Vorlesungen
zur
Didaktik der Informatik

von

Andreas Schwill

Wintersemester 1996/1997

Didaktik der Informatik von Andreas Schwill

"Didaktik der Informatik" im Erweiterungsstudium Informatik

Umfang: 2V+2Ü
Beginn: 4.4.98
Weitere Termine: 18.4./ 9.5./ 16.5./ 6.6./ 13.6./ 27.6./ 4.7.
Ort: Kleiner Physik-Hörsaal 1.09.1.11 (Universität Potsdam)

Literaturhinweise

R. Baumann: "Didaktik der Informatik", Klett Verlag 1990
J.S. Bruner: "The process of education", Cambridge Mass. 1960
Frey, K.: "Die sieben Komponenten der Projektmethode - mit Beispielen aus dem Schulfach Informatik", LOGIN 3,2 (1983) 16-20
Hänsel, D.: "Was ist Projektunterricht, und wie kann er gemacht werden?", D. Hänsel, H. Müller (eds.): "Das Projektbuch Sekundarstufe", Beltz Verlag (1988) 13-45
E. Modrow: "Didaktik des Informatikunterrichts", Dümmler Verlag 1991, 2 Bände
A. Schwill: "Fundamentale Ideen der Informatik", Zentralb. für Did. der Mathem. (1991)
Themenheft "Projektunterricht" der Zeitschrift LOGIN, Heft 5/6 (1992)

Skriptum

Begleitend zur Vorlesung erscheint ein Skript. pdf

TEIL A

1 Was ist Informatik ?

Informatik ist die Wissenschaft von der systematischen Verarbeitung und Speicherung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von Computern.

Alternative Definitionen bezeichnen die Informatik als die Wissenschaft

- von der Struktur, den Computer-Sprachen und der Programmierung von DV-Anlagen sowie der Methodik ihrer Anwendung einschließlich der Mensch-Maschine-Wechselwirkung. [Zweites Datenverarbeitungsprogramm der Bundesregierung 1971]
- (*algorithmenorientierte Sichtweise*) mindestens von den Algorithmen und Datenstrukturen sowie deren Darstellung und Realisierung unter besonderer Berücksichtigung digitaler Rechenanlagen. [Claus 1974]
- (*informationstheoretische Sichtweise*) that has as its domain information processes and related phenomena in artifacts, society and nature. [Nygaard 1990]
- (*arbeitswelt-orientierte Sichtweise*) von der Analyse von Arbeitsprozessen und ihrer konstruktiven, maschinellen Unterstützung. Nicht die Maschine, sondern die Organisation und Gestaltung von Arbeitsplätzen steht als wesentliche Aufgabe im Mittelpunkt der Informatik. Die Gestaltung der Maschinen, der Hardware und der Software ist dieser primären Aufgabe untergeordnet. Informatik ist also nicht „Computerwissenschaft“. [Coy 1989]
- of the systematic study of algorithmic processes that describe and transform information; their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is „What can be (efficiently) automated?“. [Association for Computing Machinery 1989]

2 Teilgebiete der Informatik

Man unterscheidet innerhalb der Informatik sechs Teilgebiete: die Theoretische, die Praktische, die Technische und die Angewandte Informatik, ferner Gesellschaftliche Bezüge der Informatik und Didaktik der Informatik. Theoretische, Praktische und Technische Informatik sowie ingenieurmäßige Anteile der Angewandten Informatik faßt man oft unter dem Oberbegriff „*Kerninformatik*“ zusammen.

Theoretische Informatik: Sowohl für die Formulierung und Untersuchung von Algorithmen als auch für die Rechnerkonstruktion spielen Methoden und Modelle aus der Mathematik eine wesentliche Rolle. Da die Struktur von Computern ständig komplexer wird, nimmt auch der Abstraktionsgrad einer angemessenen Beschreibung zu. Für die Untersuchung von Fragestellungen in diesem Bereich sind gute Kenntnisse der strukturellen Mathematik nötig, die eine Reihe von formalen Methoden zur Beschreibung von Systemen bietet. Bei der Untersuchung von Sprachen spielen Methoden der Logik eine wichti-

ge Rolle.
Beispiele für Teilgebiete der Theoretischen Informatik sind Formale Sprachen, Theorie der Netze und Prozesse, Automatentheorie, Semantik und Komplexitätstheorie.

Praktische Informatik: Algorithmen lassen sich zwar prinzipiell rechnerunabhängig formulieren; um sie aber auf Rechenanlagen bearbeiten zu lassen, muß der Computer zu einem komfortablen Werkzeug gemacht werden. Programme, geschrieben in *maschinenunabhängigen* Programmiersprachen, müssen von speziellen Übersetzungsprogrammen in eine dem Rechner verständliche und ausführbare Form übertragen werden; ein *Betriebssystem* überwacht die Ausführung der Programme, die bei größeren Rechenanlagen oftmals gleichzeitig ablaufen, und übernimmt die Steuerung der Ein- und Ausgabe; *Informationssysteme* verwalten umfangreiche Datenbestände; Programmsysteme sorgen für viele Hilfsfunktionen (Programme erstellen, testen, archivieren usw.). Diese oder ähnliche Aufgaben haben andererseits wieder Auswirkungen auf den Entwurf neuer Computer. Dafür sind auch Kenntnisse der Elektrotechnik nötig, damit der Informatiker die Möglichkeiten und Grenzen technischer Realisierungen abschätzen kann.
Beispiele für Teilgebiete der Praktischen Informatik sind Übersetzerbau, Informationssysteme, Betriebssysteme, Simulation und Künstliche Intelligenz.

Technische Informatik: In der Technischen Informatik befaßt man sich mit dem funktionellen Aufbau von Computern und den zugehörigen Geräten sowie mit dem logischen Entwurf und der konkreten Entwicklung von Rechnern, Geräten und Schaltungen (*Hardware*). Die Schnittstelle zu Betriebssystemen und die Zusammenstellung von Computern spielen eine wichtige Rolle.
Beispiele für Teilgebiete der Technischen Informatik sind Rechnerarchitektur, Prozeßdatenverarbeitung, Fehlertoleranz, Leistungsmessung und VLSI-Entwurf.

Angewandte Informatik: Informatik versteht sich als *anwendbare* Wissenschaft, die viele Sparten des Lebens beeinflußt. Unter Angewandter Informatik faßt man Anwendungen von Methoden der Kerninformatik in anderen Wissenschaften und die Entwicklung spezieller Verfahren und Darstellungstechniken zusammen. Die Angewandte Informatik untersucht Abläufe in den unterschiedlichsten Bereichen auf ihre Automatisierbarkeit durch Computer. Im Vordergrund steht dabei das *ingenieurmäßige* Vorgehen bei der Entwicklung von *Software*, d.h. von Programmsystemen, die solche Anwendungsfälle abdecken sollen. Gewisse aus verschiedenen Anwendungen stammende gemeinsame Gebiete sind ergonomische Fragen und Probleme der Mensch-Maschine-Kommunikation. Dies überschneidet sich auch mit Forschungsgebieten aus der Psychologie (Kognitions-Wissenschaften), aus der Pädagogik (Training und Erklärungsprobleme), aus den Sozial- und Wirtschaftswissenschaften (Arbeitsstrukturen, Organisationsfragen, Führungssysteme) usw. Die Grenze zwischen Angewandter und Praktischer Informatik ist fließend.

Beispiele für Teilgebiete der Angewandten Informatik sind die Betriebsinformatik, die Rechtsinformatik und die medizinische Informatik. (Solche Gebiete werden in der Literatur auch als „Bindestrich-Informatik“ zusammengefaßt.)

Gesellschaftliche Bezüge der Informatik: Informatik hat starke Auswirkungen auf die Gesellschaft. Eine Gesellschaft wird von Informationen und Informationsflüssen geprägt, deren Automatisierung auf Entscheidungsprozesse einwirkt. Andererseits beeinflußt der Einsatz von Computern die Arbeitswelt und den Freizeitbereich nachhaltig. Ähnlich wie bei der Entwicklung mechanischer Maschinen (Schlüsselerfindungen: Dampfmaschine, Verbrennungsmotor; industrielle Revolution) wird auch der Computer als Instrument der *Rationalisierung* eingesetzt, woraus sich für die Betroffenen oft schwerwiegende soziale Folgen (Wandel von Arbeitsplätzen und beruflichen Anforderungen) ergeben. Weiterhin sind Regelungen über den Umgang mit schutzwürdigen Daten zu treffen. Untersuchungen hierzu werden im Gebiet *Informatik und Gesellschaft* zusammengefaßt. In den letzten Jahren erkannte man auch zunehmend die Gefahren, die sich aus der schnellen Verfügbarkeit *personenbezogener Daten* und der Konzentration von Informationen in *Datenbanken* ergeben: Mögliche Einschränkung der Rechte des Einzelnen und Entstehung neuer Abhängigkeiten bzw. Machtverhältnisse durch die Verfügungsgewalt über Informationen. Schließlich kann der Computer zur Steuerung, Informationssammlung und -auswertung auf fast allen Gebieten von Wirtschaft, Wissenschaft, öffentlichem und privaten Leben eingesetzt werden und ermöglicht allein aufgrund seiner Arbeitsgeschwindigkeit die Lösung immer neuer, immer komplexerer Probleme.

Didaktik der Informatik: Hiermit befassen wir uns ausführlich in Teil B.

3 Informatik im Wissenschaftsgefüge

Historisch betrachtet hat sich die Informatik - zuvor vor allem als Spezialgebiet betrieben - aus der Mathematik, einer Grundlagenwissenschaft, und der Elektrotechnik, einer Ingenieurwissenschaft, entwickelt und ist selbst zu einer Grundlagenwissenschaft geworden, deren Methoden und Ergebnisse in nahezu allen anderen Wissenschaften verwendet werden. Letzteres dokumentiert sich z.B. in Bezeichnungen wie Betriebsinformatik, Rechtsinformatik, Medizinische Informatik, Wirtschaftsinformatik. Wie ordnet sich die Informatik als neue Wissenschaft in den Kanon der bestehenden Wissenschaften ein?

Eine Zuordnung der Informatik zu den *Geisteswissenschaften* scheidet aus, da Informatik sich nicht allein auf die Gewinnung und Darstellung neuer Erkenntnisse beschränkt. Vielmehr ist eines ihrer zentralen Ziele der Entwurf und die Herstellung praktisch einsetzbarer Produkte. Informatik behält also stets auch den anwendungsorientierten Aspekt im Auge. Dennoch besitzt die Informatik eine erhebliche geisteswissenschaftliche Komponente.

Informatik ist auch keine klassische *Naturwissenschaft*. Zwar untersucht sie Prozesse, die auch in der Natur anzutreffen sind, etwa informationsverarbeitende Prozesse, ihre wichtigsten Forschungsgegenstände (Maschinen, Algorithmen, Datenstrukturen) sind jedoch von Menschenhand geschaffen.

Ingenieurwissenschaften befassen sich überwiegend mit konkreten technischen Objekten, etwa in der Elektrotechnik oder dem Maschinenbau. Die Objekte der Informatik sind dem gegenüber häufig abstrakter Natur, etwa Information, Algorithmus, Datum, Prozeß. Daher kann man die Informatik auch nicht den Ingenieurwissenschaften zuordnen. Allerdings besitzt die Informatik vor allem im Bereich ihrer Anwendungen hohe ingenieurmäßige Anteile. Auch die professionelle Softwareentwicklung wird mehr und mehr durch ein ingenieurartiges Vorgehen geprägt. Diese Schwerpunktverlagerung hat bereits an einigen Universitäten zur Bildung neuer Studiengänge „Ingenieurinformatik“ geführt. Langfristig ist mit einer Aufspaltung der Informatik in eine ingenieurmäßige und eine grundlagenorientierte Wissenschaft zu rechnen. Die Informatik hätte dann eine ähnliche Entwicklung vollzogen, wie viele andere Wissenschaften vorher, etwa das Abspalten der Elektrotechnik und des Maschinenbaus von der Physik, der Chemietechnik und der Verfahrenstechnik von der Chemie usw.

Eine detaillierte Analyse des Wissenschaftsgefüges hat C.F. von Weizsäcker in seinem Buch „Die Einheit der Natur“ angestellt. Er ordnet die Informatik - wie auch die Mathematik - in die Klasse der **Strukturwissenschaften** ein. Innerhalb der Informatik (und der Mathematik) untersucht und erforscht man auf formaler Ebene strukturelle Eigenschaften von Objektklassen (z.B. das operationale Verhalten der Objekte, die Eigenschaften von Operationen), zunächst noch ohne zu berücksichtigen, welche konkreten Objekte sich dieser Struktur unterordnen und ob es überhaupt solche Objekte gibt.

Beispiel: Ganz typisch ist diese strukturorientierte Denkweise bei der Spezifikation. Module spezifiziert man häufig durch *abstrakte Datentypen*, wobei zunächst nur die Struktur des Datentyps, d.h. die Eigenschaften der zugeordneten Operationen auf den *Sorten* (=Bezeichner für Wertebereiche), festgelegt wird. Eine Bedeutung erhält die Definition erst durch *Interpretation*, also durch Assoziierung der Sorten und Operationen mit Objekten der realen Welt. Ggf. stellt sich dabei heraus, daß es überhaupt keine (widersprüchliche Definition), genau einen (*monomorpher* Datentyp) oder mehrere (*polymorpher* Datentyp) reale Objektwelten gibt, die sich der definierten Struktur unterordnen. In der Praxis interessieren aber meist nur monomorphe Datentypen.
Das folgende Beispiel illustriert die Begriffe:

```
type Beispiel
    sorts: M
    operations:
        a: → M
```

s: $M \to M$
p: $M \to M$

laws:

©x∈ M: p(s(x))=x
©x∈ M, x≠a: s(p(x))=x
©x∈ M: s(x)≠x
©x∈ M, x≠a: p(x)≠x

end.

(© bedeutet hier für alle“)Offenbar gibt es in dem abstrakten Datentyp einen Wertebereich M und drei Operationen, eine nullstellige Operation, also eine Konstante a aus M, und zwei einstellige Operationen s und p von M in sich, die sich gemäß den Gesetzen verhalten. Mindestens zwei Objektwelten ordnen sich dieser Spezifikation unter:

1. Man assoziiert M mit IN_0, a mit der Null, s mit der Nachfolgerfunktion und p mit der Vorgängerfunktion. Diese Zuordnung erfüllt die Spezifikation.
2. Man assoziiert M mit {false, true}, a mit false, s und p mit der Negation. Auch diese Interpretation erfüllt die Spezifikation.

Der abstrakte Datentyp ist also polymorph.

Von der Mathematik unterscheidet sich die Informatik durch die Betonung unterschiedlicher Aspekte: Mathematik ist die Wissenschaft statischer idealisierter Strukturen, Informatik die Wissenschaft dynamischer realer Strukturen. von Weizsäcker spricht in diesem Zusammenhang von der Informatik als der „Mathematik zeitlicher Vorgänge, die durch menschliche Entscheidung, durch Planung, durch Strukturen, die sich darstellen lassen, als seien sie geplant, oder schließlich durch Zufall gesteuert werden“. Die Mathematik stellt also vor allem begriffliche, die Informatik überwiegend methodische Hilfsmittel bereit. Ein weiterer wichtiger Unterschied besteht in der Art der in der Mathematik und Informatik überwiegend behandelten Größen und Objekte. Anders als in der Mathematik, die meist mit kontinuierlichen und elementaren Größen operiert, sind die in der Informatik vorkommenden Objekte strukturiert (d.h. aus mehreren unterscheidbaren elementaren Größen zusammengesetzt) und in höchstem Maße diskret. Letzteres ergibt sich aus der Darstellung von Größen auf einem Digitalrechner. Da nur ein endliche Menge unterscheidbarer Zeichen zur Verfügung steht, müssen alle Objekte durch diskrete Größen approximiert werden.

V. Claus faßt die Unterschiede von Mathematik und Informatik in sieben Thesen zusammen:

1) Mathematik trainiert das Denken in statischen abstrakten Räumen, während in der Informatik geübt wird, im zeitlichen Nach- und Nebeneinander zu denken.
2) Die Objekte der betrachteten Räume werden in der Mathematik meist nur selten auf ihre innere Struktur untersucht, dagegen sind in der Informatik die Datenstrukturen von zentraler Bedeutung.
3) Die Darstellung der Algorithmen und Datenstrukturen und Untersuchungen über Zeit und Platz, die für die Ausführung und Speicherung notwendig sind, spielen in der Ma-

thematik im Gegensatz zur Informatik eine untergeordnete Rolle.

4) Die Untersuchungs-Objekte der Mathematik unterliegen im allgemeinen keinen Einschränkungen, während in der Informatik eine Bevorzugung diskreter Strukturen vorherrscht. (Die Mathematik ist daher „abstrakter“, die Informatik „konkreter“)
5) Die Mathematik-Ausbildung betont den „Einzelkämpfer“, die Informatik hat in hohem Maße auch zum Mitarbeiter in Gruppen auszubilden. (Einzelaufgaben contra Projektdurchführung)
6) Der Informatiker benötigt für seinen Beruf im wesentlich größeren Umfange als der Mathematiker „handwerkliche“ Fähigkeiten und Erfahrungen (konkreter Umgang mit Rechnern, Beherrschung von mindestens zwei Programmiersprachen, genaue Kenntnis von Programmsystemen usw.).
7) Die Auswirkungen der Informatik betreffen die Gesellschaft und den Einzelnen unmittelbar, während der Mathematiker von der Öffentlichkeit kaum bemerkt wird (nur indirekt über andere Wissenschaften).

4 Informatik als Kulturtechnik, Computer literacy

In immer stärkerem Maße wird von jedem einzelnen erwartet, daß er mit Computern umgehen kann. Die Kenntnis über Computer ist aber bei der Mehrzahl der Bürger noch relativ beschränkt und häufig von falschen Vorstellungen geprägt. Die Möglichkeiten, mit der Arbeitsweise und Leistungsfähigkeit einer Rechenanlage vertraut zu werden, sind relativ gering und meist auf den Arbeitsbereich begrenzt. So wird in Zeitungen gewöhnlich nur über spektakuläre Fehler berichtet, die Computer verursacht haben, während die ständigen Veränderungen, die durch neue Einsatzmöglichkeiten der Computer in der Gesellschaft bewirkt werden, kaum behandelt werden.

Die neue Wortschöpfung „Computer literacy (Computer-Alphabetisierung)“ konkretisiert dieses Defizit. Sie drückt die Forderung aus, daß der Umgang mit und die gezielte Nutzung von Computern zu einer zentralen *Kulturtechnik* wie Lesen, Schreiben und Rechnen wird. Computer literacy umfaßt die Kenntnis über Computer und deren Verwendungsmöglichkeiten und die Fähigkeit, Computer zur Bewältigung von Aufgaben im Beruf, im häuslichen Bereich, in der Gestaltung der Freizeit usw. einzusetzen.

Es ist anzunehmen, daß in Zukunft jeder in folgenden Gebieten Kenntnisse besitzen wird:

- Geschichte und Entwicklung von Datenverarbeitungsanlagen;
- Funktionseinheiten, aus denen sich ein Computer zusammensetzt, Kommunikationseinheiten und Vernetzung sowie Kenntnisse zugehöriger Begriffe wie „Eingabe“, „Ausgabe“, „Programm“, „Programmiersprache“, „Datei“, „Bit“, „Byte“, „Peripherie“ usw.;
- Einsicht in die Arbeitsweise des Betriebssystems und die Datenverarbeitungsanlage und Fähigkeit, als Benutzer mit einem Computer zu arbeiten;
- Kenntnis über Sprachen und vorhandene Anwendungs- und Hilfsprogramme (Software)

sowie Erstellen und Verändern einfacher Programme;
- Analyse eines Problems hinsichtlich der Möglichkeit, es durch einen Computer lösen zu lassen, und logisch fundierte Vermittlung des Problems an einen Programmierer;
- Bewertung von Programmen nach verschiedenen Gesichtspunkten (Effizienz, Benutzerfreundlichkeit, Nutzen des Einsatzes, Zuverlässigkeit usw.);
- Auswirkungen der Datenverarbeitung auf die Gesellschaft und den einzelnen; Chancen, Nutzen und Gefahren ihres Einsatzes.

Von einer Computer literacy breiter Kreise der Bevölkerung kann zur Zeit noch keine Rede sein. Momentan bestehen noch zu große Barrieren, die es den meisten Menschen nicht ermöglichen, eine entsprechende Grundqualifikation in Informatik zu erwerben. Dies wird sich durch die Einführung eines Pflichtfachs „Informationstechnische Grundbildung“ in die Schule sowie mit wachsender Durchdringung aller Schulfächer mit Informatikmethoden und -denkweisen in absehbarer Zeit ändern.

TEIL B

1 Was ist Didaktik der Informatik?

Laut Meyers Enzyklopädischem Lexikon Bd. 6 von 1974 ist Didaktik allgemein

> „Unterrichtslehre; Kunst des Lehrens; Wissenschaft von der Methode des Unterrichtens“,

wobei unterschiedliche Schwerpunkte möglich sind:

1. Didaktik als Wissenschaft und Lehre vom Lehren und Lernen überhaupt,
2. Didaktik als Wissenschaft vom Unterricht bzw. Theorie des Unterrichts,
3. Didaktik als Theorie der Steuerung von Lernprozessen,
4. Didaktik als Theorie der Lehr- bzw. Bildungsinhalte, ihrer Struktur, Auswahl und Zusammensetzung,
5. Didaktik als Theorie der Unterrichtsformen und -verfahren.

Während die Akzentuierungen auf die Punkte 1.-3. und teilweise auch 5. weitgehend fachneutral innerhalb der Pädagogik behandelt werden können, setzt eine Fachdidaktik, also eine Unterrichtslehre für ein bestimmtes Fachgebiet, ihren Forschungsschwerpunkt vor allem im Bereich 4. Denn die Auswahl und Zusammensetzung von geeigneten Inhalten ist von zentraler Bedeutung für die Vermittlung von Begriffen und Prinzipien der Informatik. Besonderes Gewicht ist auf prägnante Beispiele zu legen: Sie bleiben einerseits lange haften, und sie komprimieren Information, d.h., selbst komplizierte Sachverhalte lassen sich meist außerordentlich leicht wieder über die Beispiele erschließen. Die Vorlesung wird sich daher zum Schluß und in den Übungen vor allem mit Beispielen befassen, die für den Informatikunterricht einerseits interessant und fesselnd sind und andererseits zur Vermittlung unterschiedlicher Konzepte der Informatik besonders geeignet erscheinen.

Die zentrale Fragestellung einer Fachdidaktik lautet also pauschal:

Was soll wann, wie und mit welchem Ziel gelehrt werden?

Genauer: Fachdidaktik stellt einen Bezug zwischen einer Fachwissenschaft und der Lebenswelt her; sie macht die von der Fachwissenschaft gewonnenen Erkenntnisse für die Schule oder allgemein für Aus-, Fort- und Weiterbildung von Kindern und Erwachsenen verfügbar. Im einzelnen gehören zu diesem Prozeß insbesondere folgende Aufgaben:

- Definition der Ziele des Fachunterrichts,
- Entwicklung von Konzepten zur Methodik und zur Organisation des Unterrichts,
- Festlegung, welche Ideen, Methoden und Erkenntnisse der Fachwissenschaft im Unterricht vermittelt werden sollen,
- Reihung der Unterrichtsinhalte zu Lehrplänen und ihre fortlaufende Aktualisierung hinsichtlich neuester fachwissenschaftlicher und didaktischer Erkenntnisse.

Didaktik der Informatik und ganz allgemein Fachdidaktik ist keine in sich ruhende Wissenschaft, vielmehr strahlen eine Reihe anderer Wissenschaften (z.B. Psychologie) und Institutionen (z.B. die Schule selbst, als Behörde betrachtet) auf sie aus. Abb. 1 zeigt die Einbettung der Didaktik der Informatik in einen Kreis unterschiedlicher Wissenschaften und Institutionen sowie eine typische Fragestellung, die in dem jeweiligen Bereich bearbeitet wird und dessen zugehörige Lösung die Didaktik der Informatik zum Aufbau ihrer Lehre verwendet.

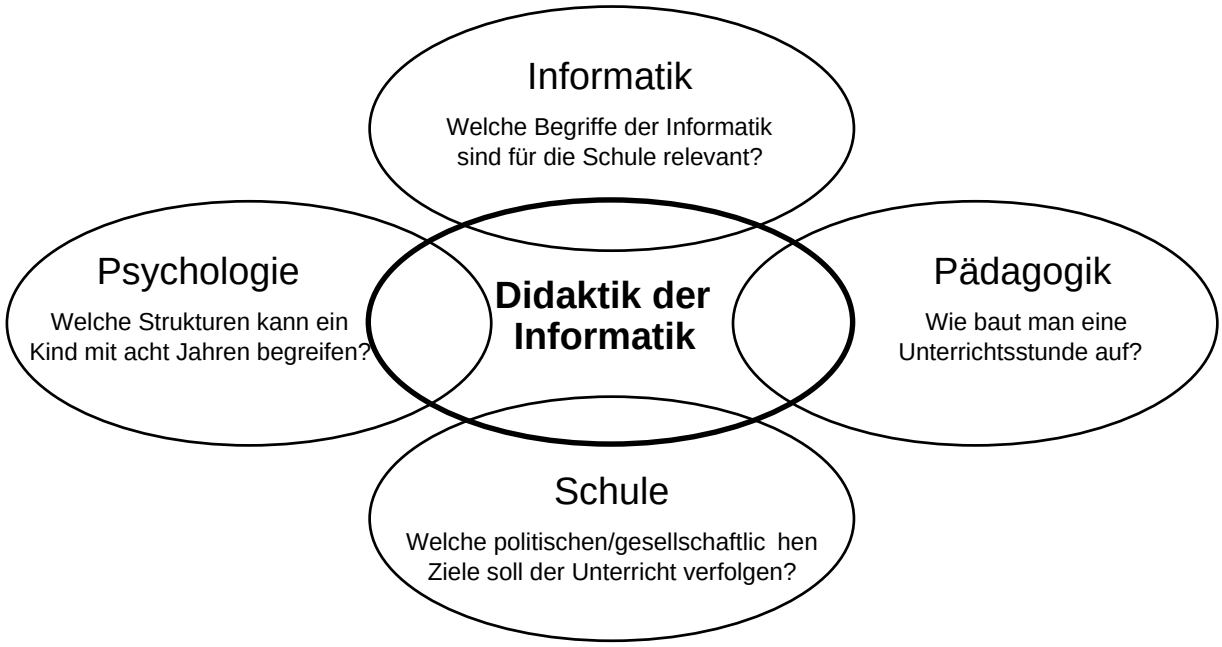


Abb. 1: Einbettung der Didaktik der Informatik

Auf einen interessanten Zusammenhang zwischen Didaktik und Informatik weist R. Baumann hin (Tab. 1): Definiert man wie oben Didaktik als die Wissenschaft von der Methode des Unterrichtens, so bezieht man sich hierbei traditionell implizit auf einen menschlichen Lehrer und menschliche Schüler. Formal argumentiert ist nach dieser Definition aber auch die Informatik selbst eine Form von Didaktik, nur sind die Schüler hier Computer. Sie werden „unterrichtet“, gewisse Probleme möglichst gut zu lösen. Den Lehrer nennt man dann Software-Technologen, Systemanalytiker, Wissensingenieur o.ä.; er erforscht Methoden, um Computern gewisse Inhalte effektiver zu vermitteln. Noch komplizierter wird der Sachverhalt, wenn man einen Computer zur Unterstützung des traditionellen Unterrichts heranzieht (Stichwort: computerunterstützter Unterricht, CUU). Nun ist der Computer ein didaktisch gebildeter Lehrer, und seine Schüler sind Menschen, womit wir wieder am Ausgangspunkt der Kette angelangt sind.

erteilt Unterricht und ist didaktisch vorgebildet	wird unterrichtet	Methodische Bezeichnung
Lehrer	Schüler	traditioneller Unterricht
Informatiker	Computer	Programmierung
Computer	Schüler	CUU

Tab. 1: Verhältnis zwischen Didaktik und Informatik

2 Unterrichtsziele und didaktische Ansätze

Offenbar richten sich die Inhalte eines Fachunterrichts stets auch nach seinen Zielen, und mit den Zielen ändern sich jeweils auch die didaktischen Ansätze, nach denen der Unterricht zu organisieren ist. So ist z.B. Informatik in der Oberstufe gänzlich anders zu unterrichten, wenn es als eigenständiges Fach eingerichtet ist, als wenn der Informatikunterricht in andere Fächern z.B. die Mathematik einbezogen ist.

Eine detaillierte Klassifikation von prinzipiell möglichen und tatsächlichen Zielen eines Informatikunterrichts zusammen mit den zugehörigen didaktischen Ansätzen hat V. Claus 1991 aufgestellt. Wir greifen die wichtigsten und interessantesten Kombinationen von Unterrichtsziel und Didaktikansatz (in leicht geänderter Form) heraus und erläutern sie etwas ausführlicher (Tab. 1).

Unterrichtsziel	Didaktikansatz
a. Wissenschaftsbezogener Bereich	
a1. Einbettung der Informatik in übergeordnete Begriffe	universeller Ansatz
a2. Fundamentale Ideen der Informatik	ideenorientiert
a3. Unterliegende Technik- und Hilfsbereiche	hardwareorientiert, logikorientiert, numer. Ansatz
a4. Gesicherte Aussagen, Systematik, Weltbild	theorie-orientiert oder informatischer Ansatz
b. Zukunftsaspekte	
b1. Visionen über Möglichkeiten der Informatik	visionsorientierter Ansatz
b2. Entwicklungslinien der Informatik	Extrapolationsansatz

c. Einsatz, Anwendungen, Auswirkungen bis heute		
	c1. Typische Anwendungen zur Zeit	anwendungsorientiert
	c2. Auswirkungen (Arbeit, Soziales, Freizeit usw.)	sozialorientiert
d. Nutzen (und Schaden)		
	d1. Nutzen für den einzelnen	joborientiert
	d2. Nutzen für die Arbeitswelt	arbeitswelt-orientiert
	d3. Nutzen für die Allgemeinheit	gesellschaftsorientiert
e. Grundtechniken und Unterstützungscharakter		
	e1. Computer als Werkzeuge in anderen Fächern	integrativer Ansatz
	e2. Informatik als wichtige Kulturtechnik	kulturorientiert

Tab. 1: Unterrichtsziele und zugehörige Didaktikansätze

Die Paare (Ziel, Ansatz) sind relativ scharf gegeneinander abgegrenzt und betonen jeweils nur einen kleinen Ausschnitt aus der Menge der möglichen Zugänge zur Informatik. Es handelt sich gewissermaßen um „Reinkulturen“, die man im praktischen Unterricht jedoch nicht in dieser reinen Form verwenden kann. Vielmehr muß man Kombinationen mehrerer Unterrichtsziele auswählen, um unterschiedliche Aspekte der Informatik zu vermitteln. Entsprechende Mischformen von Didaktikansätzen kommen bereits in der Praxis vor.

Die Ansätze im einzelnen:

- universeller Ansatz:
  Die Informatik (oder Teile davon) wird in einen übergeordneten Zusammenhang gestellt. Möglich ist z.B. ein Unterricht allgemein über informationsverarbeitende Systeme in Natur, Wirtschaft, Wissenschaft und Gesellschaft oder über natürliche und künstliche Sprachen, deren Syntax, Semantik und Pragmatik, sowie darüber, was mit unterschiedlichen Sprachbegriffen darstellbar ist.
- ideenorientierter Ansatz:
  Nach Bruner soll sich der Unterricht eines Faches in erster Linie an den „fundamentalen Ideen“ der zugrundeliegenden Wissenschaft orientieren. Da den fundamentalen Ideen eine längerfristige Gültigkeit zugeschrieben wird, kann es einem Informatikunterricht nach diesem Ansatz am ehesten gelingen, dem Druck der Wissenschaft, der durch die Halbwertszeit des Informatikwissens aufgebaut wird, zu entgehen. Allerdings gibt es kaum Untersuchungen darüber, was fundamentale Ideen der Informatik sind.
- logik-, hardwareorientierter, numerischer Ansatz:
  logikorientiert: Mit Logikkalkülen lassen sich viele Aspekte von Maschinen und Programmen eindeutig spezifizieren, beschreiben und in gewissem Umfang auch verifizieren.

Im Hochschulbereich setzt sich dieser Ansatz mehr und mehr durch.
Historisch hat sich die Informatik u.a. aus der Elektrotechnik und der Mathematik entwickelt. Entsprechende Didaktikansätze sind jedoch veraltet.
hardwareorientiert: s. Beispiel unten.
numerisch: Im Mittelpunkt stehen numerische Berechnungen (z.B. die Wurzelberechnung nach Heron). Weitere Anwendungen werden nicht behandelt.

- informatischer Ansatz:
  Informatische Inhalte und Methoden werden systematisch in einer Form dargestellt, wie sie sich in grundlegenden Vorlesungen, Informatiklehrbüchern und Lexika finden. Dieser Ansatz bereitet am ehesten auf ein Informatikstudium vor.
- visionsorientierter Ansatz:
  Ausgehend von einer bestimmten Zukunftsvorstellung über Arbeitsweise, Leistungsfähigkeit und Anwendungsbereich von Computersystemen versucht man, ausgewählte Teilaspekte dieser Vision zu programmieren oder zu simulieren. Gleichzeitig entwickelt man die notwendigen informatischen Grundkenntnisse hierzu, etwa zur prinzipiellen Leistungsfähigkeit von Rechnern (Berechenbarkeit), zur Spracherkennung usw.
- Extrapolationsansatz:
  Hier geht man umgekehrt zum visionsorientierten Ansatz vor: Ausgehend von den aktuellen Gegebenheiten versucht man die Entwicklung der nächsten 10 bis 15 Jahre vorzuzeichnen und vermittelt die entsprechenden Inhalte im Unterricht. Zu diesen Inhalten könnten z.B. Parallellismen aller Art gehören: Parallelrechner, Netze, Sprachen für Parallelrechner usw.
- anwendungsorientierter Ansatz:
  Welche aktuellen Anwendungen und Anwendungsmöglichkeiten bestehen für die Informatik und für Computer zur Zeit? Ausgehend von dieser Fragestellung analysiert man die in den entsprechenden Bereichen vorkommenden Systeme hinsichtlich der verwendeten Informatikmethoden und -konzepte.
- sozialorientierter Ansatz:
  Schwerpunkt dieses Didaktikansatzes sind die persönlichen und gesellschaftlichen Auswirkungen der Informatik.
- joborientierter Ansatz:
  Es ist unbestritten, daß Personen mit Informatikkenntnissen bessere Berufschancen besitzen. Die kurzfristigen Chancen erhöhen sich meist deutlich für diejenigen, die ein bestimmtes marktgängiges Produkt (Computersystem, Betriebssystem, Programmiersprache, Textsystem) beherrschen. Diese Tatsache greift der joborientierte Ansatz auf. Er vermittelt *eine* Programmiersprache, *ein* Betriebssystem, *ein* Textsystem usw. Grundprinzipien der Informatik treten in den Hintergrund.
- arbeitswelt-orientierter Ansatz:
  Unterschiedliche Systeme werden erläutert, analysiert und hinsichtlich ihres Nutzens und Schadens für Benutzer, Firmen, die Wirtschaft klassifiziert.

- gesellschaftsorientierter Ansatz:
  Hier bilden weniger Einzelpersonen oder Firmen sondern die Gesellschaft den Schwerpunkt, für den Nutzen und Schaden klassifiziert werden.
- integrativer Ansatz:
  In vielen Schulfächern werden bereits Computer als Unterrichtsmittel eingesetzt. Der integrative Ansatz verfolgt nun das Ziel, zugleich mit den Anwendungen des Computers auch in die Grundprinzipien der Informatik einzuführen.
- kulturorientierter Ansatz:
  Lernziel dieses Ansatzes ist die **computer literacy**.

Beispiele:

1) Der veraltete (bis etwa 1976 in den Schulen vertretene) **hardware-orientierte Ansatz** ist eine Kombination von a3, a4 und c1 mit Schwerpunkt auf a3. Bei diesem Ansatz wird die Informatik als technische Disziplin gesehen, deren Forschungsgegenstand der Computer (als technisches Gerät betrachtet) ist. Die schaltalgebraischen Grundlagen, ihre Realisierung sowie die prinzipielle Arbeitsweise von Computern stehen im Vordergrund. Algorithmen werden mit Flußdiagrammen dargestellt und in BASIC oder Maschinen- und Assembler-Sprachen formuliert und auf Mikroprozessoren oder Modellrechnern ausgeführt. Höhere Programmiersprachen werden nur vereinzelt verwendet. Es überwiegen Anwendungen aus der Mathematik (Numerik).
   Beispiel für ein Lehrbuch, das dieses didaktische Prinzip verfolgt, ist: K. Flensberg, I. Zeising: „Praktische Informatik“, Bayerischer Schulbuch-Verlag 1974.
2) Zur Zeit überwiegt in der Schule der **algorithmen-orientierte Ansatz**. Hierbei wird die Informatik als Wissenschaft gesehen, die Methoden zum Entwurf und zur Spezifikation von Algorithmen bereitstellt, die Programmierung unterstützt und Techniken zur Darstellung und Realisierung von Problemlösungen sowie zur Analyse und Verifikation von Programmen erarbeitet. Höhere Programmiersprachen dominieren. Hardwarebezogene Konzepte treten in den Hintergrund, weil damit zu rechnen ist, daß diese relativ schnell veralten. Der Computer wird also weniger als Forschungsobjekt denn als Werkzeug betrachtet.
   Der algorithmen-orientierte Ansatz kann als Kombination der Ansätze a2, a4, b2 und c aufgefaßt werden, wobei das Gewicht auf a4 liegt.
   Beispiel für ein Lehrbuch, in dem der algorithmen-orientierte Ansatz verfolgt wird, ist Metzler Informatik, Metzler Verlag 1984.
   Mittlerweile scheint auch diese Methodik überholt. Ihr Nachteil besteht darin, daß der Begriff des Algorithmus überwiegend im imperativen Sinne interpretiert wird und die anderen Darstellungen (etwa prädikativ, funktional) vernachlässigt werden. Ferner werden die Grenzen der Algorithmisierbarkeit im theoretischen (Berechenbarkeit, Verifikation), im praktischen (Entwicklung unüberschaubar großer Programme) und im gesellschaftlichen Sinne (Sicherheit von Softwaresystemen) nicht hinreichend herausgestellt.

Grob gesprochen betont dieser Ansatz die Machbarkeit gegenüber der Nicht-Machbarkeit.

Generell scheint sich derzeit eine Schwerpunktverlagerung hin zu den Ansätzen a1 und a2 abzuzeichnen, wobei bei a1 die Informatik allgemein als Lehre von den informationsverarbeitenden Systemen angesehen wird.

3) Der zur Zeit in die Sekundarstufe I eingebrachte Unterricht zur **ITG** (informations- und kommunikationstechnische oder -technologische Grundbildung) umfaßt die Ziele und Ansätze a4, c, d und e2 mit Betonung auf c und d.

Für den Informatikunterricht in der Sekundarstufe II muß der Schwerpunkt auf einem der Ansätze a1 - a4 gelegt werden. Wir bevorzugen einen Ansatz, der zwar noch kaum verbreitet ist, jedoch für den Informatikunterricht besonders geeignet erscheint: den ideenorientierten Ansatz a2. Die wesentlichen Prinzipien dieses Ansatzes enthält der folgende Abschnitt.

TEIL C

Ideenorientierter Informatikunterricht - Theorie

1 Einleitung

Trotz ihrer nun mehr als 40-jährigen Entwicklungsgeschichte besitzt die Informatik noch immer eine erstaunliche Dynamik. Regelmäßig kündigen sich Paradigmenwechsel an, etwa in der Programmierung von der „straight forward“-Methode zum strukturierten Programmieren nach der Softwarekrise in den 60er Jahren, später dann zum logischen und applikativen Programmieren und zur Zeit zur objektorientierten Programmierung.

Unter den wenigen Personen, die sich forschend mit dem Themenbereich „Informatik in der Schule“ befassen, herrscht allgemein Konsens, daß die Fortschritte der Wissenschaft Informatik nicht mit gleicher Geschwindigkeit für den Schulunterricht zugänglich gemacht werden können. Daher müssen sich die Inhalte im Informatikunterricht bis auf weiteres an den langlebigen Grundlagen der Wissenschaft orientieren. Es ist unverzichtbar, daß den Schülern ein Bild von den *grundlegenden Prinzipien*, *Denkweisen* und *Methoden* (den **fundamentalen Ideen**) der Informatik vermittelt wird. Nur von diesen Ideen ist eine längerfristige Gültigkeit zu erwarten. Neuere Erkenntnisse erscheinen dann häufig nur als Variation eines bereits vertrauten Sachverhalts und können über die einmal gelernten Ideen leichter erschlossen werden. Diese Leitlinien wurden fachneutral erstmals von J.S. Bruner in „Der Prozeß der Erziehung“, Berlin 1970 propagiert.

2 Fundamentale Ideen als Unterrichtsprinzip

Bereits 1929 schlug A.N. Whitehead vor,

> „sich offenkundig auf unmittelbare und einfache Weise mit einigen wenigen allgemeinen Ideen von weitreichender Bedeutung“

zu befassen, denn:

> „Die Schüler stehen ratlos vor einer Unmenge von Einzelheiten, die weder zu großen Ideen noch zu alltäglichem Denken eine Beziehung erkennen lassen.“

Bruner hat diese Überlegungen später aufgegriffen und zu einer Theorie entwickelt.

2.1 Hintergründe und Motivation

Bruner begründet seinen Zugang wie folgt: Lernen in der Schule hat vor allem den Sinn, uns zu präparieren, das zukünftige Leben erfolgreicher zu bewältigen. Da kontrolliertes Lernen nach Anleitung - sieht man mal von der beruflichen Fort- und Weiterbildung ab - mit dem letzten Schuljahr oder Semester weitgehend abgeschlossen ist, können später eintretende Veränderungen im Privatleben, in Wirtschaft und Gesellschaft nur durch Übertragung (**Transfer**) früher erworbener Kenntnisse auf die neuen Situationen gemeistert werden.

Man kann diese Transferleistung vor allem hinsichtlich zweier Aspekte klassifizieren:

- Ähnelt die neue Situation einer bereits bekannten, so daß deren Lösungsschema nur geringfügig erweitert oder geändert werden muß, um auch auf die neue Situation anwendbar zu sein, so spricht man vom **spezifischen Transfer**. Spezifische Transferleistungen beziehen sich auf relativ lokale Effekte und werden überwiegend dann gefordert, wenn es um die kurzfristige Anwendung handwerklicher Fertigkeiten innerhalb eines begrenzten Fachgebiets geht.
- Beim **nichtspezifischen Transfer**, der sich auf langfristige (i.a. lebenslange) Effekte bezieht, lernt man anstelle von oder zusätzlich zu handwerklichen Fertigkeiten grundlegende Begriffe, Prinzipien und Denkweisen (sog. **fundamentale Ideen**). Ferner bildet man Grundhaltungen und Einstellungen aus, z.B. zum Lernen selbst, zum Forschen, zur Wissenschaft, zu Vermutungen, Heuristiken und Beobachtungen, zur eigenen Leistung, zur Lösbarkeit von Problemen usw. Später auftretende Probleme können dann gewissermaßen als Spezialfälle dieser Grundkonzepte erkannt, eingeordnet und mit den zugehörigen Lösungsverfahren in transferierter Form behandelt werden. Während der spezifische Transfer also etwas Neues direkt auf etwas Bekanntes derselben logischen Ebene zurückführt, bezieht der nichtspezifische Transfer eine Metaebene ein (Abb. 1).

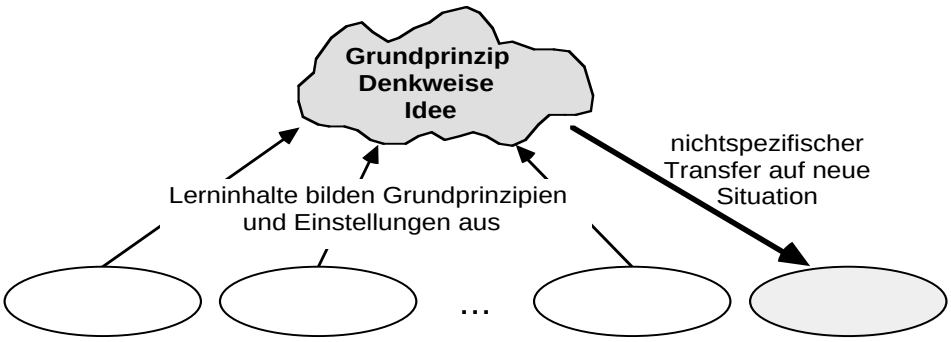


Abb. 1: Nichtspezifischer Transfer

In Berufsschulen und in der betrieblichen Fort- und Weiterbildung dominiert der spezifische Transfer. Die vermittelten Fertigkeiten werden überwiegend nicht in einer Weise behandelt, die bei den Lernenden fundamentale Ideen ausbildet. Diese Fertigkeiten können daher i.a. nur durch spezifischen Transfer auf neue Probleme und Situationen übertragen werden. Dem gegenüber steht der nichtspezifische Transfer im Zentrum des gesamten Bildungsprozesses an allgemeinbildenden Schulen: Fortwährendes Erzeugen, Erweitern und Vertiefen von Wissen in Form fundamentaler Ideen. Die vermittelten Kenntnisse haben nicht in erster Linie den Zweck, unmittelbar angewendet werden zu können.

Daher - und jetzt kommen wir wieder auf die Ausgangsforderung Bruner’s zurück - hat sich der Unterricht vor allem an den fundamentalen Ideen zu orientieren. Jeder Unter-

richtsgegenstand ist daraufhin zu überprüfen, welche Ideen ihm zugrundeliegen. Alle Lehrpläne und Unterrichtsmethoden sind darauf abzustellen, bei jedem Unterrichtsgegenstand die fundamentalen Ideen hervorzuheben.
Natürlich werfen die letzten Forderungen eine Fülle von Fragen auf: Was sind überhaupt fundamentale Ideen? Wie sind Curricula umzustellen, damit fundamentale Ideen eine zentrale Rolle erhalten? Welche Themen eignen sich auf den verschiedenen Schulstufen am besten zur Vermittlung von fundamentalen Ideen? Einigen Fragen werden wir im folgenden auf den Grund gehen, zuerst allgemein und in Abschnitt 3 dann informatikbezogen.

Zunächst ein Beispiel aus der Informatik, das zeigt, wie bedeutsam die Forderung Bruner's ist, den Unterricht nach Ideen auszurichten und wie häufig gegen dieses Prinzip in der Praxis verstoßen wird.

Beispiel: In einem Informatiklehrbuch für die Schule wird die Parameterübergabeart call by reference in Prozeduren eingeführt. Der Mechanismus, der beim Aufruf einer Prozedur mit Referenzparametern abläuft, wird dort auf etwa einer Seite u.a. am Behältermodell wie folgt erläutert:

> „Der Kasten der aufrufenden Variablen wird mit in den Prozedurraum genommen und erhält während der Prozedurbearbeitung einen anderen Namen, den Namen des Parameters. Nach Abschluß der Bearbeitung wird der aufgeklebte Name wieder entfernt, der Kasten wird unter seinem ursprünglichen Namen in den Hauptraum zurückgenommen. Damit sind jetzt im Kasten die Werte enthalten, die während der Prozedurbearbeitung hineingelegt wurden."

Diese recht verwirrenden und zum Teil falschen Erklärungen überdecken vollständig die Idee, die hinter dem Konzept call by reference steckt:

Vergabe eines Synonyms für ein Objekt.

Der Bezeichner des formalen Parameters wird innerhalb der Prozedur als Synonym für den aktuellen Parameter verwendet. Diese sehr einfache wie treffende Idee beschreibt das Prinzip vollständig. Die ursprüngliche Flut von Informationen wird zu einer griffigen „Formel" *verdichtet*, die im Gedächtnis haften bleibt, auch weil sie im Gegensatz zur obigen Erläuterung noch einen sprachlichen *Bezug zur Lebenswelt* besitzt.

2.2 Fundamentale Idee: Begriffspräzisierung

Die Gretchenfrage, die hinter den bisherigen Überlegungen stand, lautet: Was sind fundamentale Ideen? Bruner selbst weicht aus, wenn es um eine explizite Definition oder Charakterisierung des Begriffs geht. Spätere Arbeiten anderer Autoren besitzen hier mehr Substanz. Bruner überläßt es im wesentlichen dem Leser, sich anhand vieler Beispiele eine intuitive Vorstellung von fundamentalen Ideen zu verschaffen. Lediglich einige wenige, aber wichtige Hinweise sind über das Buch verstreut.

Bezogen auf die Informatik definieren wir eine fundamentale Idee wie folgt:

„Definition“:
Eine **fundamentale Idee** der Informatik ist ein Denk-, Handlungs-, Beschreibungs- oder Erklärungsschema, das vier Kriterien erfüllt - das Horizontalkriterium, das Vertikalkriterium, das Sinnkriterium und das Zeitkriterium -, die wir im folgenden der Reihe nach beschreiben.

Das Horizontalkriterium. Eine fundamentale Idee ist in verschiedenen Bereichen der Informatik vielfältig anwendbar oder erkennbar ist.
Fundamentale Ideen besitzen also eine umfassende Anwendbarkeit in vielen Bereichen, und sie ordnen und integrieren eine Vielzahl von Phänomenen. Durch den Begriff „Horizontalkriterium“ ist folgende Anschauung verbunden: Eine horizontale Achse, die eine Vielzahl von Wirkungsbereichen durchstößt (Abb. 2).

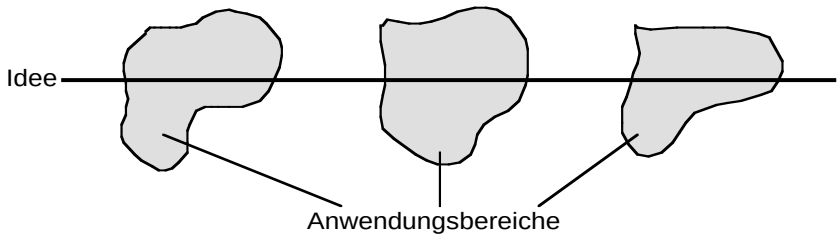


Abb. 2: Veranschaulichung des Horizontalkriteriums

Das Vertikalkriterium. Eine fundamentale Idee kann auf jedem intellektuellen Niveau aufgezeigt und vermittelt werden.
Bruner vermutet in seiner berühmte These

> „... daß die Grundlagen eines jeden Faches jedem Menschen in jedem Alter in irgendeiner Form beigebracht werden können.“.

Dies legt nahe, daß fundamentale Ideen den Stoff innerhalb eines Anwendungsbereiches auch vertikal strukturieren: Eine fundamentale Idee kann auf nahezu jeder beliebigen geistigen Ebene (also einem Primarschüler ebenso wie einem Hochschüler) erfolgreich vermittelt werden. Unterschiede bestehen auf den verschiedenen Ebenen nur hinsichtlich des Niveaus sowie dem Grad der Detaillierung und Formalisierung, mit dem die Idee präsentiert wird. Kontrapositiv formuliert: Was nicht prinzipiell auch einem Volksschüler vermittelt werden kann, kann keine fundamentale Idee sein.
Anschaulich kann man den Anwendungsbereich einer fundamentalen Idee in verschiedene Ebenen mit wachsenden geistigen Anforderungen unterteilen. Eine fundamentale Idee ist dann durch eine vertikale Achse repräsentiert, die jede der Ebenen schneidet (Abb. 3).
Für den Schulunterricht ist das Vertikalkriterium von besonderer Bedeutung: Erfüllt eine Idee dieses Kriterium, so kann sie als Richtschnur verwendet werden, um den Unterricht auf jeder Ebene des gesamten Bildungsprozesses daran zu orientieren. Wir kommen hierauf in Zusammenhang mit dem Spiralprinzip später noch einmal zurück.

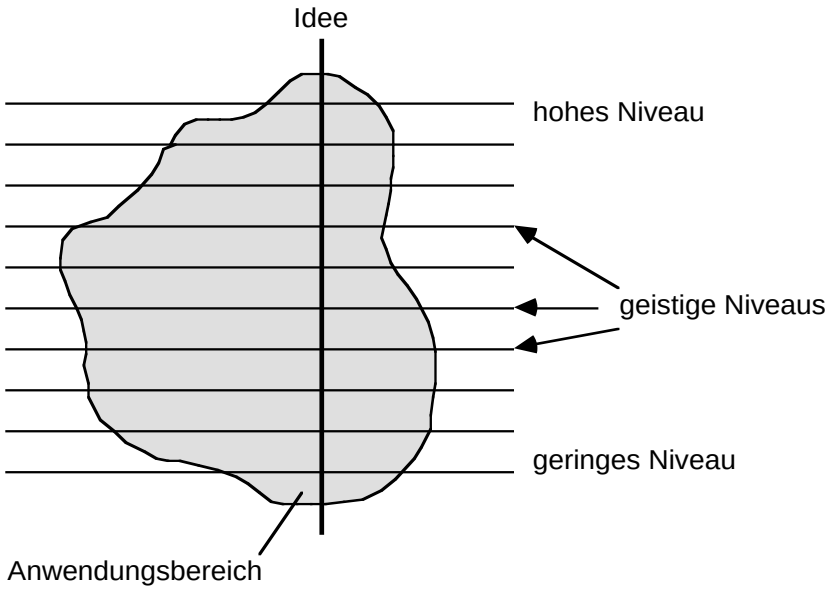


Abb. 3: Veranschaulichung des Vertikalkriteriums

Das Zeitkriterium. Eine fundamentale Idee ist in der historischen Entwicklung der Informatik deutlich wahrnehmbar und bleibt längerfristig relevant.
Dieser historische Aspekt ist aus zwei Gründen bedeutsam. Einerseits liefert er einen Anhaltspunkt, wie fundamentale Ideen zu gewinnen sind: Durch Beobachtung der geschichtlichen Entwicklung fachwissenschaftlicher Begriffe, Konzepte und Strukturen. Andererseits wird hiermit angedeutet, daß fundamentale Ideen einer Wissenschaft längerfristig gültig bleiben.
Das Sinnkriterium. Eine fundamentale Idee besitzt eine Verankerung im Alltagsdenken und eine lebensweltliche Bedeutung.
Fundamentale Ideen zählen zur Sphäre des Alltagsdenkens. Ihr Kontext ist vortheoretisch, noch unwissenschaftlich. Die Präzisierung zu einem exakten Begriff steht einer Idee noch bevor.
Beispiel: Das skizzierte Verhältnis zwischen einer Idee „mit Sinn“ und einem Begriff besteht z.B. zwischen „Reversibilität“ als Idee und „Umkehrfunktion“ als Begriff. Während „Umkehrfunktion“ ein rein mathematischer Terminus ohne Bezug zur Alltagswelt ist, läßt sich Reversibilität an vielen Stellen im täglichen Leben aufzeigen.

2.3 Wie vermittelt man fundamentale Ideen?

Den entscheidenden Beitrag liefert hier Bruner selbst durch die Forderung, ein Unterricht, der sich an fundamentalen Ideen einer Wissenschaft orientiert, müsse nach dem **Spiral-**

prinzip organisiert werden. Er beschreibt dieses Prinzip so:

> „Der Anfangsunterricht ... sollte so angelegt sein, daß diese Fächer mit unbedingter intellektueller Redlichkeit gelehrt werden, aber mit dem Nachdruck auf dem intuitiven Erfassen und Gebrauchen dieser grundlegenden Ideen. Das Curriculum sollte bei seinem Verlauf wiederholt auf diese Grundbegriffe zurückkommen und auf ihnen aufbauen, bis der Schüler den ganzen formalen Apparat, der mit ihnen einhergeht, begriffen hat. ... Man muß noch viel über die ‚Curriculum-Spirale' lernen, die auf höheren Ebenen immer wieder zu sich selbst zurückkommt." (S. 26/27).

Analysiert man die einzelnen Passagen dieser Beschreibung genauer, so entdeckt man drei Teilprinzipien, die gewissermaßen die tragenden Säulen des Spiralprinzips bilden:

- **Prinzip der Fortsetzbarkeit**. Die Auswahl und die Behandlung eines Themas an einer bestimmten Stelle des Curriculums soll nicht ad hoc, sondern so erfolgen, daß auf höherem Niveau ein Ausbau möglich ist. Zu vermeiden sind vordergründige didaktische Lösungen (auch durch Vermittlung von Halbwahrheiten), die später ein Umdenken und eine teilweise Zurücknahme von Aussagen erforderlich machen.

  Beispiel: Flußdiagramme erscheinen (mehr noch als Struktogramme) im Anfangsunterricht der Informatik besonders geeignet, um Algorithmen darzustellen und zu entwikkeln. Sie verfügen über sehr wenige Darstellungsmittel und werden leicht begriffen. Schnelle Anfangserfolge im Unterricht sind garantiert. Der Nachteil dieses Ansatzes zeigt sich erst später: Er ist nicht ausbaufähig und zwingt zum Umdenken. Denn da in Flußdiagrammen keine Beschreibungsmittel für Schleifen vorgesehen sind, gibt es kein adäquates Verfahren, um von Flußdiagrammen zu strukturierten Programmen (z.B. in PASCAL) zu gelangen. Bereits entwickelte Programme in Flußdiagrammdarstellung müssen umgeschrieben werden.
- **Prinzip der Präfiguration von Begriffen**. Die *symbolische* Darstellung eines Begriffs oder Konzepts und seine begrifflich-strukturelle Analyse sollen bereits auf niedriger Stufe durch Bilder (*ikonisch*) und Handlungen (*enaktiv*) vorbereitet werden. Bevor also ein Begriff in allen Einzelheiten und Auswirkungen theoretisch analysiert werden kann, sollte er zunächst in Gebrauch genommen werden, damit die Schüler eine intuitive Vorstellung erhalten.

  Beispiele:

  1) Statt die Syntax einer Programmiersprache oder ihrer Sprachelemente jeweils explizit einzuführen, kann man die einzelnen Konstrukte zunächst „einfach benutzen". An Beispielen wird die Syntax durch Analogschluß klar.
  2) Man kann das Prozedurkonzept an einer bestimmten Stelle des Curriculums explizit einführen. Alternativ verwendet man vorher bereits implizit Standardprozeduren und -funktionen. So entsteht bei den Schülern bereits eine Vorstellung von den Abläufen bei Aufruf und Parameterübergabe.
- **Prinzip der vorwegnehmenden Lernens**. Die Behandlung eines Wissensgebietes soll nicht aufgeschoben werden, bis eine endgültig-abschließende Behandlung möglich erscheint, sondern ist bereits auf früheren Stufen in einfacher Form einzuleiten.

  Beispiel: Innerhalb der Informatik benötigt man zur Verifikation von Programmen einen

relativ hohen technischen Aufwand, den man in der Schule allenfalls in den letzten Klassen der Oberstufe zur Verfügung hat. Zuvor kann und sollte man jedoch schon auf einfacher Stufe Grundzüge vermitteln, z.B.

= durch Plausibilitätsbetrachtungen zur Termination von Schleifen auf argumentativer oder halbformaler Ebene.
= durch Aufzeigen des Zusammenhangs zwischen Verifikation und Testen.
= durch ausgewählte kleine Programme, deren Semantik nicht unmittelbar ersichtlich ist, aber durch Testen und Nachdenken schließlich exakt ermittelt werden kann.

2.4 Vorteile des Brunerschen Konzepts

- Der Unterrichtsgegenstand wird faßlicher, wenn man seine Grundlagen versteht.
- Fundamentale Ideen schaffen Beziehungsnetze. Sie prägen Details, die man unzusammenhängend schnell wieder vergessen würde, eine verbindende Struktur auf. Über sie können Einzelheiten wieder rekonstruiert werden (Verdichtung von Information).
- Fundamentale Ideen unterstützen den nichtspezifischen Transfer. Durch ihre Allgemeinheit können viele Fälle als Spezialfälle behandelt werden.
- Da fundamentale Ideen den Stoff vertikal gliedern, verringern sie den Abstand zwischen elementarem und fortgeschrittenem Wissen. Dahinter verbirgt sich die Überzeugung Bruner's

  > „... daß geistige Tätigkeit überall dieselbe ist, an den Fronten des Wissens ebenso wie in einer dritten Klasse. ... Der Unterschied liegt im Niveau, nicht in der Art der Tätigkeit.".

  Die Tätigkeiten eines Wissenschaftlers und eines Volksschülers unterscheiden sich also nicht prinzipiell, da beide die gleichen fundamentalen Ideen, jedoch auf unterschiedlichem Niveau, verwenden.
- Ein Vorteil, der sich im Kurssystem auszahlen kann: Da die fundamentalen Ideen im Zentrum des Unterrichts stehen, kann man nahezu an jeder Stelle abbrechen, ohne die Schüler mit einem sinnlosen Fragment alleine zu lassen.
- Während fundamentale Ideen eine längerfristige Gültigkeit besitzen (vgl. Zeitkriterium), sind Details schnell wieder veraltet. Dies trifft vor allem auf die Informatik zu, so daß das Brunersche Konzept hier besonders schlagkräftig erscheint.
- Es gibt bisher keine Philosophie der Informatik. Hier kann eine Zusammenstellung von fundamentalen Ideen als Einstieg dienen und helfen, das „Wesen" von Informatik zu klären und sie gegen andere Wissenschaften abzugrenzen.

3 Fundamentale Ideen der Informatik

Algorithmisierung.

Über die Bedeutung der Algorithmisierung als fundamentale Idee der Informatik besteht weitgehend Konsens. Eine genauere Analyse der Aktivitäten beim Algorithmisieren liefert noch eine Fülle weiterer Ideen. Wir unterscheiden vier große Bereiche:

- Beim Entwurf eines Algorithmus bedient man sich häufig der oben schon erwähnten

Grundmuster, sog. Entwurfsparadigmen, wie Divide-and-Conquer, Backtracking.
- Diesen Entwurf setzt man anschließend in ein Programm um. Hierfür stehen gewisse grundlegende sprachunabhängige Konzepte zur Verfügung, um den Daten- und Kontrollbereich zu beschreiben, wie z.B. Konkatenation, Alternative, Rekursion usw.
- Das fertige Programm wird auf einem (oder mehreren) Prozessoren ausgeführt. Damit verbunden ist die Idee des Prozesses, d.h. die Trennung von Beschreibung und Ausführung eines Algorithmus.

Die letzte Ideengruppe im Rahmen der Algorithmisierung befaßt sich mit der Bewertung von Algorithmen unter Qualitätsgesichtspunkten. Die beiden zentralen Bewertungskriterien sind Korrektheit und Komplexität, jeweils mit einer Reihe weiterer zugehöriger Ideen.

Sprache. Nicht nur in der Programmierung (Programmiersprachen), bei der Spezifi- kation (Spezifikationssprachen), bei der Verifikation (Logikkalküle), in Datenbanken (Anfragesprachen), bei Betriebssystemen (Kommandosprachen) spielt die Idee der Sprache eine herausragende Rolle, vielmehr besteht in der Informatik eine generelle Tendenz zur Versprachlichung von Sachverhalten. Dies gilt auch für Bereiche, bei denen zunächst kein unmittelbarer Bezug zu einer sprachlichen Darstellung besteht, z.B. beim VLSI-Entwurf oder beim Entwurf logischer Schaltungen. Dieses Vorgehen bietet u.a. folgenden Vorteil: Es vereinheitlicht die Sichtweise, denn jedes Problem reduziert sich dann auf ein Problem über Wörtern; die Manipulation von Sprachen und Wörtern ist andererseits wiederum gut erforscht.
Beispiel: Eine Rechnerarchitektur veranschaulicht man sich häufig durch ein Ebenenmodell (Abb. 4). Jeder Ebene wird ein anderes Sprachniveau zugeordnet, mit dem die Objekte und Aktionen der Ebene adäquat beschrieben werden können. Der Benutzer arbeitet üblicherweise auf der obersten Ebene. Jede seiner Aktionen wird schrittweise in das jeweils tiefer liegende Sprachniveau und schließlich auf die Hardware transformiert.

In engem Zusammenhang mit Sprachen stehen offenbar die Ideen von *Syntax* und *Semantik*, ferner die verschiedenen Ansätze zur semantikerhaltenden Transformation von Wörtern einer Sprache in eine andere, z.B. die Ideen der *Übersetzung*, *Interpretation* oder *operationalen Erweiterung*.

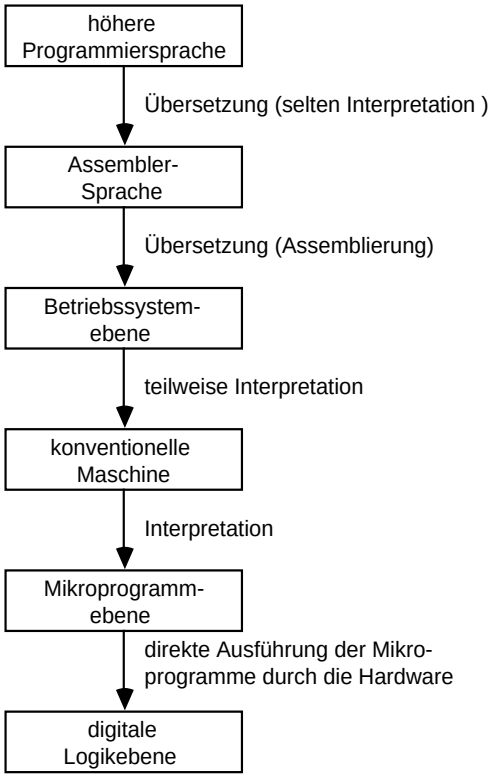


Abb. 4: Ebenenmodell der Rechnerarchitektur

Strukturierte Zerlegung. Die Zerlegung konkretisiert sich z.B

- bei der Istanalyse an der wiederholten Zerlegung des bestehenden Systems in Komponenten und an der Bildung von Teams,
- beim Entwurf an der hierarchischen Modularisierung,
- bei der Implementierung an der wiederholten Zerlegung von Abläufen in Einzelschritte (auch Algorithmisierung) oder an der Zerlegung eines Problems in einfachere (Divide-and-Conquer),
- beim Software life cycle selbst an der Zerlegung des Entwicklungsprozesses in sechs Phasen, die jeweils wiederum aus mehreren Unterphasen bestehen.

Offenbar können wir bei der Idee der Zerlegung einen horizontalen und einen vertikalen Aspekt unterscheiden, wobei die Hierarchisierung (Abb. 5) den vertikalen und die Modularisierung (Abb. 6) den horizontalen beschreibt. Die hierarchische Modularisierung entsteht dann als Mischform aus diesen beiden Grundformen (Abb. 7).

Die Idee der Hierarchisierung finden wir auch noch in vielen anderen Zusammenhängen: Ebenenmodelle der Rechnerarchitektur (s.o.), Sprachhierarchien (wichtigster Vertreter ist hier die Chomsky-Hierarchie), Maschinenmodelle, Komplexitäts- und Berechenbarkeitsklassen, virtuelle Maschinen, ISO-OSI-Ebenenmodell.

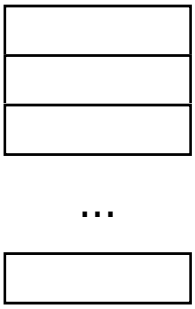


Abb. 5: Hierarchisierung

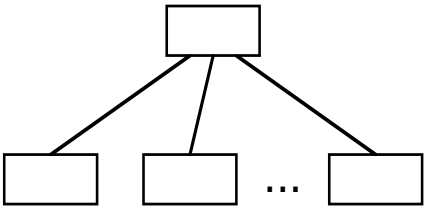


Abb. 6: Modularisierung

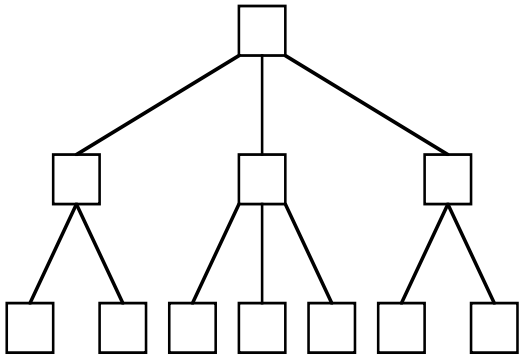


Abb. 7: Hierarchische Modularisierung

Einen weiteren wichtigen Aspekt der Zerlegung haben wir bisher noch nicht erwähnt: Offenbar kommt jeder Zerlegungsprozeß irgendwann zum Ende, spätestens dann, wenn ein atomares Niveau erreicht wird.
Beispiel: Beim Divide-and-Conquer-Verfahren bricht die Zerlegung ab, sobald ein Problem vorliegt, für das die Lösung unmittelbar angegeben werden kann.
Bei der hierarchischen Modularisierung stoppt sie spätestens dann, wenn eine Modulspezifikation erreicht wird, die sich direkt in eine einzelne Anweisung der Programmiersprache umsetzen läßt.

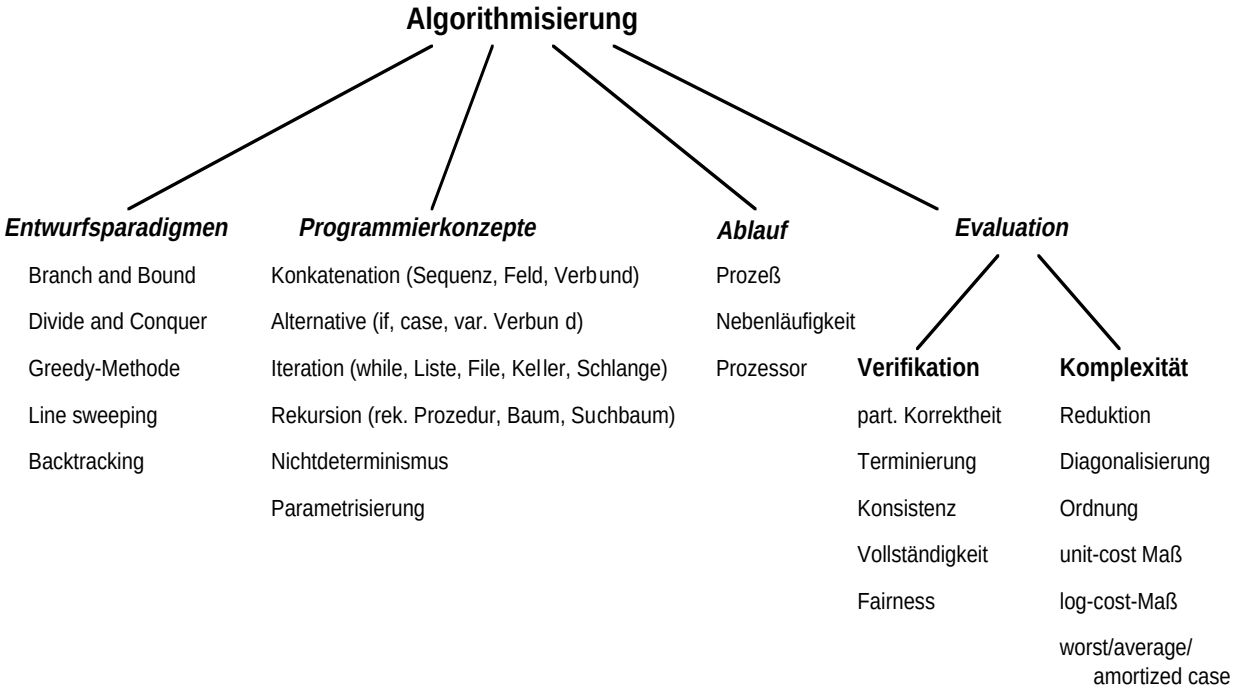
Diese Beobachtung führt auf die Idee der Erzeugendensysteme, wir wollen in Anlehnung an eine ähnliche Operation in der linearen Algebra von der Idee der *Orthogonalisierung* sprechen. Unter Orthogonalisierung eines Objektbereiches verstehen wir die Angabe einer endlichen Zahl von Basiselementen des Bereiches sowie von Operationen auf dieser Basis, so daß jedes beliebige andere Objekt des Bereichs durch endliche Anwendung der Operationen aus den Basiselementen erzeugt werden kann. Die Fundamentalität der Orthogonalisierung erkennt man an ihrer vielfältigen Anwendbarkeit innerhalb und außerhalb der Informatik (vgl. Horizontal- und Sinnkriterium), z.B. bei

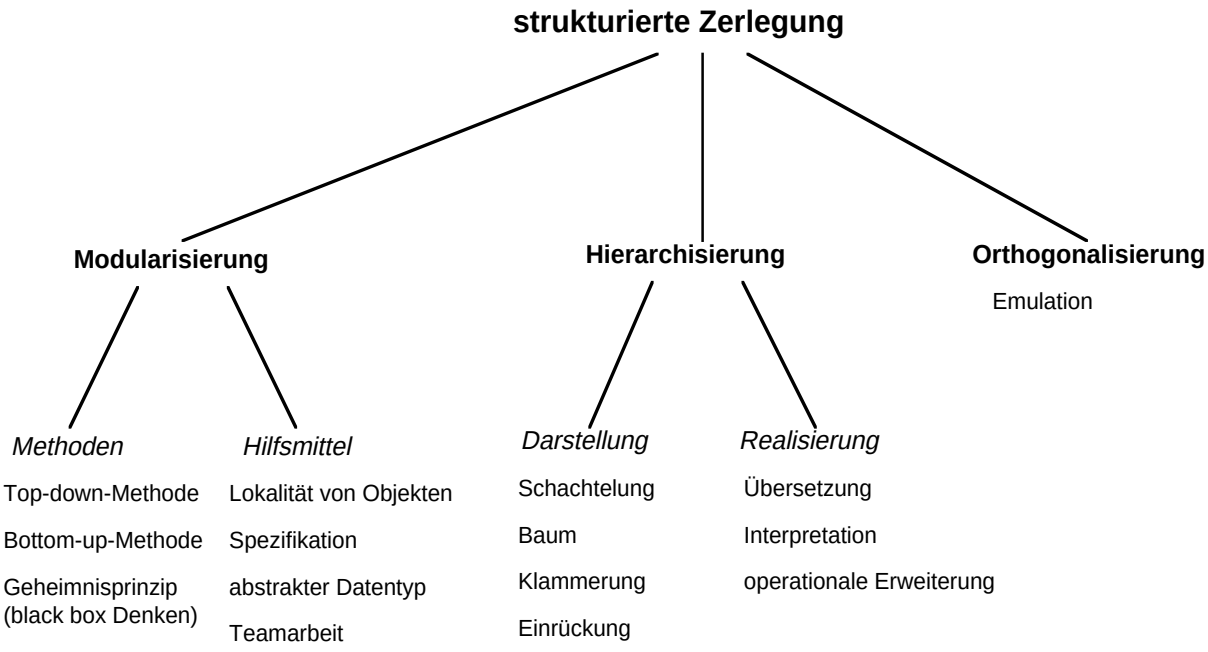
- Programmiersprachen. Zentrales Prinzip der Sprache ALGOL68 ist der Orthogonalentwurf: Es gibt einige wenige elementare Daten- und Anweisungstypen, die jedoch in beliebiger Weise kombiniert werden dürfen. Zum Beispiel sind in ALGOL68 Felder ebenso wie alle anderen Datentypen als Ergebnistypen von Funktionen zugelassen. PASCAL ist hier wesentlich restriktiver; von Orthogonalität ihrer Strukturen kann nicht gesprochen werden.
- imperativen Programmiersprachen. Die Strukturen Zuweisung, Konkatenation und while-Schleife bilden eine Basis der Kontrollstrukturen. Jede andere Anweisung läßt sich durch diese drei Typen darstellen.
- funktionalen Programmiersprachen. Die Grundoperationen bei Sprachen, die sich am

λ-Kalkül orientieren, sind Abstraktion (=Definition einer Funktion mit Parametern), Applikation (=Aufruf einer Funktion mit aktuellen Parametern) und Substitution (=Parameterersetzung).
- Maschinen. Die universelle Turingmaschine bildet die (einelementige) Basis der Klasse aller Turingmaschinen.
- primitiv-rekursiven und μ-rekursiven Funktionen. Es gibt eine Menge von Grundfunktionen (z.B. konstante Funktion 1, Nachfolgerfunktion) und eine Reihe von Operationen (z.B. Komposition, Einsetzung von Funktionen, μ-Operator), mit der man jede primitiv-rekursive bzw. μ-rekursive Funktion erzeugen kann.
- formalen Sprachen. Die Dyck-Sprache bildet gewissermaßen eine Basis der kontextfreien Sprachen (Satz von Chomsky-Schützenberger).
- booleschen Funktionen. UND, ODER und NICHT bilden eine Basis (einen sog. Bausteinsatz) für alle Booleschen Funktionen. NAND ist ebenfalls ein Bausteinsatz.
- der Fertigung von Automobilen im Baukastensystem,
- bei Schrankwänden oder Regalsystemen,
- bei Häusern aus Fertigteilen,
- bei der DNS, die aus vier Grundsubstanzen Adenin, Guanin, Cytosin und Thymin besteht.

Zum Nachweis der Nicht-Orthogonalität verwendet man üblicherweise die Idee der *Emulation:* Gegeben sei ein Erzeugendensystem. Kann man eines der Elemente des Systems durch die übrigen darstellen, so ist das System nicht orthogonal.

Nach diesen Vorüberlegungen können wir jetzt den vollständigen Katalog fundamentaler Ideen der Informatik aufstellen. Er enthält alle bisher genannten Ideen, thematisch gruppiert und hierarchisch (da ist die Idee wieder!) strukturiert. Einige neue Ideen runden die einzelnen Gruppen ab. „Master“-Ideen sind wie erwähnt Algorithmisierung, Zerlegung und Sprache (Abb. 8). Man beachte: Bei den kursiv dargestellten Bezeichnungen handelt es sich um Oberbegriffe für Ideengruppen, die zur Systematisierung hinzugenommen wurden, und nicht um Ideen.


Algorithmisierung
Entwurfsparadigmen
Branch and Bound
Divide and Conquer
Greedy-Methode
Line sweeping
Backtracking
Programmierkonzepte
Konkatenation (Sequenz, Feld, Verbund)
Alternative (if, case, var. Verbun d)
Iteration (while, Liste, File, Kel ler, Schlange)
Rekursion (rek. Prozedur, Baum, Suchbaum)
Nichtdeterminismus
Parametrisierung
Ablauf
Prozeß
Nebenläufigkeit
Prozessor
Evaluation
Verifikation
part. Korrektheit
Terminierung
Konsistenz
Vollständigkeit
Fairness
Komplexität
Reduktion
Diagonalisierung
Ordnung
unit-cost Maß
log-cost-Maß
worst/average/
amortized case


strukturierte Zerlegung
Modularisierung
Methoden
Top-down-Methode
Bottom-up-Methode
Geheimnisprinzip
(black box Denken)
Hilfsmittel
Lokalität von Objekten
Spezifikation
abstrakter Datentyp
Teamarbeit
Hierarchisierung
Darstellung
Schachtelung
Baum
Klammerung
Einrückung
Realisierung
Übersetzung
Interpretation
operationale Erweiterung
Orthogonalisierung
Emulation

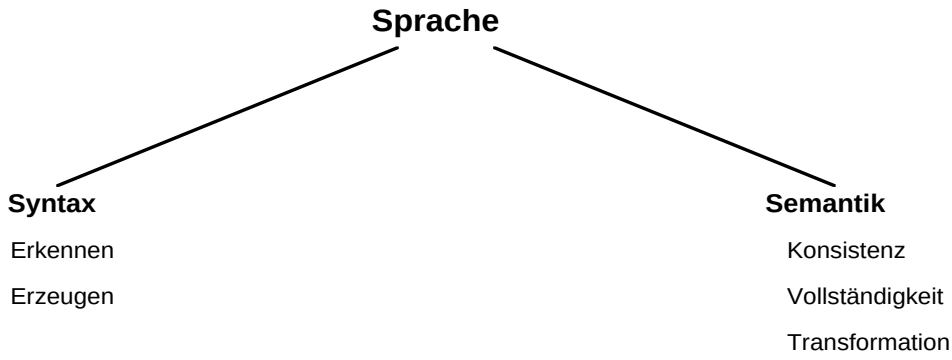


Abb. 9: Fundamentale Ideen der Informatik

TEIL D

Ideenorientierter Informatikunterricht - Praxis

Das Spiralprinzip am Beispiel der Verifikation

Die beiden fundamentalen Ideen der Informatik im Zusammenhang mit der *Bewertung von Algorithmen* sind *Verifikation* (d.h. Überprüfung der Korrektheit des Algorithmus bzgl. einer vorgegebenen Spezifikation) und *Komplexität* (d.h. Analyse des Laufzeit- und Speicherplatzverhaltens des Algorithmus). Mit beiden Ideen sind eine Reihe weiterer fundamentaler Ideen verbunden. Während Fragen der Komplexität in der Schule im angemessenen Umfang und mit formalen Mitteln behandelt werden, führen Verifikationen und Korrektheitsanalysen noch ein Schattendasein (s. etwa die Lehrplanentwürfe von Rheinland-Pfalz oder Niedersachsen aus dem Jahre 1992). Sie gelten als zu formal und zu schwierig, um sie in der Schule in einer Weise zu behandeln, die ihrem Stellenwert in der Informatik entsprechen. Häufig wird dieses Gebiet nur im Zusammenhang mit dem Testen von Programmen thematisiert.

In den folgenden drei Unterrichtsbeispielen wollen wir zwei zentrale Aspekte der Verifikation, namentlich die partielle Korrektheit und die Terminierung eines Algorithmus, auf den jeweiligen kognitiven Niveaus der Primarstufe und den Sekundarstufen I und II vermittelt bzw. spiralig vertieft werden können. Zugleich untermauern diese Beispiele die Leistungsfähigkeit eines an fundamentalen Ideen orientierten Unterrichts.

Beispiel 1: Das Bohnenproblem

Thema der Unterrichtseinheit: Gegeben sind zwei Urnen, eine *Spielurne* gefüllt mit weißen und schwarzen Bohnen und eine *Vorratsurne* gefüllt mit einer (theoretisch) unbegrenzten Menge von schwarzen Bohnen (Abb. 1). Ein Spieler verändert den Inhalt der Urnen durch eine Folge von Spielzügen. Jeder Zug verläuft wie folgt:

Ziehe blind zwei Bohnen aus der Spielurne.
Falls sie die gleiche Farbe besitzen, wirf beide weg und lege
    eine Bohne aus der Vorratsurne in die Spielurne.
Anderenfalls wirf die schwarze Bohne weg und gib die weiße
    zurück in die Spielurne.

Das Verfahren endet, wenn sich nur noch eine Bohne in der Spielurne befindet.
Dieses Problem stammt aus Gries, D., The Science of Programming, Springer Verlag 1989 und wird dort „Coffee Can Problem“ genannt.

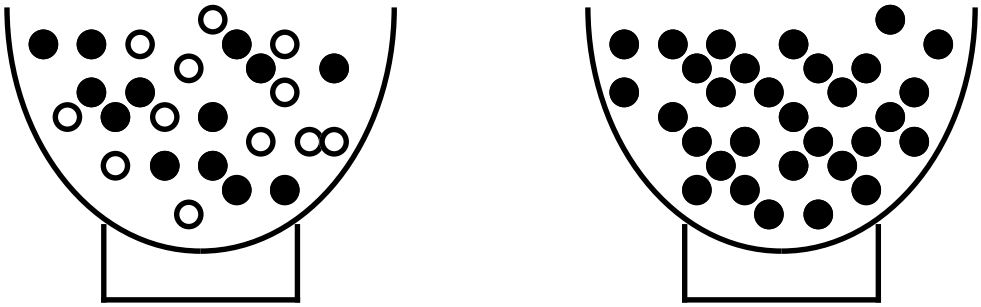


Abb. 1: Bohnenspiel

Problem 1: Endet das Spiel für jede Anfangsbelegung der Spielurne?

Problem 2: Wenn das Spiel endet, welche Farbe besitzt dann die letzte Bohne in der Spielurne?

Lösung zu 1: Mit jedem Spielzug werden zunächst zwei Bohnen aus der Spielurne entfernt und anschließend eine wieder zurückgelegt. Die Zahl der Bohnen in der Spielurne vermindert sich also je Zug um Eins. Befinden sich also anfangs n Bohnen in der Spielurne, so ist das Spiel nach n-1 Zügen beendet.

Lösung zu 2: Über die Zahl der schwarzen Bohnen im Verlaufe eines Spiels ist keine zweckdienliche Aussage möglich, da das Verfahren nichtdeterministisch ist. Zwischenzustände hängen also bei gleichen Startbedingungen vom Zufall ab, da blind gezogen wird. Für die Zahl der weißen Bohnen gilt jedoch: Entweder vermindert sie sich in einem Zug um 2, wenn zwei weiße Bohnen gezogen wurden, oder aber sie bleibt konstant in allen übrigen Fällen. Also wird die *Parität* der weißen Bohnen durch einen Spielzug *nicht* beeinflußt. Folglich ist die letzte Bohne weiß bzw. schwarz, wenn die Zahl der weißen Bohnen anfangs ungerade bzw. gerade war.

Man beachte, daß das Verfahren zwar nichtdeterministisch, aber determiniert ist, also bei gleichen Startbedingungen unterschiedliche Zustandsfolgen durchlaufen kann, die aber immer zum gleichen Ergebnis führen.

Sachanalyse: Aus informatischer Sicht beschreibt die Aufgabenstellung einen nichtdeterministischen Algorithmus, dessen berechnete Funktion zu ermitteln ist. Zum Beweis der totalen Korrektheit des Algorithmus bzgl. des vermuteten funktionalen Zusammenhangs zwischen der Zahl der weißen Bohnen zu Beginn und der Farbe der letzten Bohne ist der Nachweis der Terminierung des Verfahrens sowie der partiellen Korrektheit erforderlich. Beides ist oben schon gezeigt worden. Der Nachweis der partiellen Korrektheit beruht dabei auf der Konstruktion einer Schleifeninvariante, also einer Eigenschaft, die durch die Ausführung eines Spielzugs nicht verändert wird. Schleifeninvariante ist hier die Parität der weißen Bohnen (gerade oder ungerade).

Zielgruppe: Schüler der Primarstufe oder der Sekundarstufe I.

Dauer der Unterrichtseinheit: ca. zwei bis drei Doppelstunden.

Lernziele:
- Einfache Algorithmen nachvollziehen können.
- Die Ideen von Alternative und Iteration entwickeln.
- Einführung in die Verifikation, speziell die Terminierung.
- Präfiguration von Schleifeninvarianten.
- Präfiguration von Nichtdeterminismus und Determiniertheit durch handlungsorientiertes (*enaktives*) vorwegnehmendes, dem kognitiven Niveau der Schüler entsprechendes Lernen.
- Vorbereitung des Übergangs vom empirisch-induktiven zum hypothetisch-deduktiven Problemlösen.

Voraussetzungen: Vier Grundrechenarten, Begriff der geraden und ungeraden Zahl.

Materialien: Ein Urnensatz für je zwei Schüler, je Schüler ca. 6 Protokollblätter (s.u.).

Möglicher Unterrichtsverlauf:
Der Lehrer schreibt die Spielregeln ohne die Abbruchbedingung an die Tafel und führt anschließend einige Spielzüge durch. Dann werden die Schüler in Zweiergruppen aufgeteilt. Jede Gruppe erhält einen Urnensatz sowie mehrere Protokollblätter der Form aus Abb. 2, auf denen die Spielzüge aufgezeichnet werden können.

Name: _____ Nr. des Spiels: _____

Wieviele weiße Bohnen waren zu Beginn des Spiels in der Spielurne: _____

Wieviele schwarze Bohnen waren zu Beginn des Spiels in der Spielurne : _____

Nr. des Zuges	Welches Bohnenpaar wurde gezogen? Kreuze an.		
	zwei weiße	zwei schwarze	eine weiße und eine schwarze

Welche Farbe hat die letzte Bohne in der Spielurne? _____

Abb. 2: Protokollblätter

Anschließend führen die beiden Schüler jeder Gruppe abwechselnd je zwei bis drei Spiele durch und protokollieren Anfangsbedingungen und Verläufe der Spiele. Jeder Spieler kann sich pro Spiel aus einem zentralen Behälter eine beliebige Kombination von weißen und schwarzen Kugeln zusammenstellen.
Nach diesen Experimenten beginnt die Analyse des Spiels. Zunächst ist zu klären, wann das Spiel beendet ist. Hierzu können etwa die Schüler befragt werden, die mit dem Ausruf „fertig“ auf sich aufmerksam gemacht haben: „Warum meint ihr, euer Spiel sei beendet?“ Möglicherweise haben die Schüler bereits während der Experimente Überlegungen angestellt, was denn zu tun sein, wenn sich nur noch eine Bohne in der Spielurne befindet, wo man doch immer zwei herausnehmen muß. In diesem Fall problematisiert man die Fragestellung an Ort und Stelle. Die Schüler sollen bei den Überlegungen erkennen, daß Verfahren vollständig definiert werden müssen und keine Ungenauigkeiten enthalten dürfen. Nun wird die Bedingung „Spielende, falls weniger als zwei Bohnen in der Spielurne enthalten sind“ zu den an die Tafel geschriebenen Spielregeln hinzugefügt.

Es folgen weitergehende Analysen unter Verwendung der Spielprotokolle:

Wird das Spielende immer erreicht?
Da alle Zweiergruppen ihre Spiele beendet haben, werden die Schüler schnell die Vermutung äußern, daß jedes Spiel irgendwann einmal endet. Zur Begründung werden sie die Beobachtung heranziehen, daß die Bohnenzahl der Spielurne fortwährend kleiner wird.

Diese Vermutung ist nun genauer zu begründen, indem die Zugregel daraufhin überprüft wird, wie sich ein Spielzug auf die Zahl der Bohnen in der Spielurne auswirkt. Diese Untersuchungen legen den Grundstein zur Beantwortung der Frage:

Nach wievielen Zügen endet das Spiel?
Möglicherweise führen die Vorüberlegungen schon direkt zum richtigen Ergebnis:

„Zugzahl eines Spiels“=“Anzahl der weißen Bohnen zu Beginn“ +
„Anzahl der schwarzen Bohnen zu Beginn“ - 1.

Anderenfalls fordert man die Schüler auf, ihre Protokollzettel tabellarisch an der Tafel zusammenstellen, etwa in der Form aus Abb. 3.

Name	Nr. des Spiels	Anzahl der weißen Bohnen in der Spielurne zu Beginn des Spiels	Anzahl der schwarzen Bohnen in der Spielurne zu Beginn des Spiels	Anzahl der Züge bis zum Spielende

Abb. 3: Tafelprotokoll

Sodann regt man die Schüler an, über Beziehungen zwischen den Zahlenwerten nachzudenken. Die erwartete Vermutung ist wiederum anhand der Spielregeln zu verifizieren. Hier ist Wert auf eine exakte Begründung zu legen, z.B.:

> „In beiden Alternativen eines Spielzugs entfernt man immer erst zwei Bohnen aus der Spielurne und legt dann eine wieder hinein, so daß sich die Zahl der Bohnen jeweils um Eins vermindert. Eine Bohne verbleibt am Schluß des Spiels in der Spielurne.“

Die letzte Frage erfordert umfangreichere Überlegungen:

Welche Farbe hat die Bohne, die am Schluß des Spiels in der Spielurne verbleibt?
Zunächst läßt man die Schüler die ggf. zuvor an der Tafel erstellte Tabelle aller Spiele um eine Spalte für die Farbe der jeweils letzten Bohne ergänzen. Wiederum sollen die Schüler über funktionale Beziehungen zwischen den Einträgen nachdenken. Anregende Fragen, die die Schüler zur Lösung geleiten können, sind in diesem Zusammenhang:

Welche Elemente des Spiels bestimmen, welche Bohne zuletzt übrigbleibt?
Ist es die Zugfolge?
Ist es die Zahl der weißen Bohnen?
Ist es die Zahl der schwarzen Bohnen?
Ist es die Zahl aller Bohnen zu Beginn des Spiels?

Durch Analyse der Protokolle, der Tabelle an der Tafel und ggf. durch Probespiele in Eigen-

arbeit und vor der ganzen Klasse lassen sich die jeweiligen Vermutungen verwerfen oder erhärten. Darüberhinaus kann durch Variation einzelner Parameter und Festhalten der übrigen ein funktionaler Zusammenhang zwischen Startsituation und Endsituation hergestellt und Parameter ausgesondert werden, die auf die Farbe der letzten Bohne keinen Einfluß haben. Schließlich sollten die Schüler die Anzahl der weißen Bohnen zu Beginn eines Spiels als maßgebende Größe identifizieren. In weiteren Versuchsspielen kann dann eine Wertetabelle durch Gegenüberstellung der anfänglichen Zahl weißer Kugeln zur Farbe der letzten Kugel aufgestellt werden. Hieraus läßt sich leicht die gewünschte Vermutung entwickeln, daß die Farbe der letzten Kugel genau dann weiß ist, wenn die Zahl der weißen Kugeln anfangs ungerade war. Diese letzten Schritte können von den Schülern (in Gruppenarbeit) weitgehend selbständig durchgeführt werden. Geringfügige Hilfen des Lehrers sollten genügen. Die Vermutung ist schließlich wiederum anhand der Zugregel genau zu überprüfen: Ein Spielzug ändert die Parität der weißen Kugeln nicht.

Nun sollte man die gesamte Unterrichtseinheit noch einmal Revue passieren lassen, wobei die Schüler den Ablauf erläutern. Hierbei kann man überprüfen, ob sie schon ein Gespür für die Vorgehensweise bei der Verifikation entwickelt haben (Abb. 4).

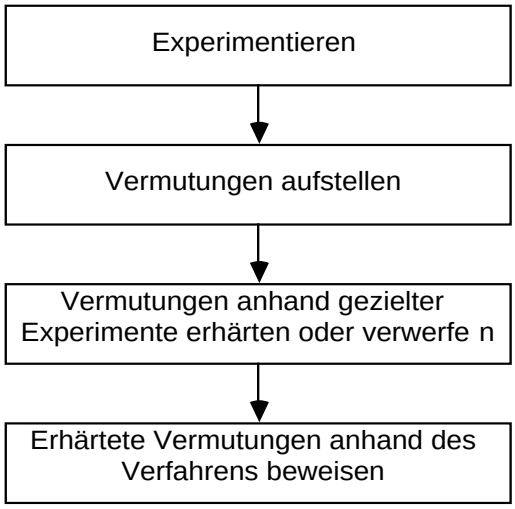


Abb. 4: Unterrichtliche Vorgehensweise

Eine Verinnerlichung dieser Strategien ist jedoch erst im Laufe der Zeit zu erwarten, wenn man die Thematik spiralig vertiefend mehrfach wieder aufgegriffen hat. Überdies befinden sich die Schüler erst im Übergang vom empirisch-induktiven zum hypothetisch-deduktiven Problemlösen. Diese generelle Vorgehensweise beim Problemlösen ist ihnen also noch nicht geläufig und auch nur eingeschränkt vermittelbar.

Im weiteren Verlauf kann der Lehrer den Begriff des Nichtdeterminismus problematisieren: Bei gleichen Anfangsbedingungen können zwei Spieler ganz unterschiedliche Spielzüge durchführen, weil die Bohnen blind gezogen werden. Beide Spieler werden aber am Schluß jeweils genau eine Bohne in der Spielurne behalten, und diese beiden Bohnen besitzen die gleiche Farbe. Die Schüler können nun weitere Beispiele für Verfahren oder Situationen aus ihrer Lebenswelt aufzählen, in denen nichtdeterministische Effekte auftreten, die jedoch auf das Gesamtergebnis keinen Einfluß besitzen.

Beispiel 2: Die Collatz-Funktion

Thema der Unterrichtseinheit: Die Collatz-Funktion ist definiert durch

$$C: \mathbb{N} \to \mathbb{N} \text{ mit } C(n) = \begin{cases} n/2, & \text{falls } n \text{ gerade} \\ 3n+1, & \text{sonst.} \end{cases}$$

Es wird vermutet, daß es zu jedem $n \in \mathbb{N}$ ein $k \in \mathbb{N}$ gibt, so daß $C^k(n)=1$ ist. Bis heute ist kein formaler Beweis für diese Aussage bekannt. Die Aussage ist allerdings für alle natürlichen Zahlen bis etwa 10^{40} verifiziert.

Sachanalyse: Hintergrund der Collatz-Funktion bilden die sog. *Conway-Spiele* (benannt nach J.H. Conway). Idee dieser Spiele ist das wiederholte Anwenden einer einfachen arithmetischen Funktion, die durch endlich viele Alternativen definiert ist. Formal:

Definition:
Ein Zweitupel (R,g) heißt *Conway-Spiel* ($k \in \mathbb{N}$), wenn gilt:

(1) $R=((a_1,b_1),...,(a_k,b_k)) \in (\mathbb{Q}_+ \times \mathbb{Q}_+)^k$ ist ein k-Tupel von Paaren positiver rationaler Zahlen;

(2) $g: \mathbb{N} \to \mathbb{N}$ ist eine Funktion mit

$$g(n) = \begin{cases} a_j \cdot n + b_j \text{ mit } j=\min\{i \mid a_j \cdot n + b_j \in \mathbb{N}\}, & \text{falls } j \text{ existiert,} \\ \text{undefiniert,} & \text{sonst.} \end{cases}$$

Man wählt also bei einem Funktionsaufruf von g das Paar (a_j,b_j) mit kleinstem Index j, für das $a_j \cdot n + b_j$ eine natürliche Zahl ist, sofern solch ein j existiert.

Beispiel: Sei k=2 und $R=((1/2,0),(3,1))$. Dann beschreibt (R,g) die Collatz-Funktion C.

Interessant werden Conway-Spiele, wenn man ihre sog. *Kontraktionseigenschaft* untersucht, also ihr Verhalten bei wiederholter Anwendung.

Definition:
Ein Conway-Spiel (R,g) heißt *kontrahierbar* auf 1, falls für jedes $n \in IN$ eine der beiden folgenden Bedingungen gilt:
(1) $g(n)=n$,
(2) Es gibt ein $k \in IN$ mit $g^k(n)=1$.

Die Kontraktionseigenschaft ist algorithmisch nicht entscheidbar.

Satz (Conway, J.H.: Unpredictable iterations. Proceedings of the Number Theory Conference (1972) 45-52):
Es gibt keinen Algorithmus, der zu jedem Conway-Spiel (R,g) entscheidet, ob (R,g) kontrahierbar ist.

Die Kontraktionseigenschaft von C ist auch noch nicht genau bekannt. Der folgende Satz läßt jedoch eine gewisse Kontraktion vermuten.

Satz (Everett, C.J.: Iterations of the number-theoretic function f(2n)=n, f(2n+1)=3n+2. Advances in Mathematics 25 (1977) 42-45):
Für fast alle $n \in IN$ gibt es ein $k \in IN$ mit $C^k(n)<n$.

Zielgruppe: Schüler der Sekundarstufe I. Bezieht man sich nicht auf eine konkrete Programmiersprache, sondern beläßt es bei einem Algorithmus in umgangssprachlicher Notation, so eignet sich diese Einheit auch für die Primarstufe.

Dauer der Unterrichtseinheit: etwa zwei Doppelstunden.

Lernziele:
- Weiterentwicklung der fundamentalen informatischen Ideen von *Alternative* und *Iteration*.
- Erstmalige Verwendung der PASCAL-Sprachelemente für bedingte Anweisungen und bedingte Schleifen.
- Vertiefung der Idee der *Terminierung* im Zusammenhang mit bedingten Schleifen.

Voraussetzungen: Kenntnis der Zuweisung, der Sequenz und des Datentyps integer mit seinen wichtigsten Operationen.

Möglicher Unterrichtsverlauf:
Der Lehrer gibt folgende Zahlenfolge vor
(A) 17 52 26 13 40 20 10 5 ...

und bittet die Schüler herauszufinden, welches Bildungsgesetz sich hinter der Folge verbirgt und die Folge korrekt fortzusetzen. Erfahrungsgemäß wird es hierbei noch Schwierigkeiten geben, da die Folge zur Erkennung eines Bildungsprinzips noch zu kurz ist. Daher setzt der Lehrer fort:

... 16 8 4 2 1 4 2 1 4 2 1 ...

Bei der zweiten Folge mit gleichem Bildungsgesetz wird es bereits einige Vorschläge geben, wie man fortzusetzen hat:

(B) 37 112 56 28 14 7 22 11 34 17 52 ...

Die meisten Schüler werden wie in Folge (A) fortsetzen, da die Zahlen 17 und 52 dem Beginn der Folge (A) entsprechen. Hierbei setzen sie stillschweigend voraus, daß zur Berechnung eines Elements der Folge nur das unmittelbar vorhergehende heranzuziehen ist. Auch wenn diese Annahme im allgemeinen bei solchen Folgenspielen nicht zutreffen mag, empfiehlt es sich hier auf eine diesbezügliche Bemerkung zu verzichten, um den Verlauf der Diskussion nicht zu unterbrechen.

Die dritte Folge schließlich offenbart den meisten Schülern das Bildungsgesetz:

(C) 30 15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1 4 2 1 ...

Die vierte Folge dient zur Erhärtung der Vermutung:

(D) 75 226 113 340 170 85 256 128 64 32 16 8 4 2 1 4 2 1 ...

Man läßt das Bildungsgesetz nun für alle von einem Schüler an die Tafel schreiben. Das Ergebnis könnte etwa sein:

a) umgangssprachlich:

Wenn die letzte Zahl eine gerade Zahl ist, dann halbiere sie
und erhalte die nächste;
Wenn sie ungerade ist, dann multipliziere sie mit Drei und addiere Eins.

b) formal: Sei x_1, x_2, x_3, ... die gesuchte Folge. x_1 ist gegeben. Dann berechnet sich x_i für $i \geq 2$ durch

$$x_i = \begin{cases} x_{i-1}/2, & \text{falls } x_{i-1} \text{ gerade,} \\ 3x_{i-1}+1, & \text{sonst.} \end{cases}$$

Einige weitere Übungen mit unterschiedlichen Anfangswerten x_1 werden die Schüler recht schnell zur Vermutung führen, daß alle Folgen schließlich in ... 4 2 1 4 2 1 ... übergehen. Diese Vermutung kann der Lehrer im allgemeinen scheinbar entkräften, indem er z.B. den Anfangswert x_1=27 vorgibt:

27 82 41 124 62 31 94 47 142 71 214 107
322 161 484 242 121 364 182 91 274 137 412
206 103 310 155 466 233 700 350 175 526 263
790 395 1186 593 1780 890 445 1336 668 334 167
502 251 754 377 1132 566 283 850 425 1276 638
319 958 479 1438 719 2158 1079 **3238** **1619** **4858**
2429 **7288** **3644** 1822 911 2734 1367 4102 2051 6154

3077 9232 4616 2308 1154 577 1732 866 433 1300
650 325 976 488 244 122 61 184 92 46 23 70
35 106 53 160 80 40 20 10 5 16 8 4 2 1...

Wenn die Schüler beim Rechnen in den fett markierten Zahlenbereich gelangen, werden sie zunächst den Glauben an ihre Vermutung verlieren. Da die Berechnung der Folgenglieder nun schon recht aufwendig geworden ist, wird leicht der Wunsch nach Programmierung der Berechnung auf einem Computer geweckt.

Zunächst wird die Aufgabe des Algorithmus genau spezifiziert: Er soll eine natürliche Zahl einlesen und nacheinander alle Zahlen der Folge nach obiger Formel berechnen und ausgeben.

Man beginnt zunächst mit einem umgangssprachlichen Algorithmus, den die Schüler selbst entwickeln können (man sollte sich hierbei nach der Ausdrucksweise der Schüler richten, meist hat sich allerdings im vorangegangenen Unterricht schon ein Sprachgebrauch etabliert), z.B.:

Eingabe einer Zahl x;
Ausgabe von x;
Wiederhole:
    Wenn x gerade ist, dann setze x auf x/2
        anderenfalls setze x auf 3x+1;
    Ausgabe von x.

Als nächstes ist herauszuarbeiten, welche bereits bekannten und welche neuen Elemente in diesem Algorithmus vorkommen. Welche Teile können in PASCAL bereits dargestellt werden? Offenbar:

Eingabe einer Zahl x —> read(x)
Ausgabe von x —> write(x)
setze x auf x/2 —> x:=x div 2
setze x auf 3x+1 —> x:=3*x+1

Die beiden neuen Kontrollstrukturen „Wenn ... dann ...“ und „Wiederhole ...“ werden als *bedingte Anweisung* und als *Schleife* identifiziert. Mit bedingten Anweisungen kann man (hier zunächst einseitige) *Alternativen*, mit Schleifen *Iterationen* formulieren.
Wo treten Alternativen im täglichen Leben auf? Z.B. bei der Wahl einer Busverbindung, bei der Kurswahl zu Beginn eines Schuljahres, bei der Wahl eines Mopeds, der Ausbildungsstelle.
Wo findet man Iterationen? Z.B. als Reim, Rhythmus, Refrain, Ornament in Sprache, Musik und Kunst, bei der Autowaschanlage, die zyklisch dasselbe Waschprogramm abfährt, bei der Unterrichtswoche mit ständig wiederkehrenden Fächern, beim Tagesrhythmus der Schüler und Eltern.

An dieser Stelle empfiehlt es sich, auf die fundamentale informatische Idee der *Hierarchisierung* durch *Schachtelung* und ihre Darstellung (hier: durch Einrückung) hinzuweisen. Als Aufhänger kann etwa die Frage dienen: Welche Anweisungen gehören zur Schleife, welche nicht?

Der Lehrer kann nun einen Schüler auffordern, den Algorithmus etwa für die Zahl 5 nachzuvollziehen: 5 16 8 4 2 1 4 2 1 4 2 1; spätestens an dieser Stelle wird den Schülern die Problematik klar: es wurde eine *Endlosschleife* definiert. Das Programm *terminiert* (zumindest für die gewählte Eingabe) nicht. Eine sprachliche Präzisierung des Begriffs Termination für einzelne/alle Eingaben kann sich anschließen.
Der Lehrer rät nun dazu, den obigen Algorithmus so abzuändern, daß die Erzeugung von neuen Folgengliedern nur *solange* fortgesetzt wird, wie x *noch ungleich* 1 ist, da man anderenfalls in den bekannten Zyklus 4 2 1 4 2 1 ... hineinläuft. Die Schüler werden auf diese Anregung hin vermutlich genau auf eine umgangssprachliche Fassung der while-Schleife kommen:

Solange x≠1 ist, wiederhole: ... .

Der Begriff der *bedingten Schleife* wird nun präzisiert und ggf. an einigen kleineren ad hoc-Beispielen (wie Summe der ersten n Zahlen) weiter erläutert.

Eine nochmalige Überprüfung des Algorithmus für den Eingabewert 5 liefert jetzt ein zufriedenstellendes Ergebnis. Terminiert das Programm nun für *alle* Eingaben? Eine Antwort hierauf wird vorerst bis zum Schluß der Unterrichtseinheit zurückgestellt.

Es folgt eine Beschreibung der Syntax von (einseitiger) bedingter Anweisung und Schleife in PASCAL. Hierbei ist ggf. zu erklären, daß der Anweisungsteil der beiden Strukturen aus einzelnen Elementaranweisungen oder einer durch begin/end geklammerten Sequenz bestehen kann. Auf die exakte Bedeutung dieser beiden Kontrollstrukturen und die möglichen Ausprägungen der Bedingungen bei Alternative und Schleife braucht zu diesem Zeitpunkt noch nicht eingegangen zu werden, da die Schüler den Sinn intuitiv begreifen können. Einzelne Details können zu passender Gelegenheit (s. unten) ergänzt werden (genetisches Unterrichtsprinzip).

Nun entwickeln die Schüler leicht folgendes Programm:

```
program folge1(input,output);
var x:integer;
begin
      read(x); write(x);
      while x≠1 do
      begin
            if x mod 2=0 then x:=x div 2
                  else x:=3x+1;
            write(x)
      end
end.
```

Das Programm ist in dieser Form mit den Vorbemerkungen selbsterklärend.

Nach einigen Experimenten am Rechner schreibt der Lehrer einen Wettbewerb aus: Wer findet die längste Folge bei einer Anfangszahl zwischen 1 und 1000? Die Schüler werden nun nach kurzer Zeit, statt die Elemente jeder Folge abzuzählen, selbständig zusätzlich eine Zählervariable einführen, in der für den eingegebenen Anfangswert die Länge der jeweiligen Folge festgehalten wird:

```
program folge1(input,output);
var x,laenge:integer;
begin
    read(x); write(x);
    laenge:=1;
    while x≠1 do
    begin
        if x mod 2=0 then x:=x div 2 else x:=3x+1;
        write(x);
        laenge:=laenge+1
    end;
    writeln(‚Länge der Folge‘,laenge)
end.
```

Nach weiteren Testläufen wird sich der Wunsch einstellen, alle Anfangswerte zwischen 1 und 1000 systematisch durchzuprobieren, um denjenigen zu finden, der die längste Folge liefert. Wiederum wird zunächst ein umgangssprachlicher Algorithmus angefertigt:

Für jeden Anfangswert a zwischen 1 und 1000 tue folgendes:
    Ausgabe von a;
    Setze x auf a;
    Setze laenge auf 1;
    Solange x≠1 ist, wiederhole:
        Wenn x gerade ist, dann setze x auf x/2, anderenfalls setze x auf 3x+1;
        Erhöhe laenge um 1;
    Ausgabe von laenge.

Nun kann man entweder diesen Ansatz aufgreifen und die Zählschleife einführen oder aber die Schüler anregen, den Algorithmus so umzuschreiben, daß nur bereits bekannte programmiersprachliche Elemente verwendet werden können.
Wiederum ist auf die Idee der textuellen Hierarchisierung/Schachtelung durch Einrückung hinzuweisen, aus der sich jeweils genau die Reichweite beider Schleifen ergibt.

Hausaufgabe:
Setze den Algorithmus in ein PASCAL-Programm folge2 um und bestimme den Anfangswert zwischen 1 und 1000, der die längste Folge liefert.

Die obigen Überlegungen zur *Terminierung* des Programms werden nun in der nächsten Stunde wieder aufgegriffen. Geeignete Fragen zur Motivierung sind:
- Wie ist die Anfangsvermutung „Alle Folgen enden bei Eins bzw. mit 1 4 2 1 4 2 ...“

nach diesen Experimenten zu entscheiden?
- Wie reagiert das Programm folge1, wenn man eine Zahl eingibt, deren Folge nicht bei 1 endet?
- Wie ist zu bewerten, wenn der Computer bei der Ausführung des Programms folge1 ungewöhnlich lange rechnet, also z.B. nach zwei Stunden immer noch keine Eins ausgegeben hat?
- Terminiert das Programm folge1 für alle Eingaben?

Zum Schluß der Diskussion kann auf die wissenschaftliche Attraktivität dieser Fragestellung eingegangen werden:
- Hinter der Vermutung verbirgt sich ein schwieriges zahlentheoretisches Problem.
- Es ist nicht bekannt, ob alle Folgen bei Eins enden.
- Die Vermutung ist für alle Zahlen bis etwa 10^{40} nachgewiesen.

Was bedeuten diese Aussagen für die Terminierung des Programms folge1?

Beispiel 3: Formaler Terminationsnachweis

Thema der Unterrichtseinheit: Untersuchung der Terminierung des folgenden Programmstücks:

```
var x,y: integer;
while y≠0 do
    if x=0 then y:=y-1 else
    begin
        y:=x+1;
        x:=x-1
    end
end.
```

Es ist zu beweisen, daß das Programmstück für beliebige Anfangswerte $x,y \in \mathbb{N}_0$ terminiert. Hierzu wählt man eine Funktion

$$\tau: \mathbb{Z}\times\mathbb{Z}\to\mathbb{Z}$$

und zeigt, daß sich der Wert von τ angewendet auf die Variablen x und y mit jedem Schleifendurchlauf verkleinert, gleichzeitig aber für $x,y \in \mathbb{N}_0$ nach unten durch 0 beschränkt ist. Eine geeignete Wahl für τ ist z.B.

$$\tau(x,y) = \begin{cases} y, & \text{falls } x=0 \\ x+2, & \text{falls } x\neq 0. \end{cases}$$

Offenbar gilt dann $\tau(x,y)\geq 0$ für alle $x,y \in \mathbb{N}_0$. Nun ist zu zeigen, daß sich der Wert von τ mit jedem Schleifendurchlauf vermindert. Seien zu Beginn des Schleifendurchlaufs

$x=\alpha$ und $y=\beta$ mit $\alpha,\beta \in \mathbb{N}_0$.

Da die Abbruchbedingung dann nicht erfüllt ist, folgt $\beta\neq 0$, also $\beta>0$.

Fall 1: $\alpha=0$.

Dann ist $\tau(x,y)=\beta$ vor dem Schleifendurchlauf und $\tau(x,y)=\beta-1$ nach dem Schleifendurchlauf.

Fall 2: $\alpha \neq 0$.

Fall 2.1: $\alpha > 1$.

Dann ist $\tau(x,y)=\alpha+2$ vor dem Schleifendurchlauf und $\tau(x,y)=\alpha-1$ nach dem Schleifendurchlauf.

Fall 2.2: $\alpha = 1$.

Dann ist $\tau(x,y)=3$ vor dem Schleifendurchlauf und $\tau(x,y)=2$ nach dem Schleifendurchlauf.

Damit ist gezeigt, daß τ streng monoton fällt. Folglich terminiert die Schleife für $x,y \in \mathbb{N}_0$.

Sachanalyse: Den Nachweis der Terminierung von Programmen kann man nicht maschinell durchführen, genauer: Es gibt keinen Algorithmus, der für beliebige Programme nachprüft, ob sie für alle Eingaben terminieren oder nicht (*Halteproblem*). Zum Nachweis der Terminierung eines Programms ist also stets eine gewisse Handarbeit und die Intuition eines Menschen erforderlich.

Offenbar kann ein Programm nur terminieren, wenn zumindest alle Schleifen terminieren. Ein recht praktikables und häufig (aber nicht immer) zum Ziel führendes Verfahren, die Termination einer Schleife nachzuweisen, besteht darin, eine Funktion τ von den Wertemengen der in der Schleife vorkommenden Variablen in die Menge der ganzen Zahlen zu ermitteln, für die gilt: τ ist nach unten beschränkt, und τ verkleinert sich mit jedem Schleifendurchlauf. Findet man τ, so terminiert die Schleife.

Genauer: Seien $x_1,...,x_r$ die in einer Schleife vorkommenden Variablen mit den Wertebereichen $W_1,...,W_r$. Man sucht eine Funktion

$$\tau: W_1 \times ... \times W_r \rightarrow \mathbb{Z}$$

mit

(1) $\tau(x_1,...,x_r) \geq 0$ für alle $x_i \in W_i$, $i=1,...,r$.

(2) Gilt vor Ausführung des Schleifenrumpfes $\tau(x_1,...,x_r)=t>0$, so gilt nach Ausführung des Rumpfes $\tau(x_1,...,x_r)<t$.

Den Nachweis von (2) führt man im allgemeinen Fall mit Methoden der axiomatischen Semantik. Für die Schule sind solch hohe Anforderungen an die formale Strenge entbehrlich. Hier genügt es die Wirkung des Schleifenrumpfes auf die Variablen und den Wert von τ mithilfe von Ablaufprotokollen zu bestimmen.

Zielgruppe: Schüler der Sekundarstufe II.

Dauer der Unterrichtseinheit: ca. zwei bis drei Doppelstunden.

Lernziele: Vertiefung der fundamentalen Idee der Terminierung durch Formalisierungsansätze.

Voraussetzungen: Kenntnis von Ablaufprotokollen, des Begriffs der Terminierung sowie verschiedener mathematischer Grundbegriffe wie Monotonie und Beschränktheit von Funktionen.

Materialien: Um Denkfähigkeit und Abstraktionsvermögen zu trainieren sowie Ablenkungen möglichst gering zu halten, wird der Rechner *nicht* genutzt.

Möglicher Unterrichtsverlauf:
Der Lehrer schreibt das Programmstück an die Tafel und fordert die Schüler auf, Anfangswerte(-mengen) von x und y zu nennen, für die das Programm terminiert. Die Zuhilfenahme des Rechners ist nicht erlaubt. Für jeden genannten Anfangswert bzw. jede genannte Menge von Anfangswerten verlangt man vom Schüler eine Begründung, warum das Programm für diese Werte terminiert. Als Vorstufe zu einer späteren Formalisierung sollen die Schüler hier zu einer möglichst präzisen Erläuterung angehalten werden. Wertebereich und Begründung werden tabellarisch an der Tafel festgehalten, z.B. wie in Abb. 5.

Anfangswert oder Anfangswertemenge für x und y	Begründung
y=0	Die Schleife wird gar nicht betreten.
x=0 und y>0	Die Schleife wird betreten, wobei nur der then-Zweig durchlaufen wird. Hier wird y fortlaufend um 1 vermindert, bis y=0 ist und das Abbruchkriterium erreicht ist.
x>0 und y>0	Die Schleife wird betreten, wobei zuerst im else-Zweig x um 1 heruntergezählt wird, bis x=0 ist (zu diesem Zeitpunkt ist y=2). Dann weiter wie im Fall x=0, y>0.
x=-1 und y≠0	...
x>0 und y≠0	...

Abb. 5: Terminationsbereiche (mögliches Tafelbild)

Zur Veranschaulichung und um eine bessere Übersicht zu behalten, welche Eingabepaare (x,y) bereits untersucht worden sind, kann man die genannten Anfangswerte(-mengen) zusätzlich in einem Koordinatensystem festhalten (Abb. 6).

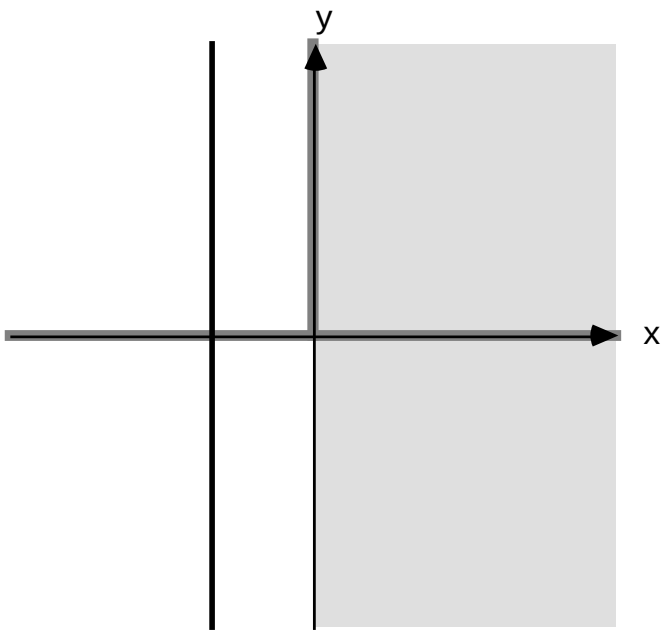


Abb. 6: Graphische Darstellung der Terminationsmenge

Schließlich wird durch Zusammenfassung aller Anfangswertemengen der mutmaßliche Terminationsbereich T des Programmstücks ermittelt:

$$T=\{(x,y) \mid x>0 \text{ oder } x=-1 \text{ oder } y=0 \text{ oder } (x=0 \text{ und } y>0)\}.$$

Nun sind die Überlegungen zu präzisieren und zu formalisieren. Der Einfachheit halber beschränkt man sich zunächst auf Anfangswerte $x,y \geq 0$. Mittels folgender motivierender Fragen kann man die Schüler zu einer Lösung geleiten:

- Was ist die notwendige Bedingung für die Termination der Schleife?
  Erwartete Antwort: Die Abbruchbedingung y=0 muß erreicht werden.
- Auf welche Weise erreicht man von den Anfangswerten die Abbruchbedingung?
  Erwartete Antwort: y muß fortlaufend vermindert werden.
- Ist dies denn der Fall, wird y also in der Schleife immer vermindert, z.B. für (x,y)=(17,1)?
  Erwartete Antwort: Nein, y kann sich im else-Zweig auch erhöhen, aber dafür wird ja dann x vermindert. Zwischenzeitliche Erhöhungen von y schaden also nicht. y muß sich nur auf lange Sicht verringern.
- Wie kann man formulieren, daß man sich trotz zwischenzeitlicher Erhöhungen von y dennoch der Abbruchbedingung y=0 nähert?
  Erwartete Antwort: Es ist nicht nur der Werteverlauf von y maßgebend, sondern man muß auch berücksichtigen, daß sich x vermindert, was dann im else-Zweig dafür sorgt, daß sich später trotz zwischenzeitlicher Erhöhungen auch y wieder vermindert.
- Wie kann man das mathematisch formulieren?

An dieser Stelle wird der geführte Dialog vermutlich abbrechen, da die Schüler nicht in der Lage sind, die beiden Beobachtungen zur gewünschten Funktion τ zu kombinieren. Als

weitere Hilfe kann man evtl. für gewisse Anfangswerte (x,y) den Werteverlauf im Koordinatensystem darstellen, z.B. für (x,y)=(4,2) und (x,y)=(3,5) wie in Abb. 7.

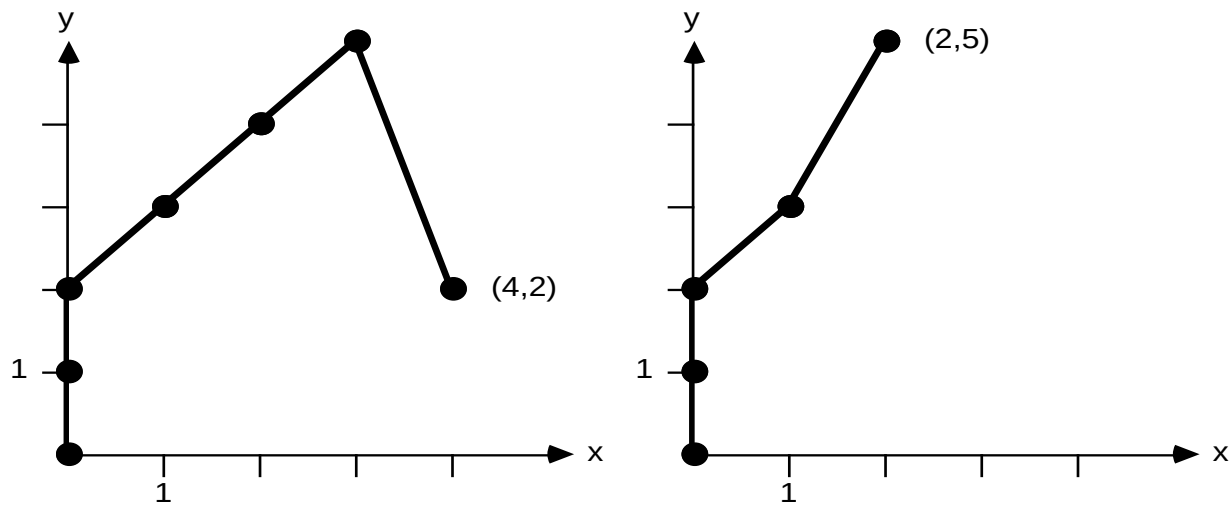


Abb. 7: Werteverläufe der Variablen x und y

Der Lehrer kann nun weiterhelfen und die Funktion

$$\tau(x,y)=\begin{cases} y, & \text{falls } x=0 \\ x+2, & \text{falls } x\neq 0 \end{cases}$$

anschreiben. Danach befragt man die Schüler nach den Eigenschaften von τ für $x,y\geq 0$ und ihrem Nutzen für den Nachweis der Terminierung der Schleife. Zur Unterstützung können die Schüler das Programmstück für einzelne Anfangswerte $x,y\geq 0$ nachvollziehen, Ablaufprotokolle hierzu anfertigen und den Verlauf von τ beobachten sowie alles im Koordinatensystem festhalten. Diese Aufgabe eignet sich auch als Hausaufgabe zwischen den beiden Doppelstunden.

Erwartete Antworten:

(1) τ ist nie negativ, falls die Argumente nicht negativ sind.

(2) τ verringert sich mit jedem Schleifendurchlauf um Eins.

(3) τ hat den Wert 0, wenn die Abbruchbedingung erreicht ist.

τ beschreibt also die Beobachtung, daß die Dekrementierung sowohl von x als auch von y zum Erreichen der Abbruchbedingung y=0 beiträgt.

Diese drei Aussagen sind jetzt allgemein zu verifizieren. Aussagen (1) und (3) sind einfach. Für Aussage (2) verwendet man wiederum Ablaufprotokolle: Sei zu Beginn des Schleifenrumpfes $x=\alpha$ und $y=\beta$, $\alpha,\beta\geq 0$:

Fall 1:	x	y	τ	
	α=0	β	β	
	α=0	β-1	β-1	√

Fall 2:	x	y	τ	
	α≥2	β	α+2	
	α-1	α+1	α+1	√

Fall 3:	x	y	τ	
	α=1	β	3	
	0	2	2	√

Zwischenfrage zur Vertiefung des Verständnisses: Die obige Funktion τ sieht recht gekünstelt aus. Wie wäre es also, wenn man folgende einfachere Funktion τ' gewählt hätte:

$$\tau'(x,y)=\begin{cases} y, \text{ falls } x=0 \\ x, \text{ falls } x\neq 0 \ ? \end{cases}$$

Erwartete Antwort ggf. unter Verwendung von Ablaufprotokollen: Dieses τ' eignet sich nicht, da beim Übergang vom else- zum then-Zweig für x=1, y≥0 der Wert von τ' von 1 auf 2 springt, sich also nicht vermindert.

Im letzten Schritt sind nun die bisherigen Überlegungen zu verallgemeinern. Wie kann man im allgemeinen Fall nachweisen, ob eine Schleife terminiert? Erwartet werden mit Unterstützung des Lehrers die im Abschnitt „Sachanalyse" genannten Kriterien.

In einer Hausaufgabe können die Schüler dann versuchen, eine entsprechende Funktion τ für den gesamten mutmaßlichen Terminationsbereich T aufzustellen. Abschließend kann ggf. in einer weiteren Doppelstunde auf das Halteproblem eingegangen werden.

VORLESUNG E

Anfangsunterricht

In jedem Fach ist Anfangsunterricht reizvoll und anspruchsvoll zugleich, sowohl auf der Lehrer- wie auf der Schülerseite. Auf beiden Seiten existiert eine Erwartungshaltung bezogen auf das Fach und das „neue Gesicht“. Zu diesen Elementen kommen beim Informatikunterricht noch eine Reihe spezifischer Probleme hinzu, die diesen Unterricht besonders „aufregend“ gestalten:

- Die Schülergruppe ist meist äußerst inhomogen. Einige Schüler besitzen bereits erhebliche Kenntnisse in Informatik (oder in dem, was sie für Informatik halten), andere haben möglicherweise noch niemals mit einem Computer gearbeitet.
  Dies betrifft vor allem das Verhältnis zwischen Jungen und Mädchen, da die Mädchen i.a. die geringeren Vorkenntnisse besitzen. Hierauf gehen wir in einem eigenen Abschnitt dieser Vorlesung ein.
- Die Schüler mit Vorkenntnissen leiden oftmals unter Selbstüberschätzung und mangelnder Kooperationsbereitschaft.
- Die Schüler mit geringen Vorkenntnissen wünschen sich einen Einblick in die Informatik, eine Anleitung zur Bedienung und Nutzung des Computers und möchten ein gewisses Verständnis für die Funktionsweise des Computers erwerben. Die Schüler mit hohen Vorkenntnissen sind vor allem an einigen weiteren Tricks zur Beherrschung der Maschine interessiert.
- Die Schule ist häufig mit überalterten Rechnern ausgestattet, die vor allem bei denjenigen Schülern, die bereits einen eigenen Rechner besitzen, auf Verachtung trifft und kaum noch einen Reiz ausübt.

Bevor wir einige Überlegungen zur prinzipiellen Durchführung des Anfangsunterrichts in Informatik anstellen, wollen wir zunächst seine beiden wichtigsten Ziele herausstellen:

(1) Der Anfangsunterricht sollte ein reduziertes, aber unverfälschtes und abgerundetes Bild der Informatik vermitteln. Dies betrifft vor allem diejenigen Schüler, die Informatik nach der Einführung wieder abwählen.

(2) Für die übrigen dient der Anfangsunterricht insbesondere zur Bildung einer Grundlage, und er liefert eine Vorschau auf mögliche vertiefende Inhalte in nachfolgenden Grund- und Leistungskursen.

Um diese Ziele zumindest ansatzweise zu erreichen und den Schülern den Zugang zu Informatik und Informationstechnik zu erleichtern, sollten die Themen, die im Anfangsunterricht behandelt werden, gewisse Eigenschaften besitzen [Koerber/Peters in LOGIN 15,1 (1995) 17-21]: Sie sollten

- in geeigneter didaktischer Reduktion für den Einsatz von Informatiksystemen typische Anwendungsfälle behandeln,

- als Problem offen, herausfordernd und so komplex sein, daß eine Lösung nicht sofort, aber mehrschrittig erreichbar ist,
- konstruktive Möglichkeiten zur Eigengestaltung des Einsatzes von Informatiksystemen bieten,
- den Schülern leicht zugänglich sein, um das Maß der notwendigen außerinformatischen Sachinformationen beschränken zu können und die Schüler nicht mit zu vielen neuen Informationen überhäufen,
- möglichst Aspekte gesellschaftlicher Bereiche berühren, wobei diese Aspekte für die Schüler im Bereich der erfahrbaren Realität liegen sollten. Dies betrifft vor allem den Anfangsunterricht in Form einer informationstechnischen Grundbildung (ITG);
- über die unmittelbaren Lernziele hinaus Gelegenheit bieten, weitere Sachinformationen als Beitrag zur Allgemeinbildung vermitteln zu können. Auch dieser Punkt richtet sich in erster Linie an den Anfangsunterricht im Rahmen der ITG.

In den ca. 20 Jahren Informatik in der Schule haben sich verschiedene, mehr oder weniger leistungsfähige Ansätze für den Anfangsunterricht Informatik herausgebildet, die wir im folgenden klassifizieren und bewerten wollen. Als Bewertungsmaßstab orientieren wir uns dabei an den Zielen (1) und (2) des Anfangsunterricht sowie den Hinweisen zur Wahl von adäquaten Themen.

Der programmiersprachliche Zugang.

Merkmale: Mit permanentem Bezug auf eine (meist imperative) Programmiersprache werden bottom-up beginnend bei den elementaren sukzessive immer komplexere Sprachkonstrukte der Programmiersprache eingeführt. Die jeweiligen Anwendungsbeispiele betreffen fiktive Problemstellungen, deren Lösung nur eine geringe oder gar keine Modellbildung vorausgeht. Sie sind folglich relativ klein und beschränken sich auf die Hervorhebung der zuletzt eingeführten Konstrukte der Sprache.

Vorteile: Diese Vorgehensweise
- entspricht meist den Wünschen der männlichen (vorgebildeten) Schüler, denen es eher auf den Erwerb weiterer Spezialkenntnisse ankommt,
- führt zu einer natürlichen systematischen Gliederung des Unterrichts durch das schrittweise Einführen von immer komplexeren Sprachelementen,
- entspricht der gewohnten unterrichtlichen Praxis, wie sie aus dem Fremdsprachenunterricht bekannt ist,
- gibt den Lehrkräften, die meist nur unzureichend fort- oder weitergebildet sind, eine gewisse fachliche Sicherheit.

Nachteile: Diese Vorgehensweise
- löst nicht das Problem, die unterschiedlichen Vorkenntnisse der Schüler aneinander

anzugleichen. Vielmehr verstärkt sie das Problem noch, vor allem wenn sich der Unterricht an der gängigen Sprache PASCAL orientiert.,
- blendet für eine längere Zeit jedwede praktische Relevanz der Informatik durch die Behandlung nur kleiner, anwendungsferner Aufgaben aus, deren Lösungen zumeist nur eine geringe Qualität besitzen,
- wirkt bisweilen demotivierend auf die Schüler, weil lange Zeiträume mit „Lernen auf Vorrat“, d.h. mit der bloßen Anreicherung von Programmiersprachkenntnissen, überbrückt werden müssen, während der die Schüler kaum vorzeigbare Produkte entwikkeln können. PASCAL, obwohl als Ausbildungssprache konzipiert, ist noch nicht schlank genug, um für schulische Zwecke uneingeschränkt empfohlen werden zu können,
- bietet daher wenig Möglichkeiten, gesellschaftliche Aspekte, wie sie in realen Anwendungssituationen auftreten, organisch in den Unterricht einzubeziehen. Überspitzt formuliert führt dieser Ansatz auf die Gleichung

  Informatikunterricht = Programmierkurs + Gesellschaftskunde [Riedel1981],
- erzeugt den falschen Eindruck, Informatik sei identisch mit Programmieren, bereitet auf ein Leben als Programmierer vor und vermittelt insgesamt kein abgerundetes Bild der Informatik.

Beispiel: Informatikunterricht als Programmierkurs in PASCAL:
- Einführung der elementaren Datentypen integer, real etc. und elementarer Anweisungen
- einfache Programme mit Sequenz, bedingter Anweisung und Zählschleife, vorzugsweise mit mathematischem Hintergrund
- strukturierte Datentypen wie Feld, Record, verschiedene Schleifentypen, Textverarbeitung
- Prozedurkonzept, Funktionen und Prozeduren, Modularisierung, erste kleinere Projekte mit anwendungsorientiertem Hintergrund
- Files, Zeigerkonzept, lineare Listen, Projekte
- Rekursion im Daten- und Kontrollbereich usw.

In der letzten Zeit hat man versucht, einige der Nachteile des programmiersprachlichen Ansatzes durch die Wahl anderer, schlankerer Programmiersprachen auszugleichen. Zu diesen Ansätzen zählt insbesondere die Verwendung von nicht-prozeduralen Programmiersprachen, speziell von PROLOG. Die bloße Beherrschung dieser Sprachen, die meist nur wenige, syntaktisch einfache Sprachmittel besitzen, tritt in den Hintergrund. Stattdessen erwerben die Schüler stärker Beschreibungs- und Problemlösekompetenzen und können sich auf die Modellbildungsaktivitäten konzentrieren, so daß praxisorientierte Anwendungen bereits frühzeitig zugänglich werden. Zugleich nivelliert dieser Ansatz die unterschiedlichen Vorkenntnisse der Schüler.

Der systemanalytische Zugang.

Merkmale: In einem top-down-orientierten Vorgehen studieren die Schüler ein komplexes Softwaresystem: Die Studie beginnt mit der Benutzung des Systems (*Blick auf das System*), zu der auch der Erwerb elementarer Bedienfertigkeiten im Umgang mit dem Computer, der Umgang mit der Dokumentation und die Bewertung des Systems gehören. Es folgt die Analyse des Systems (*Blick in das System*): Identifikation der Bestandteile des Systems unter Zuhilfenahme der Dokumentation, Analyse ihres Zusammenwirkens und ihrer Funktionalität, Modularisierung, Parametrisierung usw. Nachdem die Schüler ein gewisses Verständnis für das System gewonnen haben, schließt sich eine Wartungsphase (*Modifikation des Systems*) an: Das System wird geringfügig ergänzt oder an aktuelle Anforderungen angepaßt. In dieser Phase werden die ersten Programmiererfahrungen auf einer sehr hohen sprachlichen Ebene erworben. In der letzten offenen Phase (*Konstruktion des Systems*) werden diese Programmiererfahrungen ausgedehnt: Unter teilweiser Wiederverwendung von Bausteinen des Ausgangssystems konstruieren die Schüler ein neues System zu einem vergleichbaren Problem. Hierbei gewinnen sie bereits umfangsreiche Programmiererfahrungen im Rahmen eines projektorientierten Vorgehens.

Vorteile: Dieser Ansatz

- ist informatiknah und führt zu einer verständnisvollen Nutzung und Konstruktion von Informatiksystemen, wie sie auch bei kommerziellen Entwicklungen angestrebt wird,
- führt schon früh zu einem projektorientierten Vorgehen mit Teamarbeit und damit in die zentrale Arbeitsweise von Informatikern ein,
- ist meist fächerübergreifend; er schärft daher über die (Kern-)Informatik hinaus den Blick für die Anwendungen und vermittelt Kompetenzen auch aus dem gesellschaftlichen Umfeld. Umgekehrt beansprucht er vielfältige Kompetenzen von den Schülern (nicht nur aus der Informatik), so daß auch Schüler ohne Informatikvorkenntnisse sinnvolle Beiträge zum Unterricht leisten können,
- eignet sich im Sinne des Spiralprinzips als durchgängige Unterrichtslinie in beiden Sekundarstufen, an der sich alle Aktivitäten orientieren können. Lehmann [LOGIN 15,1 (1995) 29-37] spricht gar von einer fundamentalen Idee „Komplexes System“ verbunden mit den zugehörigen Ideen zur Bewältigung dieser Komplexität.

Nachteile: Der Ansatz

- ist intellektuell anspruchsvoll. Er erfordert erhebliche Vorarbeiten vom Lehrer im Bezug auf das zu analysierende Produkt und seine Analyse-freundliche Aufbereitung und Strukturierung,
- löst das Problem unterschiedlicher Vorkenntnisse nicht.

Beispiel: [Lehmann et al. in LOGIN 15,1 (1995) 38-50] Ausgangspunkt der Systemanalyse ist ein Programm für das Spiel „Mastermind“, das in einem früheren Kurs entwickelt wurde und in stark überarbeiteter Form vorliegt: Der Programmtext ist in PASCAL geschrieben, etwa zwei Seiten lang und unter Verwendung von Units mit pragmatischen Bezeichnern übersichtlich modularisiert worden. Zum Unterrichtsverlauf:

Vorübung: Die Schüler spielen das Spiel „Mastermind“ im Original. Anfertigung einer Verlaufsbeschreibung. Erarbeiten der Darstellungsmittel für Struktogramme und Übertragung der Verlaufsbeschreibung in diese Notation.

Blick auf das System: Einführung in die Computerbedienung. Spielen von Mastermind mit dem Programm. Wiederum Anfertigung eines Struktogramms, diesmal für den Programmverlauf. Vergleich der beiden Struktogramme. Bewertung von Vor- und Nachteilen von Computerspielen.

Blick in das System: Vergleich von Struktogramm und Programmtext. Zuordnung von Struktogrammteilen zu Units oder Prozeduren. Klärung der Bedeutung bestimmter Sprachenelemente von PASCAL. Prozedurkonzept.

Modifikation des Systems: Erweiterung des Programms um weitere Eingabe- und Spielfunktionen: Ausgabe einer Spielanleitung, Änderung des Ratemodus usw.

Konstruktion des Systems: Entwurf und Implementierung eines ähnlichen Spiels unter Wiederverwendung von Modulen.

Der Zugang über Lern- bzw. Programmierumgebungen.

Merkmale: Unter Verwendung sehr leistungsfähiger kommerzieller, meist objektorientierter Entwicklungsumgebungen, wie Hypertext-/Hypercard-Systeme oder Programmiermodelle (z.B. LOGO, Roboter NIKI), werden erste Erfahrungen im Umgang mit Informatiksystemen gesammelt. Die Objekte und Operationen orientieren sich in Form und Wirkungsweise an in der Alltagswelt vorkommenden Gegenständen und Handlungen. Die Tätigkeit des Programmierens reduziert sich weitgehend auf das Konfigurieren, d.h. auf das Zuschneidern der vorhandenen Umgebung auf die persönlichen Bedürfnisse. Geringfügige verbleibende Programmiertätigkeiten erfolgen durch ein Teach-in-Verfahren über einen Makrorecorder mittels Aufzeichnung von manuellen Aktivitätsfolgen verbunden mit einer automatischen Programmerzeugung.

Vorteile: Der Ansatz

- orientiert sich vermöge seines objektorientierten Zugangs, sofern solche Umgebungen gewählt werden, an den natürlichen kognitiven Voraussetzungen von Anfängern,
- bietet je nach Wahl der Entwicklungsumgebung die Möglichkeit, mit nur geringen Kenntnissen leistungsfähige Produkte zu konstruieren, die ein professionelles Äußeres besitzen,
- realisiert einen stufenweisen Einstieg in die Programmierung, wobei der Einblick zu

jeder Zeit begrenzt werden kann, wenn weitergehende Programmiererfahrungen, etwa im Rahmen der informationstechnischen Grundbildung ITG, nicht notwendig erscheinen,
- ist sehr modern, weil er mehrere zukunftsweisende Entwicklungsrichtungen (u.a. Objektorientiertheit, Multimedia) in sich vereinigt.

Nachteile: Der Ansatz
- beschränkt die Tiefe, bis zur der Informatikkenntnisse gewonnen werden können, da die den Entwicklungsumgebungen zugrundeliegenden Programmiersprachen zur Zeit nicht alle unterrichtsrelevanten Konzepte abdecken. Es fehlen zum Teil höhere Datentypen wie Listen, Bäume, Records oder gewisse notationelle Zwänge, etwa Datentypen vor ihrer Verwendung deklarieren zu müssen. Der Ansatz erzwingt daher zur Zeit einen späteren Umstieg auf eine „richtige“ Programmiersprache. Diese Nachteile fallen bei der ITG jedoch weniger ins Gewicht, da es hier sowieso nicht auf tiefergehendes Programmierverständnis ankommt,
- löst das Problem unterschiedlicher Vorkenntnisse nur teilweise.

Beispiel: [Glöckler/Spengler in LOGIN 15,1 (1995) 51-61] Erstellen einer Kurskartei mit dem Hypercard-System ToolBook, in der Informationen über die Teilnehmer des Informatikkurses gespeichert sind. Zum Unterrichtsverlauf:
- Vorgabe einer leeren elektronischen Musterkarteikarte, Eintragung der Daten eines Kursteilnehmers oder des Lehrers, Vertrautwerden mit Maus, Anklicken, Menüleiste, Aktionen, Buttons usw.
- Anlegen weiterer Karteikarten, Überlegungen zur endgültigen Gestaltung der Karteikarten, zum Informationsumfang, zur Sicherheit der Daten usw.
- Freies Spiel mit dem Hypercard-System, Nachvollziehen einer guided tour mit interaktiven Übungen, Entdecken weiterer Objekte, Funktionalitäten und Gestaltungsmöglichkeiten
- Systematische Behandlung der zuvor entdeckten Elemente, darunter der Objekte (Bild, Button, Text) und ihrer Eigenschaften und Operationen (Aktivieren, Vergrößern, Verkleinern, Zerren, Schriftart ändern)
- Erste Einführung in die Möglichkeiten von Buttons, Verbindung von Karten, Ausführung von Aktionen
- Aufzeichnen von Aktionsfolgen mit dem Makrorecorder, Analyse der automatisch generierten Programme (Skripte), Herstellen eines Zusammenhangs zwischen Programmelementen und Aktionen
- Überblick über die zugrundeliegende Programmiersprache, Struktogramme, systematische Einführung in einige wichtige Programmiersprachelemente
- Algorithmisierung mittels expliziter Skripterstellung, Suchen, Sortieren
- Gedanken zur Modellbildung, Zusammenhang zwischen Kartei und Realität, Datenschutz.

Exkurs

Kognitive Aspekte objektorientierter Programmierung

Bei Erwachsenen ebenso wie bei kleinen Kindern kann man das typisch menschliche Verhalten beobachten, alle Dinge zunächst danach zu beurteilen, was man mit ihnen machen kann. So ist z.B. ein Schraubenzieher ein Werkzeug, mit dem man in erster Linie Schrauben lösen und festziehen kann; als Vertreter einer Klasse mit allgemeineren Eigenschaften wie „länglich", „spitz" kann man ihn zur Not aber auch als Brechstange, Meißel, Bohrer oder Stichwaffe verwenden. Umgekehrt unterscheidet man diverse Teilklassen mit spezielleren Merkmalen: Schraubenzieher für Schlitzschrauben, für Kreuzschlitzschrauben, mit Einrichtung zur Spannungsprüfung usw. Gerade Kleinkinder besitzen in hohem Maße die Fähigkeit zwischen den Ebenen dieser Hierarchie hin und her zu wechseln und dabei Eigenschaften und Operationen von Objekten zu entdecken, die sie für einen völlig anderen als ihren vorbestimmten Zweck geeignet erscheinen lassen. An diesem Beispiel konkretisiert sich bereits die klassische objektorientierte Denkweise. Ausgehend von einer Reihe von Untersuchungen zur Problemlösefähigkeit und den verfügbaren Problemlösemethoden von Menschen in der ersten Hälfte dieses Jahrhunderts (u.a. K. Duncker, O. Selz und M. Wertheimer) gibt es in der Psychologie eine Vielzahl von Untersuchungen über die Wahrnehmung von Objekten und die Repräsentation von Wissen bei Kindern und Erwachsenen sowie darüber, wie dieses Wissen menschliche Entscheidungen und Handlungen leitet. Alle diese Untersuchungen belegen, daß Menschen Objekte überwiegend anhand der Handlungen identifizieren, die mit ihnen möglich sind, als anhand äußerlicher Eigenschaften wie Farbe oder Form. Einige dieser Resultate erscheinen heute im Zusammenhang mit der objektorientierten Programmierung in neuem Licht. Sie belegen einerseits, daß sich diese Form der Programmierung in besonderer Weise harmonisch den elementaren kognitiven Prozessen unterordnet, die beim Denken, Erkennen und Problemlösen im menschlichen Gehirn ablaufen, zeigen andererseits aber auch gewisse Schwächen dieser Denkweise.

Quelle: A. Schwill: Programmierstile im Anfangsunterricht, in: Innovative Konzepte für die Ausbildung (S. Schubert, ed.), 6. GI-Fachtagung Informatik und Schule (1995) 178-187

Der projektorientierte fächerübergreifende Zugang.

Merkmale: Ein umfangreiches fächerübergreifendes Projekt wird innerhalb einer Projektwoche oder eines Kurshalbjahres behandelt. Die Aufgabenstellung knüpft an den Erlebnis- und Erfahrungsbereich der Schüler an und enthält als eine Komponente innerhalb eines größeren Zusammenhangs die Anwendung eines Informatiksystems, wozu auch die Berücksichtigung sozialer und wirtschaftlicher Auswirkungen sowie die Diskussion über Chancen und Risiken der Informationstechniken gehört. Der kerninformatische Anteil der Projekte liegt oftmals unter 50%, möglicherweise enthalten Teile des Projekts überhaupt keine Berührungspunkte zur Informatik.

Vorteile: Der Ansatz

- spricht Schülerinnen und Schüler gleichermaßen an, fördert deren Interesse und fordert außerinformatische Kompetenzen,
- löst damit das Problem unterschiedlicher Vorkenntnisse weitgehend, indem er die unterschiedlichen Kompetenzen der Anfänger aufgreifen und ihnen adäquate Aufgaben

zuordnen kann,
- basiert auf projektorientierter Arbeitsform mit Teamarbeit und damit auf der zentralen Arbeitsweise von Informatikern,
- erweitert durch Eingliederung außerinformatischer Bezüge den Gesichtskreis der Schüler im Sinne einer Allgemeinbildung von Informatikunterricht,
- schafft ein Bewußtsein für die Auswirkungen der Informatik.

Der projektorientierte Ansatz eignet sich daher besonders im Rahmen der informationstechnischen Grundbildung.

Nachteile: Der Ansatz
- läßt je nach Auswahl des Themas den kerninformatikischen Elementen meist nur geringen Raum,
- betont die Nutzung von Informatiksystemen anstelle eines Verständnisses für die Funktionsweise und Konstruktion von Informatiksystemen.

Beispiel: [Koerber/Peters in LOGIN 15,1 (1995) 17-21] Einstieg in die informatische Bildung über das Projekt „Kurszeitung“, bei dem ein Magazin mit informatikbezogenen Inhalten mit Hilfe eines Computers in professioneller Form von den Schülern erstellt werden soll. Zeitbedarf: etwa fünf volle Tage im Rahmen einer Projektwoche. Zum Unterrichtsverlauf:
- Einführung in das Projekt, Planung der Arbeiten, Themenauswahl für die Zeitung, Zielgruppenanalyse, erste Recherche-Arbeiten
- Einführung in das Werkzeug Computer und in Textverarbeitungssysteme, Erwerb von Fertigkeiten im Umgang mit dem System
- Entwurf der geplanten Artikel, Erfassung der ersten Versionen der Rohtexte, Überlegungen zur Textgestaltung
- Weiterentwicklung der Texte, Korrigieren, Textbearbeitungsfunktionen beherrschen, Diskussion über Veränderungen der Arbeitswelt
- Bildauswahl, -bearbeitung und -integration, Erwerb weiterer Kenntnisse im Zusammenhang mit DTP-Systemen und deren Funktionalität
- Endredaktion, Produktion der Zeitung
- Veröffentlichung der Zeitung, Verteilung in der Schule, Auswertung von Rückmeldungen, Podiumsdiskussion
- Abschlußbesprechung, Reflektion, Besichtigung einer Druckerei oder eines graphischen Betriebs

TEIL F

Projektunterricht

1 Unterrichtsformen

Unterricht wird wesentlich durch die Wechselwirkungen zwischen Lehrer und Schülern sowie der Schüler untereinander geprägt. Diese Wechselwirkungen kann man bezgl. zweier Aspekte genauer klassifizieren, bezgl. der *Sozialform*, in der der Unterricht stattfindet, und bezgl. der *Aktivitäten*, die die Schüler während des Unterrichts entfalten können oder sollen (vgl. das Buch von E.C. Wittmann).

Hinsichtlich der Sozialform unterscheidet man:

(S1) *Unterricht im Klassenverband:* Alle Schüler nehmen gleichberechtigt am Unterricht teil und können miteinander kommunizieren und in Wechselwirkung treten.

(S2) *Gruppenunterricht:* Die Schüler sind in Gruppen aufgeteilt. Kommunikation vollzieht sich nur innerhalb der Gruppe, aber nicht gruppenübergreifend.

(S3) *Einzelunterricht:* Die Schüler sind voneinander isoliert. Wechselwirkungen sind nicht möglich oder nicht erlaubt.

Bezüglich der von den Schüler erwarteten bzw. entfalteten Aktivitäten unterscheidet man:

(A1) *Kommunizierende Form:* Die Aktivitäten der Schüler beschränken sich im wesentlichen auf die Aufnahme des vom Lehrer präsentierten Stoffes.

(A2) *Gelenktes Entdecken:* Die Schüler gestalten den Verlauf des Unterrichts durch ihr Handeln mit; durch Hilfen des Lehrers werden entdeckende Aktivitäten angestoßen.

(A3) *Freies Forschen:* Der Lehrer gibt lediglich Anregungen; im wesentlichen bestimmen die Aktivitäten der Schüler den Unterrichtsverlauf.

Durch Kombination von Sozial- und Aktionsform gewinnt man aus dieser Klassifikation insgesamt neun verschiedene Unterrichtsformen (S1,A1) bis (S3,A3). Hierbei handelt es sich um „Reinformen“, die in dieser reinen Form jedoch nur selten im Unterricht auftreten. Üblich sind vielmehr Mischformen.

Für einige der neun Unterrichtsformen sind feste Bezeichnungen üblich, z.B.:

(S1,A1): **Frontalunterricht**.

Beispiel: Der Lehrer trägt einen bestimmten Stoff vor (Vorlesungsstil).

(S1,A2): **Fragend-entwickelnder Unterricht**.

Beispiel: Der Lehrer entwickelt mit den Schülern einen Algorithmus zum Sortieren, indem er die Beiträge der Schüler aufgreift und in die gewünschte Richtung kanalisiert.

(S1,A3): **Freies Unterrichtsgespräch**.

Beispiel: Der Lehrer stellt der gesamten Klasse ein offenes Problem, also ein Problem, das nicht explizit formuliert ist und sich in verschiedene Richtungen präzisieren

läßt, z.B.: Wir wollen unsere Schulbücherei automatisieren. Was ist zu tun? Den weiteren Lösungsweg begleitet der Lehrer weitgehend passiv, er fungiert vor allem als „Rettungsanker“.

Von den möglichen Unterrichtsformen greifen wir im folgenden Abschnitt exemplarisch eine heraus, die im Informatikunterricht eine besondere Rolle spielt, den Projektunterricht.

2 Projektunterricht

Der Projektunterricht läßt sich als Mischung der Unterrichtsformen gem. Abb. 1 einordnen. Es bestehen also große Möglichkeiten für Wechselwirkungen der Schüler untereinander sowie viel Freiraum für eigene Aktivitäten.

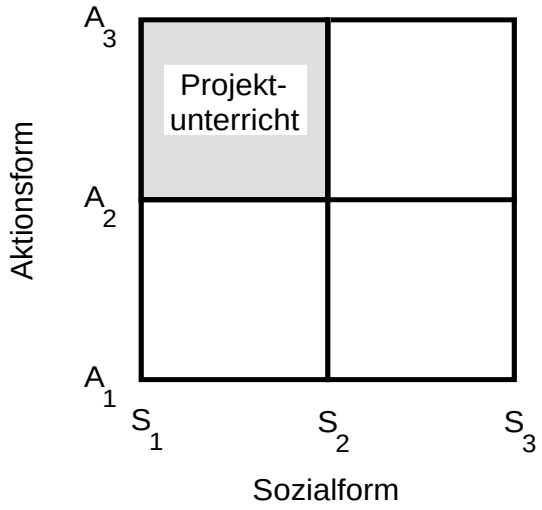


Abb. 1: Einordnung des Projektunterrichts

2.1 Pädagogische Aspekte des Projektunterrichts

Der Begriff des Projekts im Bereich von Schule und Unterricht geht auf Arbeiten von J. Dewey und W.H. Kilpatrick in den 20er Jahren zurück. Sie definieren ein Projekt als

> „planvolles Handeln von ganzem Herzen, das in einer sozialen Umgebung stattfindet“.

Etwas sachlicher spricht man von einem Projekt

> bei einer längeren, fächerübergreifenden Unterrichtseinheit, die durch Selbstorganisation der Lerngruppe gekennzeichnet ist und bei der der Arbeits- und Lernprozeß ebenso wichtig ist wie das Ergebnis oder Produkt, das am Ende des Projekts steht.

Nach H. Gudjons sind folgende Merkmale für den Projektunterricht charakteristisch:

- Situationsbezug und Lebensweltorientierung.
  Projektthemen entstammen der Lebenswelt der Schüler und sind inhaltlich nicht an Fachwissenschaften und somit auch nicht an Schulfächer gebunden.
- Orientierung an Interessen der Beteiligten.
  Wünsche, Bedürfnisse und Abneigungen der Projektbeteiligten (Lehrer *und* Schüler) beeinflussen den Projektverlauf. Nicht immer sind die Bedürfnisse von Lehrern und Schülern in Einklang. Vom Geschick des Lehrers hängt es dann ggf. ab, in den Schülern für seinen Vorschlag Interesse zu wecken.
- Selbstorganisation und Selbstverantwortung.
  Dies ist eines der wichtigsten Merkmale, das den Projektunterricht vom traditionellen Unterricht abgrenzt. Schüler und Lehrer bestimmen gleichberechtigt Ziel, Planung, Durchführung und Bewertung des Projekts. Wichtige Elemente zur Realisierung dieses Merkmals sind regelmäßig eingeschobene Reflexionsphasen (sog. *Fixpunkte*), bei denen sich die Schüler über den Stand ihrer Aktivitäten unterrichten und die weiteren Schritte planen. Ferner sollten diese Phasen zur Diskussion über gruppeninterne Prozesse, wie z.B. den gegenseitigen Umgang, Diskussionsstile, Sympathien und Antipathien genutzt werden.
- Gesellschaftliche Praxisrelevanz.
  In Projekten soll die Wirklichkeit nicht nur beobachtet, gespeichert, analysiert oder simuliert sondern auch *verändert* werden, und seien die Veränderungen auch noch so klein. Dieser Aspekt hilft zu verhindern, daß Projektarbeit zur „Bastelarbeit in der Dachkammer“ degeneriert.
- Zielgerichtete Projektplanung.
  Projektarbeit ist kein Lernen mit offenem Ende; stets steht am Ende ein gewisses Ziel, auf das man sich durch fortlaufende Planung und Korrektur bisheriger Aktivitäten zubewegt und nach dessen Erreichen das Projekt endet.
- Produktorientierung.
  Am Schluß des Projekts steht nicht nur wie beim traditionellen Unterricht ein schwer bezifferbarer Lernerfolg sondern vor allem ein vorzeigbares Produkt (z.B. ein Film, ein Bericht, ein Modell, ein Programm mit Dokumentation), das der Öffentlichkeit zugänglich gemacht wird und sich der öffentlichen Bewertung und Kritik stellen muß.
- Einbeziehen vieler Sinne.
  Dieses Merkmal betrifft in erster Linie die Wiedervereinigung von Denken und Handeln (Lernen als ganzer Mensch), also das Entwickeln und Einbeziehen körperlicher Fähigkeiten und handwerklicher Fertigkeiten in den Unterricht, der traditionell vor allem durch geistige Tätigkeit geprägt ist.
- Soziales Lernen.
  Gemeinsames Lernen und Handeln in Gruppen durch Kommunikation der Schüler untereinander und mit dem Lehrer als gleichberechtigtes Mitglied. Beim Projektunterricht ist nicht nur das Ziel von Bedeutung sondern auch der Weg dorthin. So kann ein Projekt

auch dann ein Erfolg sein, wenn das Ziel nicht erreicht wird, die Schüler aber gelernt haben, Konflikte zu lösen und kooperativ zu arbeiten.
Dieser Aspekt gewann vor allem Ende der 60er Jahre zunehmende Bedeutung, als der Projektunterricht durch die Bestrebungen zur Reform des allgemeinbildenden Schulwesens systemkritischen Charakter erlangte: „Projekte sollten durch soziales Lernen ein Gegengewicht zum traditionellen leistungsorientierten Lernen schaffen“. Dies hat schließlich dazu geführt, daß ein projekt*orientierter* Unterricht (s.u.) heute in allen Schulformen seinen festen Platz in Form von Projekttagen und -wochen gefunden hat.

- Interdisziplinarität.
Projektunterricht ist fächerübergreifend, was nicht heißen soll, daß das Thema nicht einem Schwerpunktfach zugeordnet werden kann. Vielmehr sollen auch andere (benachbarte) Wissenschaften auf ihre Beiträge zum Projektthema hin analysiert werden.

Offenbar ist es im traditionellen Bildungssystem nahezu unmöglich, einen Projekt- unterricht durchzuführen, der *alle* obigen Merkmale erfüllt. Ursachen sind vor allem die gesetzlichen Rahmenbedingungen wie Curricula, fachliche und zeitliche Zergliederung des Unterrichts, Bewertungsmaßstäbe sowie das Lehrer-Schüler-Verhältnis. Werden dennoch Ansätze des Projektunterrichts realisiert, so spricht man zur Unterscheidung von *projektorientiertem Unterricht*.

Es folgen zwei Beispiele für Projektunterricht, die sich weitgehend den o.g. Merkmalen unterordnen.

Beispiele:

1) Das wohl berühmteste Beispiel für Projektunterricht hat E. Collings 1923 veröffentlicht, das sog. **Typhusprojekt** (aus: D. Hänsel):

> „Das Typhusprojekt nimmt von einem Problem, das die Schüler bewegt, seinen Ausgang, nämlich von der Frage, warum Angehörige der Familie Smith, der zwei ihrer Klassenkameraden angehören, regelmäßig im Herbst an Typhus erkranken. Die Schüler versuchen zunächst eine gedankliche Analyse des Problems, indem sie Hypothesen über mögliche Ursachen anstellen (Brunnenwasser, verdorbene Milch, Fliegen). Die Schüler erkunden dann die realen Lebensbedingungen der Familie Smith, indem sie ihr einen Besuch abstatten. Sie identifizieren nach diesem Besuch die Fliegen als mögliche Ursache des Typhus und versuchen, eine reale Problemlösung zu erarbeiten. Diese Problemlösung, die unter Verwendung von Literatur und durch Befragung eines Experten erfolgt, mündet u.a. im Bau einer Fliegenfalle und eines Müllkübels mit Deckel, die Herrn Smith mit einem Bericht übermittelt werden. Herr Smith wendet diese Lösung in der Praxis an und hat damit Erfolg. Sein Haus bleibt künftig von Fliegen und damit von Typhus verschont.“

2) R. Schweinegruber beschreibt ein Mini-Projekt von G. Dommermuth:

> „Ich mußte eine Kollegin in einer fünften Klasse vertreten, die mir völlig unbekannt war, und versuchte, schlecht und recht im Unterricht weiterzumachen. Nach einer Viertelstunde rief plötzlich ein kleiner Portugiese: ‚Schauen Sie mal, die Straßenlampen brennen immer noch, wir haben doch Energiekrise!‘

Tatsächlich, dabei war es schon elf Uhr, wie mir sofort ein Dutzend Knirpse versicherte. Es folgte eine kurze Debatte über Erscheinungen, Ursachen und Folgen der Energiekrise. Bei der Suche nach Abhilfemöglichkeiten landeten die Schüler wieder bei der verschwenderischen Gemeinde, die ihre Straßenlampen brennen ließ.
Ich fragte: ‚Was kann man denn da machen?'. Die Antwort kam spontan: ‚Wir schreiben einen Brief an den Bürgermeister!'. Ein Schüler schrieb, die ganze Klasse diktierte. Der Brief war gerade fertig, als es läutete. Die Schüler waren so eifrig bei der Sache, daß sie die Pause gar nicht beachteten. Sie stellten sich ungebeten in einer langen Schlange auf - denn jeder wollte seine Unterschrift unter den Brief setzen. ... Was haben nun die Schüler gelernt? Wahrnehmen, Beschreiben, Diskutieren, Formulieren, Rechtschreiben. Ich brauchte während der ganzen Zeit keine zehn Sätze zu sagen. Auch die mustergültig vorbereitete Stunde eines Superpädagogen (ich war nicht vorbereitet und bin ein ziemlich mittelmäßiger Lehrer) hätte keine besseren Lernergebnisse gebracht."

2.2 Informatische Aspekte des Projektunterrichts

Der Informatikunterricht scheint für die Projektmethode besonders geeignet. Anders als in anderen Fächern, wo Projektunterricht nur als Unterrichtsmethode fungiert, *keinen* unmittelbaren Bezug zum Fach besitzt und daher leicht aufgesetzt wirken kann, ist er innerhalb der Informatik auch *wissenschaftlich verankert*. Platt formuliert: Weil Informatiker projektartig arbeiten, gehört Projektarbeit auch in den Informatikunterricht. Daher bietet sich hier die einzigartige Möglichkeit, Projektunterricht als Lehr- und Wissenschaftsmethode organisch zu verbinden.
Diese Überlegung verquickt allerdings den pädagogischen und den informatischen Projektbegriff. In der Informatik dominiert projektartiges Vorgehen bei der Softwareentwicklung, wo es als Methode zur Verbesserung der Produkte und zur Leistungs- und Effizienzsteigerung eingesetzt wird. Offenbar sind die Intentionen des pädagogischen und des informatischen Projektbegriffs damit diametral entgegengesetzt, denn diese Leistungsorientierung soll durch ein Projekt im pädagogischen Sinne (soziales Lernen) ja gerade abgemildert werden.
Diese unterschiedlichen Auffassungen des Projektbegriffs konkretisieren sich auch in der Rolle der Teamarbeit: Während die Teammitglieder in pädagogischen Projekten für die gesamte Projektlaufzeit konsequent kooperieren, erledigen die Mitglieder informatischer Projekte ihre Aufgabe über weite Strecken unabhängig voneinander in Einzelarbeit. Ja, es ist sogar das Ziel gängiger Software Engineering-Methoden (z.B. Modularisierung), die Gruppe möglichst frühzeitig zu zerschlagen und die Teammitglieder voneinander zu trennen, um im Hinblick auf die gewünschte Leistungssteigerung eine möglichst optimale Parallelisierung der Entwicklungsaktivitäten zu erzielen.
Konsequenz: Teilnehmer an pädagogischen Projekten bleiben *Generalisten*, behalten stets den Überblick über das Gesamtprojekt und können ihren eigenen Beitrag einordnen; Teilnehmer an informatischen Projekten werden zu *Spezialisten* ausgebildet, die nur noch einen sehr groben Überblick über das Gesamtsystem besitzen können. Dieser Effekt ist pädagogisch bedenklich. Strategien zur Vermeidung solcher negativen Begleiterscheinungen durch geschickte Zusammenstellung von Modulteams sowie durch weitere orga-

nisatorische Maßnahmen behandeln wir unten.

Softwaretechnologische Aspekte.
Traditionell konkretisieren sich die Aktivitäten bei der Softwareentwicklung anhand des Software Life cycle, dessen Standardform Abb. 2 zeigt. Wir wollen nur grob die einzelnen Phasen skizzieren. Einzelheiten werden (hoffentlich) in anderen Vorlesungen der Kerninformatik und in Programmierpraktika vorgestellt (s. auch Stichwort „Software-Engineering“ im Duden Informatik).

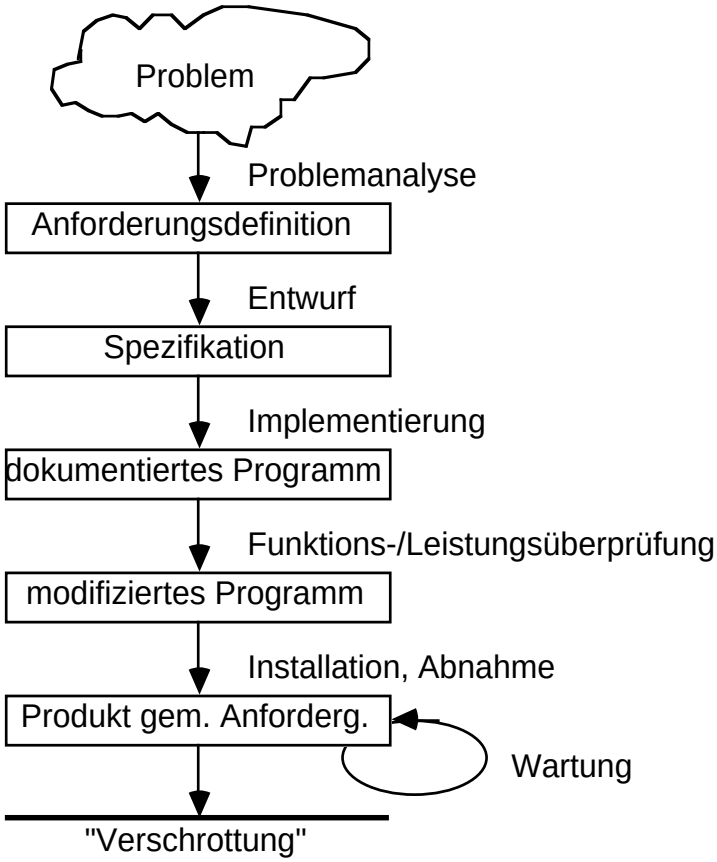


Abb. 2: Software life cycle

Gehen wir die einzelnen Phasen im Überblick durch:

- Problemanalyse.
  In dieser Phase werden das zu lösende Problem und alle wichtigen Umgebungsbedingungen vollständig und eindeutig erfaßt. Ferner wird die Durchführbarkeit des Projekts untersucht. Die Phase gliedert sich in vier Teilphasen:
  = Istanalyse: Untersuchung und Beschreibung des vorliegenden Systems durch Betrachtung seiner Komponenten, ihrer Funktionen und ihres Zusammenwirkens.

= Sollkonzeptentwicklung: Festlegung der Anforderungen an die zu erstellende Software durch Angabe u.a. des Benutzermodells, der Basismaschine, der Benutzermaschine usw. Hierbei werden noch keine Aussagen zur Realisierung gemacht.

= Durchführbarkeitsstudie: Sie liefert Erkenntnisse darüber, ob die Vorstellungen über das Softwareprodukt überhaupt realisiert werden können, ob sie prinzipiell durchführbar (Gibt es etwa Teile, die nicht berechenbar sind?) und ökonomisch vertretbar sind.

= Projektplanung: Erstellung von Zeitplänen, Zusammenstellung von Teams, Verteilung des Personals und der übrigen Hilfsmittel.

Das gesamte Ergebnis der Problemanalyse wird in der Anforderungsdefinition festgehalten, die Bestandteil des Vertrags zwischen Auftraggeber und Softwareproduzent wird.

- Entwurf.

  Während in der Problemanalysephase die Eigenschaften des Softwareprodukts ohne Hinweise auf ihre Realisierung beschrieben werden, entwickeln die Programmierer in der Entwurfsphase ein Modell des Systems, das, umgesetzt in ein Programm, die Anforderungen erfüllt. Hierzu wird das komplexe Gesamtsystem fortlaufend in unabhängig voneinander realisierbare und in ihrem Zusammenwirken überschaubare Einzelbausteine (Module) zerlegt und die Funktionen dieser Bausteine und ihre Schnittstellen spezifiziert. Üblich ist die hierarchische Modularisierung, für die es zwei verschiedene Reinformen gibt, die Top-down-Methode und die Bottom-up-Methode.

  Das Ergebnis der Entwurfsaktivität ist die Spezifikation, in der für jedes Modul seine Funktion, seine Schnittstellen und Hinweise zur Anwendbarkeit, sowie ein Gesamtüberblick über die Abhängigkeit der einzelnen Module untereinander enthalten sind.

- Implementierung.

  Erstellung eines lauffähigen Programms, das in seinem Ein-/Ausgabeverhalten der Anforderungsdefinition entspricht. Das Ergebnis der Implementierung ist ein dokumentiertes Programm.

- Funktionsüberprüfung.

  Anhand der Anforderungsdefinition wird das Ein-/Ausgabeverhalten des Programms durch eine Kombination von Verifikation und Testen überprüft. Man beginnt mit dem Modultest und prüft anhand der Spezifikation, ob jedes Modul das vorgeschriebene Funktionsverhalten besitzt. Nach dem Zusammenbau der getesteten und verifizierten Module beginnt der Integrationstest, dann der Installationstest und schließlich der Abnahmetest.

- Leistungsüberprüfung.

  Nach der Korrektheitsüberprüfung müssen Leistungsmessungen (Laufzeit- und Speicherverhalten) durchgeführt werden.

- Installation, Abnahme.

  Einbettung des Programmsystems in die Systemumgebung des Auftraggebers, z.B. durch Anpassung an die speziellen Eigenschaften der Rechenanlage und die betriebli-

chen Gegebenheiten.

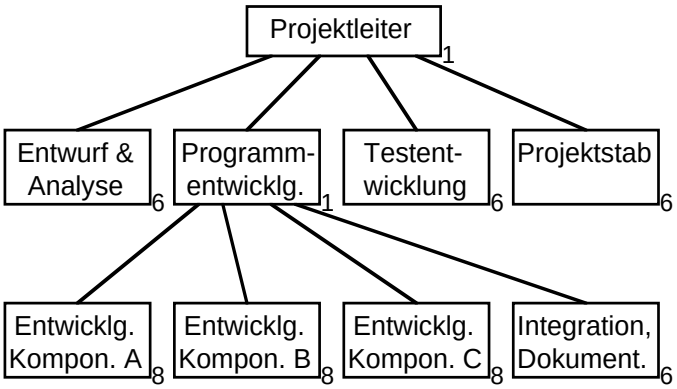
- Wartung.

  Befinden sich Programme in Betrieb, so werden an ihnen häufig Veränderungen und Erweiterungen vorgenommen. Mögliche Wartungsarbeiten sind der Austausch von bestimmten Algorithmen durch leistungsfähigere, die Anpassung an sich ändernde gesetzliche Bestimmungen oder Tarifverträge, die Beseitigung von Fehlern usw.

  Diese Phase fällt i.a. nicht mehr in den Rahmen eines Schulprojekts.

Teamarbeit.

Eines der wichtigsten Lernziele des Projektunterrichts ist die Fähigkeit zu kooperativer Arbeit und zur Konfliktlösung in Gruppen. Die klassischen Teamstrukturen bei Informatikprojekten, etwa die hierarchische Organisation (Beispiel in Abb. 3) oder das Chef-Programmierer-Team (Beispiel in Abb. 4) scheinen unter diesem Aspekt für die Schule ungeeignet, da sie entweder zwischen den Schülern hierarchische Beziehungen herstellen oder auf Expertentum einzelner Schüler basieren.



Projektleiter: Verbindungen zum Auftraggeber, Management des Projekts, Einrichtg. von Arbeitsgruppen, Einholen von Fortschrittsberichten

Projektstab: administrative Aufgaben wie Termin- und Kostenkontrolle, Schulungen, Anfertigung von Richtlinien

Entwurf&Analyse: Entwurf des Gesamtsystems, Herstellung der Entwicklungsdokumentation, Leistungsanalyse

Programmentwicklung: Implementierung, Funktions- und Leistungsüberprüfung sowie Dokumentation der Module

Testentwicklung: Integrations- und Installationstest

Abb. 3: Beispiel für ein hierarchische Teamorganisation für 50 Mitarbeiter

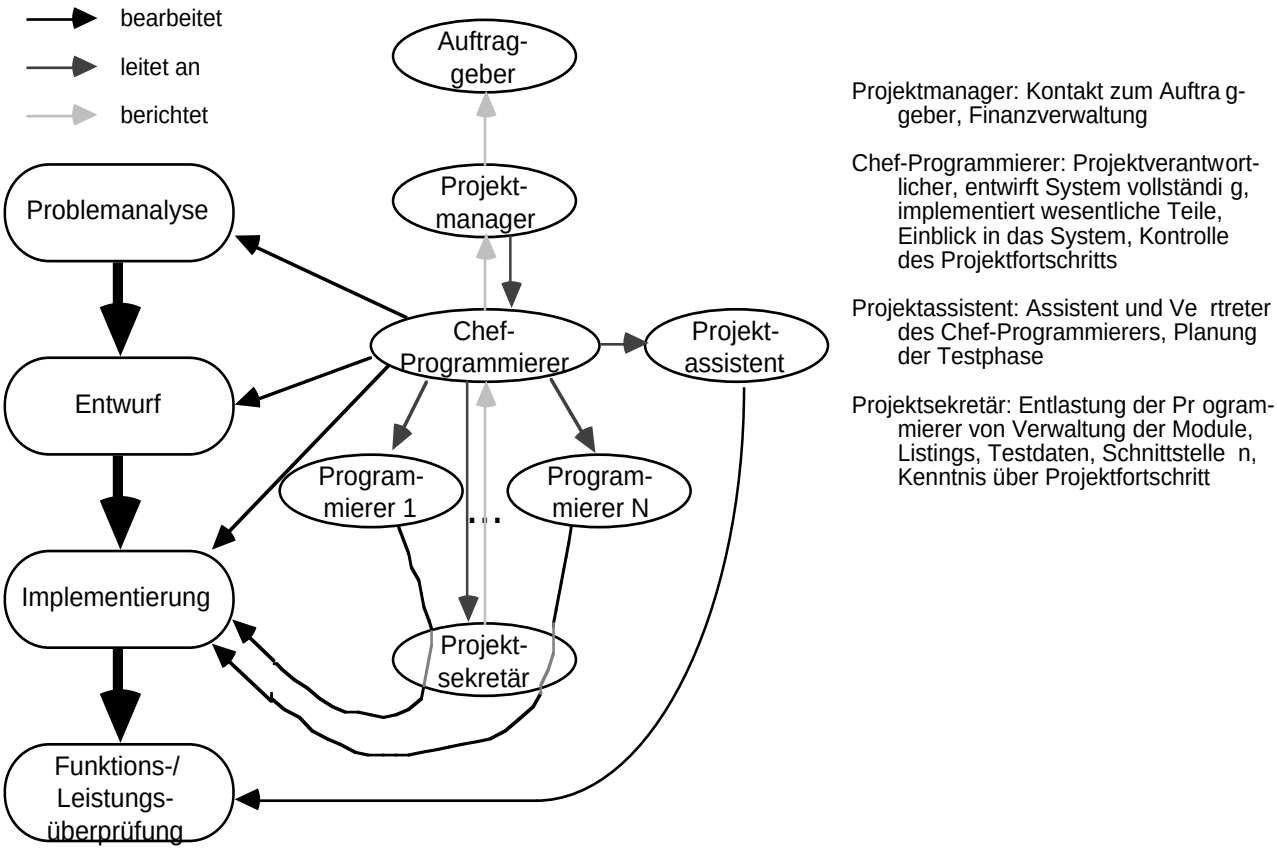


Abb. 4: Chef-Programmierer-Team (Empfehlung: N≤10)

Wir favorisieren stattdessen eine Teamstruktur,

- in der alle Teammitglieder annähernd gleichberechtigt sind,
- die möglichst über die gesamte Projektlaufzeit zwischen allen Teammitgliedern Berührungspunkte (Zwang zur Kommunikation, gemeinsame Ziele) erhält.

Folgendes Modell dient diesen Zielen:

1) Die Phasen „Problemanalyse“ und teilweise „Entwurf“ werden vom Team gemeinsam durchgeführt. Einzelfragen können im kleineren Kreis von kurzfristig zusammengestellten Ausschüssen und Arbeitsgruppen mit wechselnder Besetzung bearbeitet und die Ergebnisse der gesamten Gruppe zur Beschlußfassung vorgelegt werden.
2) Nach der Aufteilung des Gesamtsystems in Module werden Kleingruppen gebildet, die jeweils ein Modul bearbeiten. Jede dieser Gruppen ist für die Durchführung, Fertigstellung und Dokumentation ihrer Teilaufgabe allein zuständig und verantwortlich. In regelmäßig stattfindenden Sitzungen berichten die Teilnehmer dem Plenum über Lösungsansätze und den Stand ihrer Arbeit.

   Bei der Gruppenaufteilung sollten die Wünsche der Mitglieder zwar weitgehend beachtet werden, jedoch ist darauf zu achten, daß die Kleingruppen möglichst gut durch-

mischt sind, d.h., jedes Teammitglied gehört mindestens zwei Kleingruppen an, und je zwei Personen arbeiten in höchstens einer Kleingruppe zusammen. Ferner sollten innerhalb des Klassenverbandes bereits bestehende Cliquen nach Möglichkeit auf unterschiedliche Gruppen aufgeteilt werden, um die Fähigkeit zu Gruppenarbeit und Konfliktlösung zu trainieren.

3) Jedes Teammitglied stellt der gesamten Projektgruppe über seine modulspezifischen Aufgaben hinaus eine oder mehrere Dienstleistungen zur Verfügung. Mögliche Aufgaben, die je nach Größe des Projekts durch einen oder zwei Mitglieder übernommen werden können, sind:
   - *Rechnerbeauftragter*, eine mit den Feinheiten von Hardware und Betriebssoftware besonders vertraute Person,
   - *Projektüberwacher*, der die Arbeiten der einzelnen Gruppen koordiniert, Meilensteinpläne entwirft und einen Überblick über den Stand der Arbeitsgruppen besitzt. Der Projektüberwacher ist *kein* Projektleiter, er ist nicht verantwortlich und hat auch keine Weisungsbefugnis.
   - *Schnittstellenbeauftragter*, der die Einhaltung der Schnittstellen zwischen unterschiedlichen Modulen kontrolliert, Schnittstellenwünsche weiterleitet und zwischen den Gruppen vermittelt. Sind alle Module fertiggestellt und getestet, übernimmt der Schnittstellenbeauftragte die Integration des Gesamtsystems.
   - *Tester*, der die einzelnen Module entsprechend der Spezifikation testet und ggf. Testumgebungen entwirft.
   - *Dokumenteur*, der die phasenbegleitenden Dokumente der Arbeitsgruppen sammelt, systematisiert und den jeweiligen Phasenendbericht sowie schließlich die Gesamtdokumentation herausgibt.
   - *Kümmerer*, der sich um nicht-inhaltliche Kleinigkeiten sorgt, wie Herrichtung des Klassenraumes für eine Plenumssitzung (Kaffee, Kekse), Besorgung von Literatur, Planung einer Abschlußfeier etc.

Neben diesen Aufgaben fungiert jedes Teammitglied bei den Plenumssitzungen reihum als *Sitzungsleiter* und *Protokollführer*.

Themenwahl.

Gemäß der demokratischen Intention des Projektunterrichts sollten die Schüler bei der Wahl der Aufgabenstellung weitestgehend beteiligt werden. Zwei mögliche Vorgehensweisen:

- Der Lehrer stellt verschiedene Themen zur Auswahl. Jeder Themenvorschlag besteht mindestens aus der Aufgabenstellung, den Abgabeterminen für Anforderungsdefinition und Gesamtprodukt und einem Rahmen für das Sollkonzept gem. Muster in Abschnitt 2.3. Die Schüler einigen sich untereinander auf ein Thema.
  Vorteil dieses Verfahrens: Der Lehrer kann Projekte initiieren, die er bereits früher einmal erprobt hat und die so weit vorstrukturiert sind, daß der Projekterfolg gewährleistet

ist.

- Die Schüler machen in einem Brainstorming-Verfahren selbst Vorschläge und entscheiden sich unter Beteiligung des Lehrers für ein Thema.
  In diesem Fall geht dem Phasenmodell (Abb. 2) noch eine „Problemfindungsphase“ voraus, die bis zu einer Woche dauern kann. Hier muß der Lehrer für eine intensive Auseinandersetzung der Schüler mit den vorgeschlagenen Themen sorgen, um Fehlentscheidungen zu vermeiden. Eine Möglichkeit zur Durchführung des Entscheidungsprozesses besteht darin, die vorgeschlagenen Themen jeweils ein bis zwei Schülern zur Grobanalyse zu übertragen, die danach der Klasse in Vortragsform präsentiert wird (s. auch Beispiel 2 unten). Der Lehrer hat kraft seiner Erfahrung zu beurteilen, ob das gewählte Projektziel erreichbar erscheint oder ob schwer vorhersehbare Probleme bei der Durchführung auftauchen können. Von dem Ergebnis dieser Diskussion hängt es ab, ob alle Beteiligten später mit dem Ergebnis zufrieden sind.
  Es ist empfehlenswert, diesen Ansatz auf die letzten Klassen der Sekundarstufe II zu beschränken.

Beispiele:

1) B. Koerber hat einer 10. Klasse für ein Projekt im Informatikunterricht folgenden umfangreichen Katalog von Projektvorschlägen vorgelegt:
   Probleme aus Wirtschaft und Verwaltung:
   - Buchhaltung im schulinternen Getränkeladen
   - Buchhaltung und Statistik im schulinternen Eisladen
   - Simulation eines Versandhandels
   - Simulation eines Girodienstes.

   Probleme von Datenbanken und Informationssystemen:
   - Erstellen eines Fach-Auskunftssystems
   - Erstellen einer Lehrerdatei zur Information für Schüler
   - Berechnung der Zensuren zum Mittelstufenabschluß mit Beratung.

   Probleme aus künstlerischen Bereichen:
   - Kunst und Konstruktion mit Computern
   - Komponieren mit dem Computer
   - Dichten mit dem Computer.

   Probleme aus der Linguistik:
   - Sprachübersetzung einfacher Art mit Computern
   - Erstellen von Kreuzworträtseln mit Computer.

   Nach eingehender Diskussion und Informationssammlung wählten die Schüler schließlich das Projekt „Kunst und Konstruktion mit dem Computer“, wobei sie allgemeine graphische Muster, Tapeten und Stoffmuster sowie Stickmuster in unterschiedlichen Arten und Formen nach eigenen Vorgaben und zufallsgesteuert mit dem Rechner erzeugen wollten.

2) R. Buhse berichtet über ein Projekt (Dauer: 45 Stunden = 1 Halbjahr) in einem Grundkurs Informatik in der Klasse 13, bei dem die Schüler folgende Themenvorschläge entwickelten:
   - Spiele, Spielstrategien, Vergleich von Strategien
   - Simulation von Ökosystemen
   - Stundenplanerstellung
   - Schulverwaltung
   - Verbesserung der Betriebssoftware.

   Als Hausaufgabe sollten jeweils ein bis zwei Schüler Informationen zum Thema sammeln und in der nächsten Stunde vortragen. Schließlich entschied man sich für die Schulverwaltung und konzentrierte sich hierbei auf die Oberstufenverwaltung. Buhse erwähnt, daß bei dieser Entscheidung auch sachfremde Gründe mitgespielt haben. So hing es z.B. wesentlich von der Qualität der genannten Vorträge ab, ob ein Thema gleich von den Schülern abgelehnt wurde oder weiter „im Rennen“ blieb.

Es folgen einige Kriterien für die Auswahl von Projektthemen:

- Realitätsbezug: Themen sollten aus dem unmittelbaren Umfeld der Schüler stammen.
- Realisierbarkeit: Die Leistungsfähigkeit des Rechnersystems sollte für die Problemlösung ausreichen. (Diese Bedingung dürfte bei der Leistungsfähigkeit aktueller Rechnersysteme immer erfüllt sein.)
- Vorarbeiten: Der Zeitaufwand für Vorarbeiten (Literaturbeschaffung, Erarbeitung von weiteren Kenntnissen) sollte nicht zu hoch sein.
- Modularisierbarkeit: Das Thema sollte im Hinblick auf eine arbeitsteilige Erarbeitung der Lösung möglichst gut zerlegbar sein, wobei die Einzelbausteine noch hinreichend komplex sein sollten.
- Reduzierbarkeit/Erweiterbarkeit: Das Thema sollte zur Not so weit reduzierbar sein, daß die vereinfachte Version noch „eine runde Sache“ ist und alle Beteiligten zufriedenstellt. Andererseits sollten bei gutem Projektfortschritt Erweiterungsmöglichkeiten vorgesehen werden können.
- Software Life cycle: Möglichst alle Phasen des Software Life cycles sollten bei der Projektbearbeitung sichtbar werden. Dabei ist darauf zu achten, daß eine neue Phase erst begonnen wird, wenn die Schlußdokumente der vorherigen Phase vorliegen.
- Rahmen der Anforderungsdefinition: Die Problemstellung sollte noch genügend Freiraum für eigene Präzisierungen lassen und den Schülern Anreize zu selbständigen Forschungen und Entdeckungen geben. Die abgelieferte Anforderungsdefinition ist umgekehrt daraufhin zu überprüfen, ob sie die geforderte Verbindlichkeit besitzt und als Vertragsgrundlage geeignet wäre, ob also die Schüler alle Freiheiten und Unexaktheiten durch präzise Festlegungen eliminiert haben.

Leistungsbewertung.

(nach E. Lehmann in LOGIN 12,5/6 (1992)) Die traditionelle Form der Leistungsbewertung, die Klausur, ist weitgehend ungeeignet, um die innerhalb der Projektarbeit erworbenen spezifischen Fähigkeiten und Fertigkeiten zu überprüfen. Daher sind weitere schriftliche und mündliche Leistungselemente einzubeziehen, die während der Projektarbeit gewonnen werden können, etwa

- die fachliche Arbeit in den Untergruppen. Hier ist insbesondere der Schwierigkeitsgrad der übertragenen Aufgaben im Verhältnis zu den anderen Mitgliedern abzuschätzen,
- Fertigkeiten im Umgang mit dem Gerät, ressourcenschonende Arbeit mit dem Editor, sinnvolle Nutzung des Betriebssystems und von Tools,
- Reaktion auf Lehrer- und Schülerfragen, Kommunikations- und Integrationsfähigkeit, Bereitschaft zu beraten und sich beraten zu lassen, Diskussionsleitung und -protokollierung,
- Programmierkenntnisse, vor allem im Bezug auf die anderen Mitglieder, Einhaltung von Schnittstellen und anderer Verabredungen, Spezifikationstreue, angemessene Verwendung von Bausteinen, Originalität der Lösung,
- Aneignung zusätzlicher Kenntnisse, speziell auch bei den persönlichen Dienstleistungsaufgaben,
- effiziente Wahrnehmung der Dienstleistungsaufgaben an die Gruppe,
- Darstellung eigener Arbeitsergebnisse im Plenum, in den Untergruppen oder in der begleitenden Dokumentation.

Da die Leistungskriterien teilweise von den in anderen Unterrichtsfächern gewohnten abweichen, sollten die Schüler zu Beginn eines Projekts einen Überblick über die Bewertungskriterien erhalten. Darüber hinaus sind die abschließend vorgenommenen Bewertungen mit den Schülern zu diskutieren und zu rechtfertigen.

Nach einer Einführung in die Grundlagen der Informatik kann man pro Halbjahr ein Projekt durchführen.

Gruppenarbeit in der intensiven Form eines Projekts wird für die Schüler i.a. eine neue Erfahrung sein. Daher werden die Gruppenprozesse relativ breiten Raum einnehmen. Um Informatikinhalte und das Projektziel nicht zu vernachlässigen, startet man zweckmäßigerweise mit kleinen Projekten in Gruppen zu zwei oder drei und geht erst im Laufe der Zeit zu größeren Projekten mit Gruppen zu 8-10 Schüler über.

Projektablauf.

Abb. 5 zeigt den prinzipiellen Ablauf eines Projekts noch einmal in der Gesamtschau. Hierbei sind die pädogogischen und die informatischen Aspekte zu einer Einheit verknüpft.

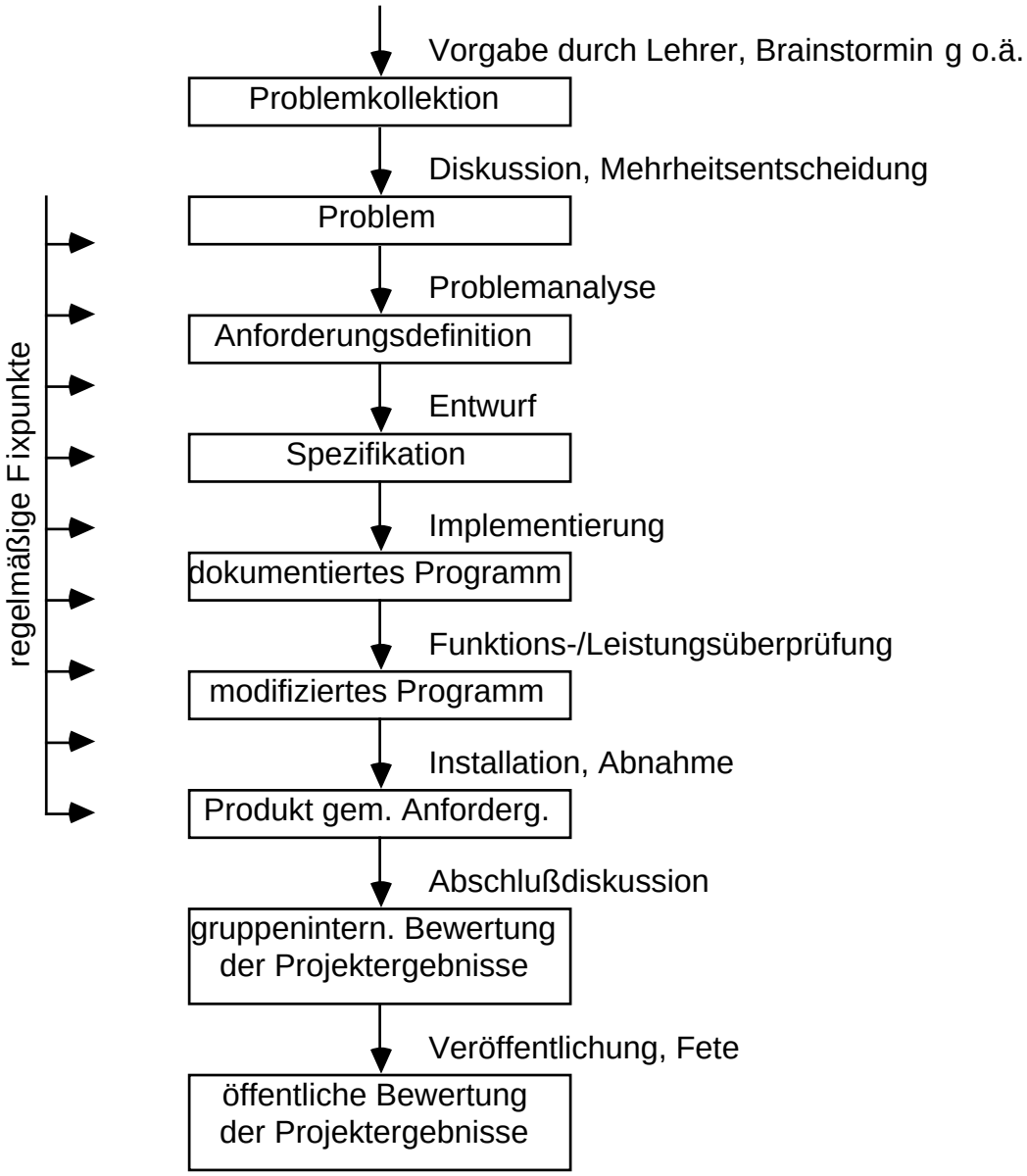


Abb. 5: Projektverlauf

2.3 Projektbeispiel: Keywords in Context (KWIC)

Zielgruppe:

etwa 12. Klasse.

Lernziele:

- Kennenlernen des Phasenmodells der Software-Entwicklung
- Entwicklung eines Programms in Gruppenarbeit
- Textverarbeitung in der Programmiersprache Z

Aufgabe:
In einer Kleingruppe von ca. 3-4 Personen ist ein Programm zu entwickeln, das folgenden Sollkonzeptrahmen erfüllt. Das Problem ist zu analysieren, das Programm zu entwerfen, zu implementieren und zu testen und alle Phasen zu dokumentieren. Nach Fertigstellung wird das Programm vom Lehrer abgenommen.
Von den angegebenen Ausbaumöglichkeiten ist mindestens eine zu bearbeiten.

Abgabetermine:
Anforderungsdefinition: etwa eine Woche nach Aufgabenstellung.
Vollständiges Produkt: etwa 3-4 Wochen nach Aufgabenstellung.

Sollkonzeptrahmen:
1. Systemziele
   Das System erleichtert das Aufsuchen von Textstellen (z.B. Buchtitel, Überschriften). Als Suchbegriff (sog. Keyword) treten alle vorkommenden Wörter auf. Das Sytem liest die Texte zeilenweise ein und gibt die Keywords in lexikographischer Reihenfolge zusammen mit den Textzeilen aus, in denen die Keywords jeweils vorkommen. Zusätzlich wird zu jeder Zeile angegeben, an welcher Stelle sie in der Ausgangsfolge vorkommt.
2. Benutzermodell
   Der Benutzer ist in der Lage, korrekte Eingaben gemäß der unten beschriebenen Spezifikation der Benutzerschnittstelle vorzunehmen. Ferner kann er Textdateien anlegen. Sonstige Kenntnisse über den benutzten Rechner bzw. die benutzte Programmiersprache sind nicht vorhanden.
3. Basismaschine
   Das Programm arbeitet auf dem Rechnermodell X unter Betriebssystem Y und ist in Programmiersprache Z erstellt. Die Minimalkonfiguration umfaßt Tastatur, Festplatte und Drucker.
4. Benutzerschnittstelle
   Jede Eingabe besteht aus einer Folge von Textzeilen beliebiger Länge. Hierfür ist ein geeignetes Eingabeformat festgelegt. Die Eingabedaten können über Tastatur eingegeben oder aus einer Datei gelesen werden. Die eingegebenen Daten werden auf Korrektheit geprüft, abgespeichert und zur Kontrolle über den Drucker ausgegeben.
   Die Textzeilen werden anschließend gemäß der lexikographischen Reihenfolge der Keywords angeordnet. Bei mehrfachem Auftreten eines Keywords wird der auf dieses Wort folgende Text zur Ermittlung der Reihenfolge herangezogen. Zu jeder Textzeile wird seine Position in der Eingabefolge angegeben. Die so bestimmte Abfolge der Textzeilen mit Positionsangabe wird gemäß einer ausgewählten Ausgabedarstellung (s.u.) ausgedruckt, wobei die Keywords besonders hervorgehoben sind.
   Folgende Ausgabedarstellungen sind vorstellbar:

Eingabe:

Hund beißt Kind.
Hund flog durch die Luft.

1. Ausgabemöglichkeit:

beißt:	Hund beißt Kind.	1
die:	Hund flog durch die Luft.	2
durch:	Hund flog durch die Luft.	2
flog:	Hund flog durch die Luft.	2
Hund:	Hund beißt Kind.	1
Hund:	Hund flog durch die Luft.	2
Kind:	Hund beißt Kind.	1
Luft:	Hund flog durch die Luft.	2

2. Ausgabemöglichkeit:

Hund bei**ßt** Kind.	1
Hund flog durch **die** Luft.	2
Hund flog **durch** die Luft.	2
Hund **flog** durch die Luft.	2
Hund beißt Kind.	1
Hund flog durch die Luft.	2
Hund beißt **Kind**.	1
Hund flog durch die **Luft**.	2

3. Ausgabemöglichkeit:

Hund	beißt	Kind.	1
flog durch	die	Luft. Hund	2
Hund flog	durch	die Luft.	2
Luft. Hund	flog	durch die	2
	Hund	beißt Kind.	1
die Luft.	Hund	flog durch	2
Hund beißt	Kind	.	1
durch die	Luft	. Hund flog	2

5. Ausbaumöglichkeiten

Der Benutzer kann die Keywordmenge durch

a) Kennzeichnung der Keywords

b) Kennzeichnung der Nicht-Keywords.

auf interessante Keywords einschränken. Hierzu ist die Eingabe geeignet erweitert.

2.4 Projektbeispiel: Wahlhochrechnung

Dies ist ein fast schon klassisches Beispiel für ein schulisches Informatikgroßprojekt, an dem auch viele andere Schulfächer beteiligt werden können. Wir skizzieren hier den Erfahrungsbericht des Lehrers R. Buhse, der das Projekt 1983 durchgeführt hat.

Zur Landtagswahl in Schleswig-Holstein am 13.3.1983 wurde in der Theodor-Heuss-Schule in Pinneberg im Rahmen einer öffentlichen Wahlparty eine Wahlhochrechnung mit dem Computer durchgeführt. Diese Veranstaltung war der Abschluß eines Projekts, an dem über einen Zeitraum von sechs Monaten über 80 Schüler und ein Lehrer beteiligt waren.

Vorbereitung.

Durch Aushang am Schwarzen Brett und durch persönliche Ansprache des Lehrers wur-

den im ersten Anlauf etwa 25 Schüler angeworben, die sich in der Auftaktsitzung zu einer Projektstruktur gem. Abb. 6 formiert haben. Zugleich ist ein Zeitplan verabschiedet worden.

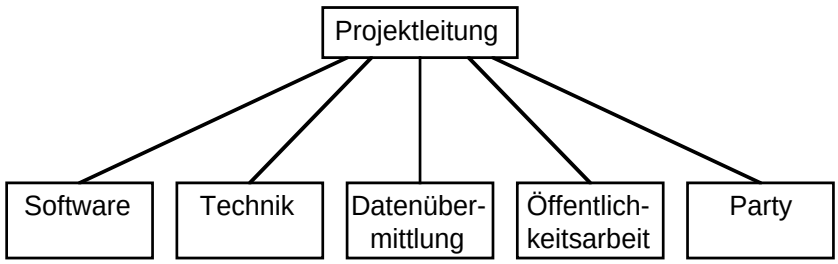


Abb. 6: Projektstruktur

Projektleitung.
An der Projektleitung (Abb. 7) waren auch Schüler beteiligt. Diese waren sehr engagiert und brachten im Laufe des Projekts viele Ideen ein. Daneben leisteten sie kleinere Hilfsarbeiten für andere Gruppen.

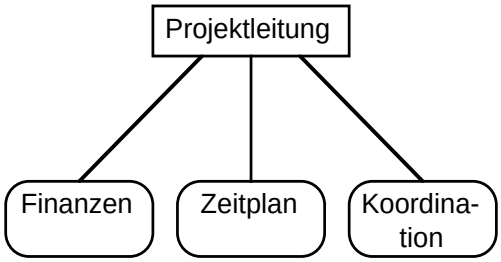


Abb. 7: Projektleitung

Software.
Diese Gruppe hat zwei Aufgaben: Bestimmung repräsentativer Stimmbezirke und Entwicklung der Hochrechnungssoftware (Abb. 8).

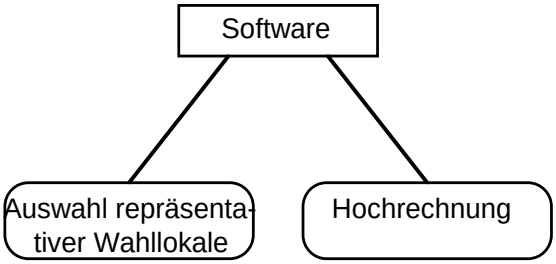


Abb. 8: Softwaregruppe

Aus den Daten der vier vergangenen Landtagswahlen 1967, 1971, 1975 und 1979 wurden insgesamt 80 Stimmbezirke ausgewählt, deren Trendgüte optimal war. Die Wähler innerhalb dieser Bezirke verhalten sich also in etwa so wie der landesweite Durchschnitt. Von den 80 Stimmbezirken befanden sich 40 im Kreis Pinneberg und 40 im Kreis Rendsburg-Eckernförde, dem Sitz einer befreundeten Schule, an der ebenfalls solch ein Projekt durchgeführt wurde.
Unabhängig davon entwickelten die Schüler das Hochrechnungsprogramm. Hier war besonders viel Wert auf benutzerfreundliches Programmverhalten zu legen, da einerseits für die Bedienung auch Schüler ohne Informatikkenntnisse eingesetzt werden mußten und es andererseits am Wahlabend vermutlich sehr hektisch zugehen würde. Die Eingabe sollte durch zwei Terminals in der Telefonzentrale (Schulsekretariat) erfolgen, wo die aktuellen Auszählungsergebnisse aus den Wahllokalen erwartet wurden. Die Ausgabe der Hochrechnungsergebnisse sollte über Drucker (zur Dokumentation) und über mehrere Fernsehgeräte in die Aula zur Wahlparty übertragen werden.
Probleme gab es mit der Rechnerkapazität, die nicht ausreichte, um Eingabe-, Verarbeitungs- und Ausgabekomponente gleichzeitig im Speicher zu halten.

Technik.
Diese Gruppe (Abb. 9) realisierte die Datenübertragungsverbindungen zwischen Computerraum und Eingabegeräten in der Telefonzentrale einerseits (ca. 50m) und der Videoausgabe der Hochrechnungen über 10 Fernsehgeräte in der Aula andererseits. Ferner mußten Mikrofone für die Party, eine Beleuchtungsanlage sowie eine Haustelefonanlage zur Verbindung aller Aktivitätszentren untereinander installiert werden.

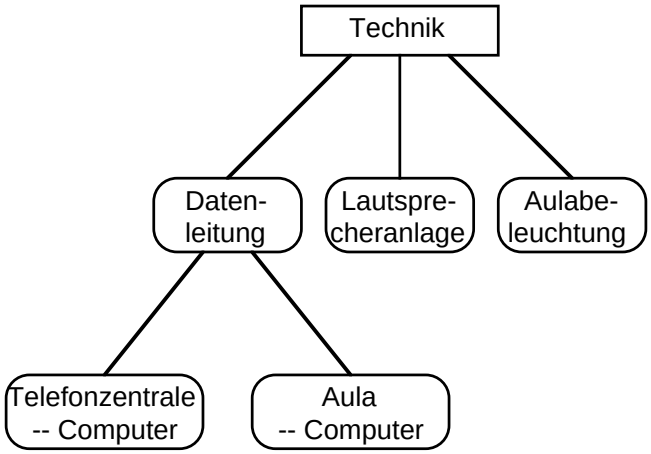


Abb. 9: Technikgruppe

Datenübermittlung.
Hauptaufgabe dieser Gruppe (Abb. 10), der anfangs zwei und später etwa 50 Schüler angehörten, war die Besetzung der Wahllokale und die Übermittlung der Ergebnisse an die Telefonzentrale. Die betreffenden Schüler mußten hierzu angelernt werden. Ferner gestaltete die Gruppe die Telefonzentrale und den organisatorischen Ablauf bei der Eingabe der Daten.

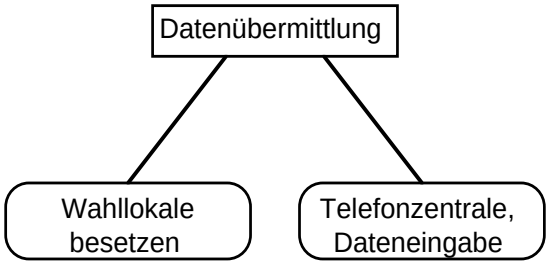


Abb. 10: Datenübermittlungsgruppe

Öffentlichkeitsarbeit.
Diese Gruppe suchte den Kontakt zur Presse und zur Lokalpolitik, um die Party in der Öffentlichkeit bekannt zu machen. Mit Glück konnte man schließlich die Direktkandidaten aus Pinneberg für eine Podiumsdiskussion gewinnen.
Ferner führte die Gruppe zwei Wahlumfragen mit jeweils ca. 1300 Pinneberger Bürgern durch; die erste wurde etwa sechs Wochen vor der Wahl öffentlichkeitswirksam in der Presse bekannt gemacht, die zweite blieb bis zum Wahlabend geheim.
Schließlich hatten die Mitglieder dieser Gruppe dann am Wahlabend die Party zu moderieren (Abb. 11).

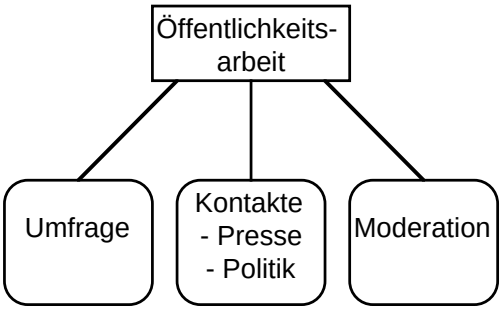


Abb. 11: Öffentlichkeitsarbeit

Party.
Aufgabe dieser Gruppe war der Entwurf und die Umsetzung eines Party-Programms (Abb. 12). Dazu gehörten die Gestaltung der Aula, die Positionierung der Fernsehgeräte, die

Auswahl und Aufstellung der Musikgruppe, die Beköstigung der Gäste usw.

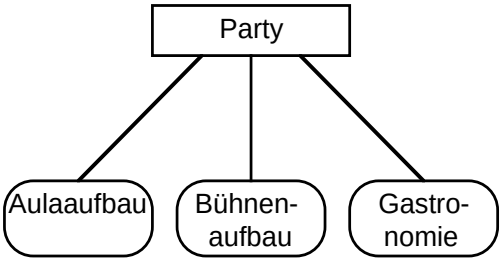


Abb. 12: Partygruppe

Verlauf und Fazit.
Bereits um 18.30 Uhr konnte die erste schon sehr gute Hochrechnung präsentiert werden. Um 20.00 Uhr begann die Podiumsdiskussion mit den Lokalpolitikern. Den Abschluß bildete eine Tanzparty.

Buhse wertet das Projekt als vollen Erfolg und stellt insbesondere folgende positve Effekte heraus:
- den Kontakt zwischen Informatikfachleuten und -laien
- das Wir-Gefühl unter den beteiligten Schülern
- das gegenseitige Aufeinanderangewiesensein
- die öffentlichkeitswirksame Präsentation einer sonst recht abgeschlossenen Behörde.