urban & Schwarzenberg, München/Weinheim, 2. Aufl. 1986
- Eickel J.: Einführung in die Informatik I. Skriptum zur Vorlesung aus dem WS 1994/95. Institut für Informatik der TU München 1995
- Engbrink D.: Kultur- und technikgeschichtlich begründete Bildungswerte der Informatik. In: Schubert (1995), S. 68–77
- Facchi C.: Formal Semantics of Time Sequence Diagrams. Technischer Bericht der TU München, TUM-I9540, Dezember 1995
- Fakultätentag Informatik: Empfehlungen zum Schulfach Informatik, insbesondere zur Ausbildung von Informatiklehrern. Chemnitz 1996
- Forneck H.J.: Entwicklungstendenzen und Problemlinien der Didaktik der Informatik. In: Cyranek et al. (1990), S. 18–53
- Frank H.: Ein Ansatz zu einer kybernetisch-pädagogischen Lehrplanungstheorie. Neue Unterrichtspraxis (1974), S. 340–347
- Fraser B.J., Walberg H.J., Welch W.W., Hattie J.A.: Synthesis of Educational Productivity Research. *International Journal of Educational Research* 11 (1987), pp. 145–252
- Friedrich S., Neupert H.: Lernen mit Netzen – Lernen über Netze. *LOG IN* 17:6 (1997), S. 18–23
- Friedrich S., Schubert S., Schwill A.: Informatik in der Schule – ein Fach im Wandel. In *LOG IN* 16:2 (1996), S. 29
- Friedrich S.: Grundpositionen eines Schulfaches. *LOG IN* 15: 5/6 (1995), S. 30–34
- Frost & Sullivan (2003)
- Füller K.: Objektorientiertes Programmieren in der Schulpraxis. In: Schwill (1999), S. 190–201
- Gagné R.M.: *The Conditions of Learning*. Holt, Rinehart & Winston, N.Y. 1985
- Gerstenmaier J., Mandl H.: Wissenserwerb unter konstruktivistischer Perspektive. *Zeitschrift für Pädagogik* 6 (1995), S. 867–885
- Gesellschaft für Informatik: Empfehlungen zur Lehrerausbildung und Lehrerweiterbildung für Informatik und informationstechnische Grundbildung. *LOG IN* 19:1 (1999), S. I–XII
- Giese B.: Informations- und Kommunikationstechnik: Die Basis für die Märkte der Zukunft in der Wissensgesellschaft. Aktualisierte Übersicht. DLR, IT-GE im Auftrag des Bundesministeriums für Bildung und Forschung. Bonn, November 1997
- Glaser P.: Arbyter aller Länder! Süddeutsche Zeitung, Magazin 17 (1995), S. 16
- Glöckel H.: Vom Unterricht. Lehrbuch der allgemeinen Didaktik. Klinkhard, Bad Heilbrunn, 3. Aufl. 1996
- Gordon C.W., Wayne C.: Die Schulkasse als ein soziales System. In: Heintz P. (Hrsg.): *Soziologie der Schule*. Kölner Zeitschrift für Soziologie und Sozialpsychologie, Sonderheft 4 (1965)
- Hampel Th., Magenheim J., Schulte C.: Dekonstruktion von Informatiksystemen als Unterrichtsmethode. Zugang zu objektorientierten Sichtweisen im Informatikunterricht. In: Schwill (1999), S. 149–164
- Hauf-Tulodziecki A.: Informatische Bildung und Medienerziehung. In: Schwill (1999), S. 121–129
- Hebb D.O.: *The Organisation of Behaviour*. Wiley, N.Y. 1949
- Heidegger M.: Sein und Zeit. Niemeyer, Tübingen, 14. Aufl. 1977

- Heimann P., Otto G., Schulz W.: Unterricht – Analyse und Planung? Schroedel, Hannover 1965
- Heimann P.: Didaktik als Theorie und Lehre. Die Deutsche Schule (1962), S. 407ff.
- Herbart J.F.: Allgemeine Pädagogik. In: Asmus W. (Hrsg.): Pädagogische Schriften, Bd. 2: Pädagogische Grundschriften. Küpper, Düsseldorf 1965
- Hopcroft J.E., Ullman J.D.: Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie. Addison-Wesley, Bonn, 1994
- Hoppe U.H., Luther W.J.: Informatik und Schule. LOG IN 16:1 (1996), S. 8–14
- Horn C., Kerner I.O. (Hrsg.): Lehr- und Übungsbuch Informatik, Bd. 1: Grundlagen und Überblick. Fachbuchverlag, Leipzig 1995
- Horn C., Kerner I.O. (Hrsg.): Lehr- und Übungsbuch Informatik, Bd. 3: Praktische Informatik. Fachbuchverlag, Leipzig 1997
- Hubwieser P.: Informatik als Pflichtfach an bayerischen Gymnasien. In: Schwill (1999), S. 165–174
- Hubwieser P.: Modellierung in der Schulinformatik. LOG IN 19:1 (1999a), S. 24–29
- Hubwieser P., Broy M.: Der informationszentrierte Ansatz. Ein Vorschlag für eine zeitgemäße Form des Informatikunterrichtes am Gymnasium. Technischer Bericht der TU München, TUM-I9624, Mai 1996
- Hubwieser P., Broy M.: Ein neuer Ansatz für den Informatikunterricht am Gymnasium. LOG IN 17:3/4 (1997), S. 42–47
- Hubwieser P., Broy M.: Grundlegende Konzepte von Informations- und Kommunikationssystemen für den Informatikunterricht. In: Hoppe H.U., Luther W. (Hrsg.): Informatik und Lernen in der Informationsgesellschaft. 7. GI-Fachtagung „Informatik und Schule“, Duisburg 1997. Informatik aktuell, Springer, Berlin 1997a, S. 40–50
- Hubwieser P., Broy M.: Educating Surfers or Craftsmen: Introducing an ICT Curriculum for the 21st Century. In: Downes T., Watson D. (eds.): Communications and Networking in Education: Learning in a Networked Society. IFIP WG 3.1/3.5 Conference, Aulanko-Hämeenlinna, Finland 1999. Helsingin Yliopisto, Helsinki 1999, pp. 162–170
- Hubwieser P., Broy M., Brauer W.: A New Approach to Teaching Information Technologies: Shifting Emphasis from Technology to Information. In: Passey u. Samways (1997), pp. 115–121
- Hubwieser P.: Functional Modelling in Secondary Schools Using Spreadsheets. Education and Information Technologies 9 (2), June 2004: pp. 175–183
- Frey E., Hubwieser P., Winhard F.: Informatik 1. Objekte, Strukturen, Algorithmen. Klett, Stuttgart, 2004
- Hubwieser P.: Von der Funktion zum Objekt - Informatik für die Sekundarstufe 1. In Friedrich S. (Hrsg.): Unterrichtskonzepte für die informatische Bildung. INFOS 2005 – 11. GI-Fachtagung Informatik und Schule, 28.–30.9.2005 in Dresden, GI-Edition Lecture Notes in Informatics. P-60, Bonn, 2005, S. 27–41
- Hubwieser P., Schneider M., Spohrer M., Voß S.: Informatik 2. Tabellenkalkulationssysteme, Datenbanken. Klett, Stuttgart, 2007
- Jacobson I., Christerson M., Jonsson P., Övergard G.: Object-Oriented Software Engineering – a Use Case Driven Approach. Addison-Wesley, Reading, Mass. 1991
- Kahneman D.: Attention and Effort. Prentice-Hall, Englewood Cliffs, N.J. 1973
- Keller J. M.: Strategies for stimulating the motivation to learn. Performance and Instruction 26 (8) 1987, pp. 1–7. The systematic process of motivation. Performance and Instruction 26 (9) 1987, pp. 1–8
- Kemper A., Eickler A.: Datenbanksysteme. Eine Einführung. Oldenbourg, München, 3. Aufl. 1999
- Kerschensteiner G.: Begriff der Arbeitsschule. Oldenbourg, München 1950

- Klafki W.: Didaktische Analyse als Kern der Unterrichtsvorbereitung. *Die Deutsche Schule* 50 (1958), S. 450–471
- Klafki W.: Studien zur Bildungstheorie und Didaktik. Beltz, Weinheim 1963
- Klafki W.: Das pädagogische Problem des Elementaren und die Theorie der kategorialen Bildung. Beltz, Weinheim 1964
- KMK (Sekretariat der Ständigen Konferenz der Kultusminister der Länder in der Bundesrepublik Deutschland, Hrsg.): Einheitliche Prüfungsanforderungen in der Abiturprüfung Informatik. Beschuß vom 1.12.1989. Luchterhand, Neuwied 1991
- Knapp T., Fischer H.: Objektorientierung im Informatikunterricht. Ein didaktisches Konzept zum Einstieg in den Informatikunterricht der Sekundarstufe I. *LOG IN* 18:3/4 (1998), S. 71–76
- Koerber B., Peters I.R.: Grundlagen einer Didaktik der Informatik. FU Berlin, Sommersemester 1988
- Koerber B., Peters I.R.: Informatikunterricht und informationstechnische Grundbildung – ausgrenzen, abgrenzen oder integrieren? In: Troitzsch (1993), S. 108–115
- Koffka K.: Perception. An Introduction to Gestalt Theory. *Psychological Bulletin* 19 (1922), pp. 531–585
- Köhler W.: Intelligenzprüfungen an Menschenaffen. Springer, Berlin 1921
- Kopp F.: Didaktik in Leitgedanken. Auer, Donauwörth, 6. Aufl. 1970
- Korth H. F., Silberschatz A.: Database System Concepts. McGraw-Hill, N.Y. 1991
- Krathwohl D.R., Bloom B.S., Masia B.B.: Taxonomy of Educational Objectives, Handbook 2: Affective Domain. Addison-Wesley, Reading, Mass. 1984
- Kruglanski H.W. et al.: Retrospective Misattribution and Task Enjoyment. *Journal of Experimental Social Psychology* 8 (1972), pp. 439–501
- Kultusministerium des Landes Mecklenburg-Vorpommern: Rahmenplan Informatische Bildung, Schwerin 1998
- Lefrancois G.R.: Psychologie des Lernens. Springer, Berlin, 3. Aufl. 1994
- Lewin K.: Die Lösung sozialer Konflikte. Christian, Bad Nauheim 1953
- LISW (Landesinstitut für Schule und Weiterbildung): Neue Informations- und Kommunikationstechnologien. Soest 1987
- Mager R.F.: Lernziele und Programmierter Unterricht. Beltz, Weinheim 1969
- Malz P.: Ausnahmestatus. Manager Magazin Juli 1998, S. 120ff.
- Meier A.: Relationale Datenbanken. Eine Einführung für die Praxis. Springer, Berlin, 3. Aufl. 1998
- Meier M.W.: Anforderungen an die Informatikausbildung in den neunziger Jahren aus der Sicht der Wirtschaft. In: Cyranek et al. (1990), S. 55–73
- Meißner H. (Hrsg.): Informatik I, II. Der Mathematikunterricht 1 (1971), 5 (1975)
- Meyer H.: Unterrichtsmethoden I: Theorieband. Cornelsen, Frankfurt/Main 1987
- Niemeyer P., Peck J.: Java – Expeditionen ins Programmierreich. O'Reilly, Bonn, 1997
- Nievergelt J.: „Roboter programmieren“ – ein Kinderspiel. Bewegt sich auch etwas in der Allgemeinbildung? *Informatik-Spektrum* 22:5 (1999) S. 364–375
- Nievergelt J.: Was ist Informatik-Didaktik? Gedanken über die Fachkenntnisse eines Informatiklehrers. *Informatik-Spektrum* 16:1 (1993), S. 3–10
- Passey D., Samways B. (eds.): Information Technology. Supporting Change through Teacher Education. IFIP WG 3.1/3.5 Conference, Kiryat Anavim, Israel 1996. Chapman & Hall, London 1997, pp. 115–121
- Pattis R.E.: Karel the Robot: A Gentle Introduction to the Art of Programming. Wiley, N.Y. 1981
- Peterßen W.H.: Handbuch Unterrichtsplanung. Ehrenwirth, München, 4. Aufl. 1982
- Piaget, J.: Gesammelte Werke. Klett, Stuttgart 1975

- Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. Hanser, München 1997
- Rechenberg P.: Quo vadis Informatik? LOG IN 17:1 (1997), S. 25–32
- Rechenberg P.: Was ist Informatik? Eine allgemeinverständliche Einführung. Hanser, München 1994
- Reichenberger K., Steinmetz R.: Visualisierungen und ihre Rolle in Multimedia-Anwendungen. Informatik-Spektrum 22:2 (1999), S. 88–98
- Reinmann-Rothmeier G., Mandl H.: Lernen auf der Basis des Konstruktivismus. Computer und Unterricht 23 (1996), S. 41–44
- Reisig W.: Systementwurf mit Netzen. Springer, Berlin, 1985
- Robinson S.B.: Bildungsreform als Revision des Curriculum. Luchterhand, Neuwied 1971
- Roth H.: Pädagogische Psychologie des Lehrens und Lernens. Schroedel, Hannover, 7. Aufl. 1963
- Rudolph E., Graubmann P., Grabowski J.: Tutorial on Message Sequence Charts. Computer Networks and ISDN Systems 28 (1996), pp. 1629–1641
- Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W.: Object-Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs, N.J. 1991
- Rüttgers J.: Schulen ans Netz. DIE ZEIT 39 (1997), S. 50f.
- Savola T.: Using HTML. Que Corp., Indianapolis, Ind. 1995
- Schäfer K.-H., Schaller K.: Kritische Erziehungswissenschaft und kommunikative Didaktik. Quelle & Meyer, Heidelberg 1971
- Schöning U.: Theoretische Informatik - kurzgefasst. Spektrum Akademischer Verlag, Heidelberg, 1995
- Schrack G.: Grafische Datenverarbeitung. Bibliographisches Institut, Mannheim 1978
- Schröder H.: Leistung in der Schule. Arndt, München 1990
- Schubert S. (Hrsg.): Innovative Konzepte für die Ausbildung. 6. GI-Fachtagung „Informatik und Schule“, Chemnitz 1995. Informatik aktuell, Springer, Berlin 1995
- Schubert S.: Einführung in die Didaktik der Informatik. Vorlesungsskript, Fakultät für Informatik der TU Chemnitz 1997
- Schubert S.: Fachdidaktische Fragen der Schulinformatik und (un)mögliche Antworten. In: Gorny P. (Hrsg.): Informatik: Wege zur Vielfalt beim Lehren und Lernen. 4. GI-Fachtagung „Informatik und Schule“, Oldenbourg 1991. Informatik-Fachberichte, Springer, Berlin 1991, S. 27–33
- Schulmeister R.: Multimedia. Neue Technologien im Hochschulunterricht. In: Handbuch Hochschullehre. Raabe, Bonn 1994, B 1.1
- Schulz-Zander R. et al.: Veränderte Sichtweisen für den Informatik-Unterricht. GI-Empfehlungen für das Fach Informatik in der Sekundarstufe II allgemeinbildender Schulen. In: Troitzsch (1993), S. 203–218
- Schulz-Zander R.: Analyse curricularer Ansätze für das Schulfach Informatik. In: Arlt W. (Hrsg.): EDV-Einsatz in Schule und Ausbildung. Oldenbourg, München 1978, S. 40–49
- Schulz-Zander R.: Eröffnungsrede zur 6. GI-Fachtagung „Informatik und Schule“ 1995. Computer und Unterricht 11 (1996), S. 69
- Schwill A.: Programmierstile im Anfangsunterricht. In: Schubert (1995), S. 178–187
- Schwill A.: Computer Science Education Based on Fundamental Ideas. In: Passey u. Samways (1996), pp. 285–291
- Schwill A. (Hrsg.): Informatik und Schule. Fachspezifische und fachübergreifende didaktische Konzepte. 8. GI-Fachtagung „Informatik und Schule“, Potsdam 1999. Informatik aktuell, Springer, Berlin 1999
- Schwinn H.: Relationale Datenbanksysteme. Hanser, München 1992
- Seibert N., Serve H.J. (Hrsg.): Prinzipien guten Unterrichts. PimS, München 1992
- Seibert N.: Das Unterrichtsprinzip der Strukturierung. In: Seibert u. Serve (1992), S. 197–219

- Seibert N.: Das Unterrichtsprinzip der Veranschaulichung. In: Seibert u. Serve (1992a), S. 245–265
- Serve H.J.: Das Unterrichtsprinzip der Kreativitätsförderung. In: Seibert u. Serve (1992a), S. 127–164
- Serve H.J.: Das Unterrichtsprinzip der Motivierung. In: Seibert u. Serve (1992), S. 165–195
- Shannon C.E: A mathematical theory of communication. The Bell System Technical Journal 27 (1948), pp. 379–423
- Skinner B.F.: The Behaviour of Organisms. An Experimental Analysis. Appleton-Century-Crofts, N.Y. 1938
- Smith E.: Elementare Berechenbarkeitstheorie. Springer, Berlin, 1996
- Stetter F., Brauer W.: Zukunftsperspektiven der Informatik für Schule und Ausbildung. 3. GI-Fachtagung „Informatik und Schule“, München, 1989. Informatik-Fachberichte, Springer, Berlin 1989
- Tanenbaum A.S.: Moderne Betriebssysteme. Hanser, München, 2. Aufl. 1995
- Tanenbaum A.S.: Computer Networks. Prentice-Hall, Upper Saddle River, N.J., 3<sup>rd</sup> edition, 1996
- Tausch R., Tausch A.-M.: Erziehungspsychologie. Hogrefe, Göttingen, 2. Aufl. 1965
- Thorndike E.L.: The Psychology of Learning. Teacher's College, N.Y. 1913
- Tolman E.C., Honzik C.H.: Insight in Rats. University of California Publications in Psychology 4 (1930), pp. 215–232
- Treisman A.M.: Verbal Cues, Language and Meaning in Selective Attention. American Journal of Psychology 77 (1964), pp. 206–219
- Troitzsch K.G. (Hrsg.): Informatik als Schlüssel zur Qualifikation. 5. GI-Fachtagung „Informatik und Schule“, Koblenz 1993. Informatik aktuell, Springer, Berlin 1993.
- Tulving E., Donaldson W. (eds.): Organization of Memory. Academic Press, N.Y. 1972
- Uhlig A.: Zum Begriff und zur Unterscheidung der Lehrmethoden. Wissenschaftliche Zeitschrift der Friedrich-Schiller-Universität Jena 1953/54, S. 497–507
- van Lück W.: Können Lehr- oder Übeprogramme eigentlich gut sein? Computer und Unterricht 23 (1996), S. 45–48
- Wagenschein M.: Verstehen lernen. Beltz, Weinheim 1970
- Walter R.: Petrinetzmodelle verteilter Algorithmen. Beweistechnik und Intuition. Dissertation an der Humboldt-Universität. Bertz, Berlin, 1995
- Watson J.B.: Psychology as the Behaviourist views it. Psychological Review 20 (1913), pp. 157–158
- Wedekind H., Görz G., Kötter R., Inhetveen R.: Modellierung, Simulation, Visualisierung: Zu aktuellen Aufgaben der Informatik. Informatik-Spektrum 21:5 (1998), S. 265–272
- Wegener, I.: Theoretische Informatik – eine algorithmische Einführung. Teubner, Stuttgart, 1999
- Weinert F. Analyse und Unterscheidung von Lehrmethoden. In: Ingenkamp K. (Hrsg.): Handbuch der Unterrichtsforschung, Teil III. Beltz, Weinheim, 2. Aufl. 1971, S. 1217ff.
- Weniger E.: Didaktik als Bildungslehre, Teil 1: Theorie der Bildungsinhalte und des Lehrplans. Beltz, Weinheim 1956
- Wertheimer M.: Productive Thinking. Harper & Row, N.Y. 1945
- Winograd T., Flores F.: Understanding Computers and Cognition. A New Foundation for Design. Ablex Publishing Corporation, Norwood, N.J. 1986

# **Index**

## **A**

Abfrage 168  
Abfrageschema 168  
Aktion 220  
Algorithmus 51, 134, 182  
Allgemeinbildung 57, 104  
Alternative 190  
Anschaulichkeit 20  
Anwendung 51  
ARCS-Modell 17  
Artikulation 35  
Attribut 117  
Aufgabe 68  
Aufmerksamkeit 13  
Auswahl 169  
Automat 178

## **B**

Backus-Naur-Form 237  
Bedienerschulung 46  
Bedingte Anweisung 190  
Behaviourismus 3  
Benutzer 52  
Bereichsoperator 150  
Bericht 168  
Berichtsschema 168  
Berliner Didaktik 26  
Berufsvorbereitung 64  
Bestrafung 4  
Bewertung 21, 72  
Beziehung 121, 158  
Bildungsauftrag 57  
Bildungstheoretischer Ansatz 25  
binäre Bäume 225  
Botschaft 119  
Bürosoftware 73

## **C**

Chiffrierung 202

## **D**

Darstellung 79  
Datei 122  
Datenbanksystem 155  
Datenfluss 148  
Datenflussdiagramm 147, 203, 265  
Datenmodell 267  
Datenmodellierung 92  
datenorientierte Modellierung 155  
Datenredundanz 164  
Datenspeicher 148  
Datenstruktur 113  
Datenverlust 164  
Datenverschlüsselung 201  
deterministischer endlicher Automat 240  
Differenzierung 22  
Dokument 121

## **E**

E-Mail 124  
E-Mail-Adresse 128  
E-Mail-System 127  
Entity-Relationship-Model 155, 156  
Entwicklungspsychologie 7  
Equi-Join 170  
Ereigniss 220  
Erfolgssicherung 21  
Erzeuger-Verbraucher-Situation 248

**F**

- Fallunterscheidung 192
- Filtertheorie 13
- Flexibilität 22
- Formel 149
- Führungsstil 37
- fundamentale Ideen 82
- Fundamentum 100
- Funktion 147
- funktionale Abhängigkeit 166
- funktionale Modellierung 90, 147

**G**

- Gedächtnis 11
- Getränkeautomat 180
- Göttinger Schule 25
- Gymnasium 99

**H**

- Halteproblem 261
- Hardware 50
- Hauptschule 108
- Hypertext 130
- Hypertextsystem 74

**I**

- Implementierung 72
- Infixschreibweise 150
- Informatiksysteme 43
- Informatikunterricht 43, 48
- Information 78
- Informations- und Kommunikationssysteme 43
- informationstechnische Grundbildung 49
- Informationsverarbeitung 79
- Informationszentrierung 78
- informatische Bildung 43, 48
- Inkonsistenz 164
- Interaktionsdiagramm 96
- Interpretation 80

**J**

- Jahresplan 32

**K**

- Kapazitätsmodell 13
- Kardinalität 159
- kausale Beziehung 220
- Kellerautomat 180
- Klasse 117
- Klassendiagramm 123, 213
- Klassenmodell 270
- Kognitive Flexibilität 11
- Kognitive Handwerkslehre 11
- kognitiver Zwischenprozess 5
- Kognitivismus 5
- Kommunikative Didaktik 28
- Konditionierung 4
- Konstruktivismus 10
- Kreativität 17
- Kreuzprodukt 170
- Kybernetik 27

**L**

- Lebensdauer 84
- Lehr- und Lernformen 36
- Lehr- und Lernmethoden 35
- Lehrerverhalten 37
- Lehrplan 32
- Lerninhalt 30, 77
- Lernpsychologie 3
- Lernsoftware 44
- Lernstörung 13
- Lerntheoretischer Ansatz 26
- Lernziel 33
- Lernzieltaxonomie 33

**M**

- Makroprogrammierung 206
- Mealy-Automat 180
- Medien 39
- Medienerziehung 59
- Methode 119
- methodische Prinzipien 68
- Modell 86
- Modellbildung 69
- Modellierung 71, 85, 87
- Modellierungstechnik 90
- Modellzustand 187
- Monitor 221

Monitorkonzept 221

Motivation 15

Motive 15

Motivierung 15

## N

Narrativer Anker 10

Natural Join 171

Nebenhäufigkeit 219, 245

Neuron 5

Nichtterminale 238

Normalform 163

## O

Oberstufe 106, 225

Objekt 115

Objektmodell 211

objektorientierte Modellierung 209

Operante Kontrolle 7

Operation 119

Operationalisierung von Lernzielen  
34

Ordner 121, 123

## P

Parallelität 220

Petri-Netz 248

Pflichtfach 70

Problem 68

Problembegrenzung 71

Problembeschreibung 71

Problemorientierung 68

Programm 182

Programmoberfläche 75

Programmiersprache 74, 182

Programmierung 87

Projektion 169

Protokoll 246

Prozess 185, 220

Punktnotation 115

## R

Rastergrafik 138

Realschule 108

Rechnerkommunikation 245

Reizkontrolle 7

Rekursion 226

relationale Algebra 169

relationale Modellierung 160

Relationen 160

Repräsentationsform 137

## S

Schema 162

Schlüssel 162

Semaphor 221

Sequenz 184

Sequenzdiagramm 219

Serienbrief 134

Simulation 69

Situierter Erkenntnis 10

Sozialform 37

SQL 172

Standardsoftware 113

Strukturierung 18

Studienvorbereitung 65

Subsysteme 147

Symbolische Kontrolle 7

Synchronisation 221

Syntax 237

Syntaxdiagramm 239

System 86

## T

Tabellenkalkulationssystem 147

Teilsystem 147

Terminale 238

Textverarbeitungsprogramm 121

Textverarbeitungssystem 120

Thread-Konzept 221

transitiv abhängig 166

Turingmaschine 180

## U

Übung 19

Unterordner 123

Unterricht 29

Unterrichtsmedium 72

Unterrichtsmodul 100

Unterrichtsskizze 111

**V**

Variabilität 22  
Variable 183  
Vektorgrafik 140  
Veranschaulichung 20  
Verklemmung 221  
Vermittelbarkeit 84  
Verstärkung 4

**W**

Wahlmodul 103  
WENN-Funktion 151  
Wiederholung 194

**Z**

zeitliche Planung 32  
Zellbereich 150  
Zellbezug 149  
Zelle 149  
Zuordnung 148  
Zustandsmodellierung 178  
Zustandsorientierte Modellierung  
93  
Zustands-Übergangsdiagramm 178,  
269  
Zuweisung 184