Pragmatischer Konstruktivismus und fundamentale Ideen als Leitlinien der Curriculumentwicklung

am Beispiel der theoretischen und technischen Informatik

Dissertation

zur Erlangung des akademischen Grades
doctor paedagogicae (Dr. paed.)

vorgelegt der

Mathematisch-Naturwissenschaftlich-Technischen Fakultät
(mathematisch-naturwissenschaftlicher Bereich)
der Martin-Luther-Universität Halle-Wittenberg

von Herrn Eckart Modrow
geb. am 18. Februar 1948 in Reinbek

Gutachter:

1. Prof. Dr. Wilfried Herget, Halle (Saale)
2. Prof. Dr. Andreas Schwill, Potsdam
3. Prof. Dr. Peter Hubwieser, München

Halle (Saale), 12. Februar 2003

urn:nbn:de:gbv:3-000004879
[http://nbn-resolving.de/urn/resolver.pl?urn=nbn%3Ade%3Agbv%3A3-000004879]

Ich erkläre hiermit, dass ich meine Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

27.10.2002, Scheden

Vorwort

Die vorliegende Arbeit geht von meinen 1991/92 erschienenen Büchern „Zur Didaktik des Informatikunterrichts, Bd.1 und 2“[1] sowie „Automaten–Schaltwerke–Sprachen“[2] aus. Sie erweitert und ergänzt diesen Ansatz in Hinsicht auf allgemeindidaktische Fragen und hat das Ziel, nach einer Pause von zehn Jahren einige der damals aufgestellten Behauptungen kritisch vor dem Hintergrund der fachlichen, didaktischen und bildungspolitischen Entwicklung in dieser Zeit zu überprüfen und fortzuschreiben, vor allem aber mit der vorangeschrittenen fachdidaktischen Diskussion zu verknüpfen. Sie versucht dieses exemplarisch anhand eines Themas, das ich für grundlegend für die Schulinformatik halte: der theoretischen Informatik.

Didaktische Arbeiten befinden sich immer in der Gefahr, die „Bodenhaftung“ zu verlieren. Im Bestreben, pädagogische Detailentscheidungen von allgemeinen pädagogischen und fachlichen Überlegungen her zu begründen, übersehen sie oft, dass die Adressaten des Unterrichts, die Schülerinnen und Schüler, diese Überlegungen nicht mit vollziehen. Denen erscheint der Unterricht allein als Folge von Resultaten dieser Überlegungen, den Unterrichtsinhalten und Unterrichtsformen der einzelnen Stunden. Didaktiker versuchen meist, von ihren Zielen über eine Folge von Einschränkungen und Anpassungen an die Schule zu konkreten, meist also stark reduzierten Unterrichtsbeispielen zu kommen, die in ihrer Gesamtheit die Ziele erreichen sollen. Von den Unterrichteten wird erwartet, dass sich bei ihnen aus den Beispielen die angestrebten Einsichten und Fähigkeiten entwickeln, sie also z. B. die überordnete Sicht des Faches auf die Wirklichkeit rekonstruieren. Dass dies nicht immer funktionieren muss, erlebt man als Praktiker leider täglich. Hier hat in den letzten Jahren mit dem Konstruktivismus eine veränderte Position zum Lernprozess gerade in der mathematisch-naturwissenschaftlichen Didaktik an Boden gewonnen, die diesen Aspekt m. E. angemessen berücksichtigt. Wenn ich auch nicht alle Konsequenzen des Konstruktivismus für pädagogisch sinnvoll halte, so scheint doch eine pragmatische Variante[3] auch für die Informatikdidaktik fruchtbar zu sein.

Didaktische Überlegungen sind m. E. daran zu messen, ob die Abfolge der daraus resultierenden Unterrichtsstunden in ihrer Wirkung den angestrebten Zielen gerecht wird. Entscheidend ist also nicht, was gewollt, sondern was erreicht wird – und in welchem Umfang es erreicht wird. Als Praktiker ist für mich die Frage entscheidend, ob die Folge der Unterrichtsstunden trotz der meist extremen Reduktion die ursprünglichen Intentionen noch widerspiegelt, ob also – hier nur als Beispiel – eine Unterrichtseinheit zum „*Verstehen natürlicher Sprache*“[4] dem selbst gesetzten Anspruch gerecht wird, wenn darin allein eine ziemlich primitive endliche, also auch regulär formulierbare Sprache definiert und „geparst“ wird. Natürlich kann man sich anhand eines solchen Beispiels über natürliche Sprache Gedanken machen. Zu fragen ist aber, ob ein regulärer Parser in diesem Fall überhaupt notwendig oder hilfreich ist und ob eine Unterrichtseinheit, in der die entscheidenden Aspekte des Themas nicht wirklich thematisiert werden, nicht am Thema vorbei geht. Verliert sich bei der notwendigen Reduktion der Bezug zu den Zielen, dann ist es für die Schülerinnen und Schüler unmöglich, aus den übrig

[1] [Mod91], [Mod92a]
[2] [Mod92b]
[3] z. B. in [Mül98], [Ple98]
[4] [Bau96a] S. 361

bleibenden Beispielen die angestrebten Einsichten zu gewinnen – der Unterricht wird (unter diesem Aspekt) sinnlos, und das spüren alle Beteiligten sehr genau.

Übertriebene (und dann nicht eingelöste) Ansprüche führen zu falschen Erwartungen und damit zu Frustrationen. Damit wird die Chance vergeben, die erreichbaren Ziele des Informatikunterrichts als positiv zu erfahren. Versucht ein Fach, seine Existenz überwiegend über sehr anspruchsvolle Ziele zu rechtfertigen, dann gefährdet insbesondere ein „neues" Fach beim Nichterreichen dieser Ziele seine Akzeptanz, im Extremfall seine Existenz. Berücksichtigt man weiter, dass Unterrichtsziele immer nur teilweise erreicht werden, dann muss zur Rechtfertigung eines Faches eine Hierarchie von Zielen heran gezogen werden, die ausgehend von „einfachen", sicher erreichbaren, bei anspruchsvollen endet, und die sicher erreichbaren Ziele allein müssen das Fach ausreichend begründen. Ausgehend von solidem fachlichen Arbeiten sollte der Unterricht dann auch zu Fragen führen, die *innerhalb* der Schule weitgehend offen bleiben müssen, die aber einzelne Schülerinnen und Schüler motivieren können, sich *nach* der Schule auf diesen Gebieten zu betätigen. Themen wie das oben angesprochene „Verstehen natürlicher Sprache" oder die aktuellen „Agenten" des Internets, die auch innerhalb der Universität noch nicht annähernd abgeschlossen behandelt sind, können also als „Ausblicke", „Erweiterungen" u. Ä. zu sehr interessanten Stunden führen. Sie aber heute entscheidend zur Rechtfertigung des Schulfaches heran zu ziehen, halte ich für aberwitzig: Bestehenden Unterricht an solchen, m. E. derzeit kaum erreichbaren Zielen zu messen und damit zu entwerten, entzieht der real existierenden Version des Fachs seine Existenzberechtigung, ohne sie durch etwas Besseres zu ersetzen. Das hat Wolfgang Ambros schon 1991 treffend formuliert:

> *„So wird man mit dem Schuldgefühl entlassen, dass die augenblickliche Praxis nicht mehr zu verantworten ist. Es wird einem aber nicht gesagt, wie man es besser machen kann".*[5]

Die Schulinformatik spielt innerhalb der Schulfächer eine Sonderrolle, die im Weiteren zu begründen ist. Hier schon soll erwähnt werden, dass die Möglichkeiten des Fachs nicht nur im kognitiven Bereich liegen. Vielmehr spielen Aspekte der Persönlichkeitsentwicklung, des selbstständigen Arbeitens und sozialen Lernens eine wesentliche Rolle, die für die Begründung des Schulfaches entscheidend ist. Diese Aspekte müssen bei der Auswahl von Unterrichtsinhalten und -formen ebenso entscheidend sein wie fachliche Fragen. Schulinformatik bildet keine Informatiker aus, sondern leistet einen wichtigen Beitrag zur Allgemein- und Persönlichkeitsbildung der Schülerinnen und Schüler, dient also ganz altmodisch auch zur „Erziehung". Die alte Forderung der Reformpädagogik des „*Lernens mit Kopf, Herz und Hand*" kann nicht ungestraft auf den Kopf reduziert werden. Zur „Herzensbildung" dient u. a. die Freude am Lernen und am eigenen Schaffen, die sich über erfolgreiches selbstständiges Arbeiten mit Hilfe des Werkzeugs Computer entwickeln kann. Ebenso wie die „*Erfahrung Mathematik*"[6] nicht nur für erfolgreiches mathematisches Arbeiten und eine mögliche Entscheidung „zur Mathematik" (z. B. als späterem Studienfach) wichtig ist, sondern auch zur Herausbildung einer positiven Einstellung zum Fach der künftigen „mündigen Bürgerinnen und Bürger" dient, die meist auf ganz anderen Gebieten tätig sind, sollte der Informatikunterricht eine „*Erfahrung Informatik*" liefern, die es den Schülerinnen und Schülern

[5] in [Amb91] zu [Bau90]
[6] [Dav94]

gestattet, zu entscheiden, ob Arbeiten in diesem Bereich für sie eine persönliche Perspektive bilden – oder nicht. Schule, und besonders die Sekundarstufe II, hat neben der Vermittlung fachlicher Inhalte die Aufgabe, den Unterrichteten alternative Lebenswege aufzuzeigen. Informatik ist in diesem Umfeld das einzige Fach mit einem starken technisch-ingenieurwissenschaftlichen Bezug[7]. Nur in diesem Fach können die Schülerinnen und Schüler erproben, ob die konstruktive Arbeit mit technischen Werkzeugen (im Gegensatz zur weitgehend deskriptiven und interpretierenden Arbeit in vielen anderen Fächern) für sie möglich und attraktiv, eben eine Lebensperspektive ist.

Die vorgelegte Arbeit bezieht sich fast ausschließlich auf das Schulfach Informatik in der Sekundarstufe II, geht also von Schülerinnen und Schülern aus, die das Fach freiwillig und durchgehend gewählt haben. (Dies umfasst auch die Wahl „in einem Bündel von Fächern“, wie sie etwa in der geplanten Profiloberstufe vorgesehen ist.) Sie versucht nicht, das Fach als Teil einer „informationstechnischen Bildung für alle“[8] darzustellen. Sie soll neben den fachlichen Aspekten auch die angesprochenen allgemein- und persönlichkeitsbildenden berücksichtigen und am gewählten Thema zeigen, wie diese im Unterricht zu erreichen sind. Ausgehend von allgemeinen Zielen soll der Weg zur einzelnen Unterrichtsstunde konsequent und ohne Bruch gegangen werden. Ich hoffe, dass dies gelungen ist.

Ich danke Prof. Wilfried Herget, Universität Halle, und Prof. Andreas Schwill, Universität Potsdam, für ihre Ermutigungen, Hilfen und ihre Betreuung in den letzten zwei Jahren. Ohne ihre freundliche Unterstützung hätte ich diese Arbeit nicht realisiert.

Göttingen, im Oktober 2002

Eckart Modrow

[7] siehe auch [Mag00] S. 4

[8] die nach Ansicht nicht nur der GI in der „integrierten“ Form weitgehend gescheitert ist. [GI01]

1. Aufgaben und Ziele der Informatikdidaktik

Das Schulfach Informatik ist ein Baustein im Unterricht allgemein bildender Schulen, der derzeit meist nicht zum verpflichtenden Fächerkanon gehört. Insbesondere in der Sekundarstufe II werden Informatikkurse von den Teilnehmerinnen und Teilnehmern ausschließlich freiwillig belegt. In diesem Abschnitt soll zuerst der Rahmen abgesteckt werden, innerhalb dessen sich ein technisch orientiertes Schulfach bewegt. Dazu wird der Begriff der *Allgemeinbildung* mit Hilfe einiger für das Thema dieser Arbeit wichtiger Aspekte der Allgemeinen Didaktik präzisiert[9] und in Hinsicht auf die Schulinformatik bewertet[10]. Als Folgerung ergeben sich Anforderungen an die Schulfächer[11], die sich besonders auf den Stellenwert der fachlichen Inhalte beziehen. Nach einem Ausflug in die Lerntheorie wird der spezielle Beitrag des Informatikunterrichts zur Allgemeinbildung untersucht, durch den dann die Anforderungen an die das Thema betreffende Informatikdidaktik definiert sind.

1.1 Einige Vorgaben der Allgemeinen Didaktik

1.1.1 Zum Praxisbezug

Didaktik ist die Wissenschaft vom Unterricht. Sie wird verstanden als Theorie der Bildungsinhalte, ihrer Struktur und Auswahl[12]. Verglichen mit dem rasanten Tempo, in dem sich Themen und „Paradigmen“ in der Informatikdidaktik abwechseln – wesentlich dokumentiert in den Beiträgen der Zeitschrift LOG IN –, ist die Entwicklung in der Allgemeinen Didaktik geradezu gemächlich: Immerhin spielen Ergebnisse z. B. von Jerome S. Bruner[13] aus den 60er Jahren in der aktuellen Diskussion eine wesentliche Rolle, etwa bei der Diskussion der „fundamentalen Ideen der Informatik“ durch Andreas Schwill[14]. Im Gegensatz zur Allgemeinen Didaktik ist es in der Informatikdidaktik schon aus Zeitgründen kaum möglich, aktuell diskutierte Themen in breit angelegten Erprobungen zu untersuchen[15], weil die Zeiträume, in dem die Themen jeweils eine Rolle in der didaktischen Diskussion spielen, viel zu kurz sind, um sorgfältiger Evaluation Raum zu geben[16]. Damit hat die Informatikdidaktik aber ein Problem, denn die Didaktik wird nach Wolfgang Klafki „*nicht als reine theoretische Disziplin verstanden, sondern als Wissenschaft von der Praxis für die Praxis*“[17]. Er beschränkt diesen Anspruch nicht auf die *geisteswissenschaftliche* Didaktik, sondern betont durch den Begriff „*konstruktiv*“ in seiner *kritisch-konstruktiven* Version der Didaktik ausdrücklich den Praxisbezug seiner eigenen Theorie[18]. Auch Paul Heimann sieht Didaktik als Theorie des Unterrichts und den Unterricht „*als Ort, wo die ungelösten Fragen der didaktischen Gesamtsituation als konkret zu lösende Lehr- und Lernprobleme auftre-*

[9] Dabei wird auch aus Platzgründen keine systematische Darstellung versucht.
[10] Es wird also von Anfang an auf Neutralität zugunsten einer eigenen Position verzichtet. Entsprechend ist die Auswahl der Zitate sehr selektiv.
[11] also auch an die Informatik
[12] W. Klafki, zitiert nach [Kop65] S. 8
[13] [Bru70]
[14] [Schw93a]
[15] Entsprechend fundierter scheinen aber auch die von der allgemeinen Didaktik vertretenen Thesen zu sein, insbesondere die Überprüfung in Schulversuchen und normalem Unterricht spielt eine andere Rolle.
[16] Als Beispiel kann die Diskussion der didaktischen Möglichkeiten von PROLOG dienen, die nach (gut gerechnet) zehn Jahren weitgehend durch eine Diskussion über JAVA abgelöst wurde, ohne dass PROLOG im Unterricht schon wirklich flächendeckend verankert gewesen wäre.
[17] [Kla85] S. 37, [Kla98a]
[18] [Kla85] S. 38

ten“[19]. Da Klafkis Didaktik als Beitrag zum allgemeinen Rahmen einer Informatikdidaktik in der Literatur weitgehend akzeptiert zu sein scheint[20], muss der geforderte Praxisbezug sehr ernst genommen werden. Theoretische didaktische Überlegungen finden ihre Berechtigung also nicht in sich selbst, sondern im Bezug zum und in der Umsetzung in Unterricht:

> *Wissenschaftliche Pädagogik ist nach dieser Auffassung keine Instanz, die in rein theoretischem Erkenntnisinteresse – „um der bloßen Erkenntnis willen“ – beobachtend und analysierend an die pädagogische Praxis herantritt.*[21]

Will die Informatikdidaktik diesem Anspruch genügen, dann kann sie sich nicht darauf beschränken, mehr oder weniger unverbindlich jeweils neue Vorschläge zu machen, also Erwägungen abzuliefern, „was man denn tun könnte“, sondern sie muss der Praxis Zeit geben, ihre Vorschläge zu realisieren, sie kritisch zu erproben und ggf. zu verändern. Fasst man den Begriff „Praxis“ nicht nur als „einzelne Unterrichtseinheit eines einzelnen Lehrers“ auf, sondern etwas breiter wenigstens als „normaler Unterricht einer größeren Gruppe von Unterrichtenden in einem größeren geografischen Gebiet“, dann sind für die Ausbreitung neuer Ideen in dieser Gruppe, ihre Umsetzung in Unterrichtseinheiten, die Entwicklung von Unterrichtsmaterialien usw. jeweils viele Jahre erforderlich. Eine Fachdidaktik, die diese einfache Tatsache nicht berücksichtigt, gibt ihren Anspruch auf Relevanz auf, weil sie nicht mehr erwarten kann, dass ihre Ergebnisse auch umgesetzt werden[22].

Praxisbezug bedeutet auch, dass didaktische Überlegungen die Realisierbarkeit der Vorschläge, also die aktuelle Situation der Praxis, wenigstens mit bedenken müssen, z. B. die vorhandenen Kenntnisse der Unterrichtenden, den Zeitaufwand, den die Einführung neuer Themen erfordert, und den Anteil der Arbeitszeit, der den Lehrenden insgesamt dafür zur Verfügung steht. Insbesondere ist bei Brüchen der Entwicklung[23], z. B. der Einführung ganz neuer Inhalte und Werkzeuge, abzuwägen, ob der für diese Veränderungen erforderliche Aufwand wirklich den erhofften Vorteil bringt oder ob durch eine kontinuierliche Weiterentwicklung, die bestehende Ressourcen weiternutzt, nicht mehr zu erreichen wäre.[24]

---

[19] [Hei68] S. 9

[20] z. B. in [Fri95], [Tho00], [Hum00], [Hub00]

[21] [Kla76] S. 17

[22] In der informatikdidaktischen Diskussion wird fast immer darauf hingewiesen, dass sich die Inhalte des Informatikunterrichts nicht im Tempo der Bezugswissenschaft Informatik ändern dürfen. Fast immer folgt nach diesem Vorspann aber eine Erörterung *neuer* Themen, *neuer* Programmiersprachen, *neuer* Anwendungen (z. B. in [For97], [Bau96a], [Hub00]), statt Bestehendes zu festigen und zu verbessern. Als Folge hat es die Mehrzahl der unterrichtenden Kolleginnen und Kollegen m. E. aufgegeben, der aktuellen Diskussion zu folgen – wenn sie sich nicht schon ganz aus dem Informatikunterricht zurückgezogen hat. Folglich legt diese Arbeit auch weniger Wert auf „Neuigkeiten“ als auf „solide Weiterentwicklung“ des Bestehenden.

[23] Gemeint sind die in der Schulinformatik im Vergleich zu anderen Fächern merkwürdigerweise gehäuft auftretenden „Paradigmenwechsel“.

[24] Sigrid Schubert schreibt dazu, *„dass die Allgemeinbildung (...) nicht durch hektische Veränderungen vorangebracht wird, sondern durch eine Folge wohldurchdachter kleiner Schritte.“* [Schu99] S. 24

1.1.2 Zum „Allgemeinen“ der Allgemeinbildung

Informatikunterricht ist in allgemein bildenden Schulen kein Selbstzweck, sondern hat im Zusammenspiel der Fächer der Allgemeinbildung[25] zu dienen. In einer Antwort auf einen Artikel Peter Rechenbergs[26] schreibt Heide Schelhowe zum Thema Allgemeinbildung, *„dass es nicht ausreicht, die Begründung für eine Didaktik der Informatik in der Entwicklung der wissenschaftlichen Disziplin zu suchen. Vielmehr muss der Blick gerichtet sein auf die gesellschaftlichen, ökonomischen, kulturellen Entwicklungen, um von dort aus zu fragen, welche allgemein bedeutsamen Antworten die jeweilige wissenschaftliche Disziplin darauf geben kann, diese Entwicklungen zu begreifen und mit zu gestalten“*[27]. Diese Definition entspricht teilweise der Klafkis, der allerdings vorsichtiger bzgl. der Prognostizierbarkeit der Zukunft ist und soziale Kompetenzen stärker betont[28]. Insbesondere präzisiert er den Begriff „allgemein“ in der Allgemeinbildung: [29]

1. *„Allgemein“ besagt hier, dass Bildung eine Möglichkeit und ein Anspruch aller Menschen der betreffenden Gesellschaft bzw. des betreffenden Kulturkreises, ja letztlich der Menschheit im Ganzen ist.*
2. *„Allgemein“ zielt weiterhin auf das Insgesamt der menschlichen Möglichkeiten.*
3. *Die Bestimmung „allgemein“ im Begriff der Allgemeinbildung meint schließlich, dass Bildung sich zentral im Medium des Allgemeinen vollzieht oder vollziehen sollte, d. h. in der Aneignung von und der Auseinandersetzung mit dem die Menschen gemeinsam Angehenden, mit ihren gemeinsamen Aufgaben und Problemen, den in der Geschichte bereits entwickelten Denkergebnissen und Lösungsversuchen.*

Nun können die Schulfächer natürlich nicht in gleichen Maßen zu allen drei Aspekten so verstandener Allgemeinbildung beitragen. Trotzdem kann man erwarten, dass sie den Rahmen ihrer Möglichkeiten ausschöpfen. In der informatikdidaktischen Diskussion entspricht Horst Hischer dem ersten Aspekt der Allgemeinbildung, wenn er im Vorwort zum Tagungsband einer mathematisch-fachdidaktischen Arbeitstagung aus einer Rede des damaligen Bundespräsidenten Herzog zitiert:

> *Wie die Mathematik kennt die Informatik keine Klassenunterschiede, keine nationalen Grenzen und keine kulturellen Barrieren. Ihr Potenzial ist überall emanzipatorisch. Es gibt dem arbeitssuchenden Jugendlichen in Brasilien die gleiche Chance wie dem in Thüringen, Kalifornien oder Indonesien. (...) Sie kann deswegen zur Integration der Randgruppen der Gesellschaft in regionalen, nationalen oder kulturellen Gemeinschaften ebenso gut beitragen wie zum Abbau der wirtschaftlichen Ungleichgewichte zwischen Nord und Süd in der Weltwirtschaft.*[30]

Wenn ich diese Ansicht auch für reichlich optimistisch halte[31], so ist doch zu beachten, dass die Aussagen als Ausgangspunkt für die fachdidaktische Diskussion gewählt wurden. Auch die GI in ihren „Empfehlungen für ein Gesamtkonzept zur infor-

---

[25] B.F. Skinner: *„Bildung ist das was bleibt, wenn man alles vergisst, was man einmal gelernt hat.“*
[26] [Rec97]
[27] [Sche97]
[28] *„Allgemeinbildung bedeutet, ein geschichtlich vermitteltes Bewusstsein von zentralen Problemen der Gegenwart und – soweit voraussehbar – der Zukunft zu gewinnen, Einsicht in die Mitverantwortlichkeit aller angesichts solcher Probleme und Bereitschaft, an ihrer Bewältigung mitzuwirken. Abkürzend kann man von der Konzentration auf epochaltypische Schlüsselprobleme sprechen.“* Klafki, zitiert nach [Fri95]
[29] [Kla85] S. 17
[30] [His95] S. 5
[31] Es reicht schon, sich die Verteilung der Computer auf die Nationen anzusehen, um die Chancen der arbeitssuchenden brasilianischen Jugendlichen einzuschätzen.

matischen Bildung“[32] betont den Stellenwert des gleichberechtigten Zugangs zu IuK-Techniken. In jedem Fall entspricht die Unterstützung der Emanzipation Benachteiligter dem Bildungsziel der *Solidarität* (z. B. nach Klafki[33] und Rolff[34]).

Der zweite Aspekt kann so interpretiert werden, dass den Schülerinnen und Schülern die ganze Breite ihrer Möglichkeiten vor Augen zu führen ist, und dazu gehört in der Sekundarstufe II[35] sicherlich auch ein Überblick über die verschiedenen Wissenschaften. Der riesige Bereich der Ingenieurwissenschaften hat im Fächerkanon des Gymnasiums kein zugeordnetes Fach. Physik – das die Aufgabe eigentlich übernehmen könnte – wird an Universität und Schule als reines Grundlagenfach betrieben, fast ohne Bezug zu aktueller Technik. Wenn also technische Disziplinen mit ihrer anwendungsorientierten Art des Umgangs mit Wissen und ihrer teilweise heuristischen Arbeitsweise von den Heranwachsenden überhaupt als Berufsperspektiven wahrgenommen werden sollen, dann sollte ein Fach sich darum kümmern – und Informatik als einziges technikorientiertes Fach im Gymnasium wäre dafür hervorragend geeignet, weil in seinem Unterricht gerade diese Arbeitsweisen von den Schülerinnen und Schülern *erprobt*[36] werden können.

Zum „*Insgesamt der menschlichen Möglichkeiten*“[37] gehören auch Erfahrungen über die eigenen Möglichkeiten und Grenzen. Da ich diesen Punkt für entscheidend wichtig für das Schulfach Informatik halte, gehe ich darauf später ausführlicher ein.

Der dritte Allgemeinbildungsaspekt Klafkis relativiert die Bedeutung fachspezifischer Inhalte. Als allgemein bildend fasst Klafki vor allem die Beschäftigung mit *Schlüsselproblemen* der Menschheit wie Friedensfragen, Umweltprobleme usw. auf:

> *Allgemeinbildung heißt im Blick auf solche Schlüsselprobleme also: Auf den verschiedenen Stufen des Bildungsganges bzw. des Bildungswesens sollte jeder junge Mensch und jeder Erwachsene mindestens in einige solcher Zentralprobleme – im Sinne exemplarischen, gründlichen, verstehenden bzw. entdeckenden Lernens – eingedrungen sein.*[38]

Teilt man diese Auffassung, dann muss versucht werden, entsprechende Schlüsselprobleme, zu denen das betrachtete Fach einen Beitrag leistet, zu benennen, und dann fachliche Aspekte auch daraufhin zu untersuchen, ob sie *auf schulischem Niveau* einen Beitrag zur Beurteilung der Probleme liefern können, ob sie also den zukünftigen mündigen Bürger dazu befähigen, an der politischen Diskussion dieser Probleme teilzunehmen[39]. Hans-Werner Heymann formuliert das so:

> *Die Verwirklichung von Demokratie und Menschenrechten setzt Allgemeinbildung als Bildung für alle (und nicht nur für eine Elite) voraus.*[40]

In diesem Sinne leisten die naturwissenschaftlich-technischen Fächer einen wesentlichen Beitrag zur Demokratie, indem sie zur Diskursfähigkeit unterschiedlicher ge-

---

[32] [GI01]
[33] [Kla85] S. 17
[34] zitiert nach [Hop96]
[35] In den Klassenstufen 10 und 11 ist das vermutlich noch wichtiger, weil dort die Entscheidungen für die Leistungsfächer und damit oft eine grobe berufliche Vororientierung fällt – weniger als Positiv- denn als Negativentscheidung: Fächer werden „abgewählt“. Eine „Nachwuchswerbung“ z. B. für den naturwissenschaftlichen Bereich in der Sek. II erreicht damit die Mehrzahl der Schülerinnen und Schüler gar nicht mehr.
[36] im Sinne der „Erfahrung Informatik“ (s. o.)
[37] bei Heymann zu finden unter „Stärkung des Schüler-Ichs“, z. B. in [Hey95]
[38] [Kla85] S. 21
[39] Eine ausführliche Erörterung von mir zu diesem Thema findet man in [Mod92a] S. 189
[40] [Hey95] S. 47

sellschaftlicher (Interessen-)Gruppen beitragen.[41] Im Bereich der Informationstechniken ist der Bedarf im diesem Bereich gerade jetzt offensichtlich und wird vermutlich eher steigen als sinken.[42]

1.1.3 Zur Stellung der „Bildung“ in der Allgemeinbildung

Bildung dient der *„wesensmäßigen Selbstverwirklichung der Person“*[43]. Sie muss *„als Selbstbestimmungs- und Mitbestimmungsfähigkeit des Einzelnen und als Solidaritätsfähigkeit verstanden werden“*[44].

> *Unterricht schafft in diesem Sinne eine anregende Umgebung, in der Menschen auf der Grundlage bedeutsamer Kulturgüter in einen kulturellen Vermittlungsprozess eintreten können, in welchem sie ihre eigene wertvolle Persönlichkeit herausbilden bzw. sich zu dieser bilden können.*[45]

Weist der Begriff der „***bedeutsamen Kulturgüter***“ hier auf eine mehr materiale Auffassung von Bildung hin, also *„einen Kanon in sich wertvoller Lerngegenstände, die der gebildete Mensch kennen sollte“*[46], so lässt die „***Herausbildung der eigenen wertvollen Persönlichkeit***“ immer noch die Frage offen, wie denn dieser Prozess eingeleitet werden kann, bietet also auch einer formalen Bildungstheorie Raum, die Methoden[47] vermitteln möchte, mit deren Hilfe dann Inhalte, also auch die „***wertvollen Lerngegenstände***“ individuell erschlossen werden können.

Die Inhalte eines neuen Fachs werden sich wohl kaum in einem etablierten *festen* Kanon von Lerngegenständen finden. Es ist deshalb für die Schulinformatik entscheidend, dass entweder im Sinne der materialen Bildung innerhalb eines *offenen* Kanons Kriterien zur Auswahl von Lerngegenständen angegeben werden, anhand dessen sich „wertvolle“ identifizieren lassen, oder dass im Sinne einer formalen Bildung zu erwerbende Methoden genannt werden, die sich nur im Informatikunterricht, oder wenigstens dort besser als in anderen Fächern, erwerben lassen. Für die Praxis ist diese Trennung künstlich, es ist, wie so oft, ein „Sowohl-als-auch“ wünschenswert, das der *kategorialen* Bildung nach Klafki entspricht: Im Sinne des exemplarischen Lehrens und Lehrens werden anhand ausgewählter „wertvoller“ Inhalte sowohl Methoden erlernt wie auch Einsichten gewonnen[48]. Marco Thomas beschreibt das so:

> *Kategoriale Bildung meint, dass Menschen in der Lage sind, von der Welt begründete, d. h. durch Erkenntnis gewonnene, geprüfte Aussagen zu machen. Diese Fähigkeit ist stets an die Inhalte gebunden, die zur Aussage stehen. Formales und materiales Moment bilden damit eine Einheit, die auch den Bildungsprozess ausmacht, in dem die Fähigkeit zur Aussage und die Aussage selbst gewonnen werden.*[49]

---

[41] Im Bezug zur Kernenergiediskussion: siehe H. Körner in [His95] S. 72
[42] Die fehlenden Investitionen der Vergangenheit in den Bildungsbereich beginnen sich ja gerade z. B. bei der „Greencard“-Diskussion zu zeigen.
[43] Stippel, zitiert nach [Kop65] S. 11
[44] [Kla85] S. 17, [Kla98a] S. 4
[45] [Tho00] S. 1
[46] [Bau96a] S. 24
[47] Dieser Begriff wird ziemlich weit aufgefasst.
[48] [Kla85] S. 90
[49] [Tho00] S. 2

Der Konsens über einen allgemeinen, weit gefassten Bildungsbegriff in einer Gesellschaft, der u. a. Entscheidungen darüber zulässt, bei welchen Kulturgütern es sich um „*bedeutsame*“ handelt, ermöglicht es, die Elemente des Unterrichts und seiner Planung einzuordnen, zu bewerten, begründet zu akzeptieren oder zu verwerfen:

> *Eine zentrale Kategorie wie der Bildungsbegriff (...) ist unbedingt notwendig, wenn die pädagogischen Bemühungen (...) nicht in ein unverbundenes Nebeneinander oder gar Gegeneinander von zahllosen Einzelaktivitäten auseinander fallen sollen, wenn vielmehr pädagogisch gemeinte Hilfen, Maßnahmen, Handlungen und individuelle Lernbemühungen begründbar und verantwortbar bleiben oder werden sollen.*[50]

Für die praktische Arbeit ist es deshalb entscheidend, über einen hantierbaren Bildungsbegriff zu verfügen, also Kriterien zu besitzen, nach denen sich z. B. die „*bedeutsamen*“ Kulturgüter eines Faches identifizieren lassen. Ein Beispiel dafür finden wir in den *Kriterien für Bildungswerte* nach Rolff[51]:

- ***Gestaltbarkeit*** *– historische und politische Zusammenhänge aufzeigen.*
- ***Durchschaubarkeit*** *– Wissenschaftsorientierung und Erkenntniskritik.*
- ***Sinnlichkeit*** *– zu Eigentätigkeit anregen und Erfahrungen mit dem Erleben verbinden.*
- ***Ganzheitlichkeit*** *– den Zusammenhang mit der Lebenspraxis verständlich machen.*
- ***Solidarität*** *– Beschränkungen abbauen und Lernende stärken.*

In der Informatikdidaktik hat sich der Katalog von Bussmann und Heymann[52] weitgehend durchgesetzt, der sehr viel konkretere und damit direkt anwendbare Kriterien enthält. Danach dient die Bildung

- *der Vorbereitung auf zukünftige Lebenssituationen,*
- *der Stiftung kultureller Kohärenz,*
- *dem Aufbau eines Weltbildes,*
- *der Anleitung zum kritischen Vernunftgebrauch,*
- *der Entfaltung eines verantwortlichen Umgangs mit erworbenen Kompetenzen*
- *und der Stärkung des Schüler-Ichs.*

Von diesem Katalog wird in den folgenden fachdidaktischen Kapiteln ausgegangen, und dabei wird er noch genauer diskutiert.

Zu beachten ist, dass die in der öffentlichen Diskussion so stark herausgestellte „*Brauchbarkeit*“ von Bildung[53], individuell gesehen z. B. zur „*Berufsvorbereitung*“ oder Herstellung einer „*Studierfähigkeit*“, gesamtgesellschaftlich gesehen als „*Konkurrenzfähigkeit*“ der Bildungssysteme im globalen Wettbewerb, in diesen Katalogen nur sehr versteckt zu finden ist. Solche Gesichtspunkte *können* natürlich unter Begriffen wie „*Vorbereitung auf zukünftige Lebenssituationen*“ und „*Stärkung des Schüler-Ichs*“ subsumiert werden, sie *müssen* es aber nicht. Es bleibt der aktuellen bildungspolitischen und didaktischen Diskussion überlassen, wie sie die Kriterien der Allgemeinen Didaktik im aktuellen gesellschaftlichen Kontext interpretieren will.

---

[50] [Kla85] S. 13
[51] zitiert nach [Hop96]
[52] [Bus87]
[53] Bildung gesehen als *Aus*bildung

Der Wert des Rahmens, den ein allgemein gefasster, im Vergleich zu aktuellen Änderungen zeitlich relativ konstanter Bildungsbegriff für die fachdidaktische Debatte liefert, liegt gerade darin, dass mit seiner Hilfe unterschiedliche aktuelle Entwicklungen *begründet* bewertet werden können[54]. Er ermöglicht zwar keine Entscheidung darüber, welche fachlichen Themen im Einzelnen zu behandeln sind[55], er gestattet es aber sehr wohl, den *Stellenwert* von fachlichen Inhalten und die *Zielrichtung*, unter der sie unterrichtet werden sollten, zu bestimmen und damit z. B. äquivalente Themen zu ermitteln, die einander ablösen können. In diesem Sinne kann die allgemeine Didaktik der sich viel zu schnell ändernden Informatikdidaktik als dringend erforderliche zeitlich konstante Stütze dienen, die im Zusammenspiel von allgemeinen und fachlichen Kriterien einen relativ dauerhaften Themenkatalog ermittelt, in dem einzelne Bausteine durch gleichwertige aktuelle bei Bedarf ersetzt werden können, ohne gleich das ganze Ideengebäude zum Einsturz zu bringen.

Ein allgemeindidaktischer Rahmen kann die fachdidaktische Diskussion vor Entgleisungen wie z. B. dem unsäglichen „*Programmiersprachenstreit*“[56] und der Diskussion der *Brauchbarkeit* von Programmiersprachenkenntnissen schützen, deren Bewertung durch die Wirtschaft[57] ja ähnlich schnell wechselt wie die fachdidaktischen Themen. Er ermöglicht Evaluation des Bestehenden und evolutionäre Weiterentwicklung, und er sorgt für Werterhaltung bei den Kenntnissen der Unterrichtenden, ohne die ein Schulfach dauerhaft nicht existieren kann[58]. Die Vorgaben der Allgemeinen Didaktik relativieren die Bedeutung aktueller Fachthemen durch die Frage nach deren Bedeutsamkeit als Kulturgut und dem Bedarf, bei ihrer Behandlung allgemeine Methodenkenntnisse im oben angegebenen Sinne zu erwerben. Sie erzwingen so m. E. eine stärkere „Theoretisierung“ des Informatikunterrichts, da die Beiträge zur Kultur eher in den dauerhaften theoretischen Grundlagen als in technischen Details zu finden sind[59]. Sie ermöglichen durch ihre Weite aber auch das Eingehen auf aktuelle Trends, wenn diese nicht als Selbstzweck behandelt werden, sondern innerhalb des Zielkatalogs zu begründen sind[60]. Da der Reiz des Faches bei den Unterrichteten und wohl auch bei vielen Unterrichtenden gerade in der Aktualität liegt, bietet sich so die Chance, diesen speziellen Charme des Fachs zumindest teilweise zu erhalten.

---

[54] s. auch [Mod91] S. 36

[55] s. [Hey95] S. 48

[56] der m. E. nach wie vor andauert, denn in der Literatur nehmen Beiträge über die jeweils neuesten Programmiersprachen – unter wechselnden Überschriften, z. B. getarnt als „Paradigmendiskussion“ – immer noch einen unverhältnismäßigen Raum ein.

[57] Aktuelle Brauchbarkeit ist m. E. überhaupt kein Kriterium für allgemein bildende Schulen, weil die Vorlaufzeiten bis zum Berufseintritt für unsere Schülerinnen und Schüler viel zu hoch sind, als dass die aktuellen Stellungnahmen der Wirtschaft zu diesem späten Zeitpunkt noch eine Relevanz hätten.

[58] In der derzeitigen Situation meist ohne universitäre Fachausbildung kann den weitgehend autodidaktisch „ausgebildeten“ Lehrerinnen und Lehrern kaum zugemutet werden, diesen Lernaufwand dauerhaft durchzuhalten. Aber auch dann, wenn Informatik als Studienfach etabliert ist, müssen die in der Universität gewonnenen Fachkenntnisse eine längere „relevante“ Lebensdauer haben als heute.

[59] s. a [Schu99] S. 8

[60] z. B. durch persönlichkeitsbildende Ziele

1.2 Zum Lernen

1.2.1 Lernen in Strukturen[61]

Um die Wirkung von Unterricht zu beurteilen, sind die formalen Aspekte der Bildung schlechter geeignet als die materialen, und auch die müssen erst mal eine Auswirkung hinterlassen, also im Gedächtnis verankert werden. Zu fragen ist also nicht nur, *was* gelehrt werden soll, sondern ebenso, *wie* die Vermittlung zu erfolgen hat, um diese „Spuren" zu erzeugen. Jerome Bruner beschreibt das so:

> *Vielleicht das Grundlegendste, was man nach einem Jahrhundert intensiver Forschung über das menschliche Gedächtnis sagen kann, ist, dass Einzelheiten schnell wieder vergessen werden, wenn sie nicht in eine strukturierte Form gebracht worden sind. Detailliertes Material wird im Gedächtnis unter Anwendung vereinfachender Darstellungsweisen aufbewahrt. Diese vereinfachenden Darstellungen haben sozusagen eine „regenerative" Funktion.*[62]

Lernen erfordert also die Umwandlung des präsentierten „Stoffs" in mehreren weitgehend gleichzeitig ablaufenden Schritten[63] über die *Aneignung* zur *Transformation*[64] und *Wertung*. Diese sind *„nur über die aktive Beteiligung der Lernenden möglich"*[65]. *„Verstehen ist auf die geistige Aktivität der Lernenden angewiesen."*[66] Dafür müssen die Lernenden u. a. das Ziel und den Sinn des Unterrichts erkennen, sie müssen das zu Lernende mit möglichst viel vorhandenem Wissen in Beziehung setzen, es „vernetzen", und sie müssen vor allem in der Lage sein, dies zu tun. Da die vorliegende Arbeit sich nur mit Informatikkursen der Sek. II beschäftigt, sind die lernpsychologischen Voraussetzungen in der betrachteten Altersgruppe nach Piaget erfüllt, um neues Wissen konstruktiv in bestehendes einzuordnen. Zu fragen ist aber nach einem Kontext, in dem das effizient geschehen kann.

„Der Erwerb von Wissen setzt stets schon Vorwissen voraus"[67], das in sehr unterschiedlicher, nicht unbedingt bewusster Form vorliegen kann. Dieses vorhandene Weltbild prägt die Haltung der Lernenden zum Lerngegenstand, aber auch zum Lernen überhaupt, nicht nur in kognitiver, sondern auch in affektiver und operativer Hinsicht; es kann Lernen erleichtern, aber auch erschweren und sogar verhindern. Besonders die Transformation und die Wertung neuer Informationen erfordern, wenn sie fast gleichzeitig mit dem Wissenserwerb ablaufen, einen ordnenden Rahmen, also Vorwissen. Neues Wissen muss einerseits in bestehende Strukturen eingeordnet werden[68], andererseits Strukturen und Haltungen aufbauen, die weiteres Lernen ermöglichen[69]. *„Zur Beherrschung der grundlegenden Kategorien eines Lehrfachs gehört nicht nur das Begreifen allgemeiner Prinzipien, sondern auch das Herausbilden einer Einstellung gegenüber Lernen und Forschen, Vermutungen und Ahnungen, sowie der Möglichkeit, Probleme aus eigener Kraft zu lösen."*[70] Wenn Lernen also Haltungen verändert, die wiederum das

---

[61] weitgehend nach Bruner und Klafki
[62] [Bru70] S. 36
[63] [Bru70] S. 57
[64] um ihn für neue Anwendungen tauglich zu machen
[65] [Hub00] S. 10
[66] [Hey95] S. 54
[67] [Ber98] S. 41
[68] und diese dabei ggf. verändern.
[69] [Ben02] S. 4: "*Knowledge is acquired recursively: sensory data is combined with existing knowledge to create new cognitive structures, which are in turn the basis for further construction."*
[70] [Bru70] S. 33

Lernen beeinflussen, dann muss das Gelernte[71] nicht in einem Schritt erworben werden, sondern ist besser über eine „Spirale“ aus Vorbereitungs- und Präzisierungsschritten zunehmend zu erfassen. Die einzelnen Schritte müssen fortsetzbar sein, Begriffe präfigurieren[72] und können, wenn es erforderlich ist, Wissensgebiete schon in einer einfachen Sicht vorwegnehmen, bevor sie später abschließend behandelt werden. Sie müssen den Lernenden vor allem die Strukturen deutlich machen, also die Regeln, die auf dem gerade bearbeiteten Gebiet herrschen:

> *Wie kann man erreichen, dass seine Darstellung in ihrem Denken für den Rest ihres Lebens etwas bedeuten wird? Nach der vorherrschenden Ansicht (...) liegt die Antwort auf diese Frage darin, den Schülern ein Verständnis der Grundstruktur jeglichen Lehrgegenstandes zu vermitteln, den wir für den Unterricht auswählen.*[73]

Nach Bruner zeigt sich die Struktur eines Faches in seinen *fundamentalen Ideen*[74], weil erst in deren Kontext fachliche Inhalte, also spezifische Sachverhalte und Fertigkeiten, einzuordnen sind. Sie gestatten es, Gelerntes zu verallgemeinern, zu übertragen und so „*praktisch verwendbar*“ zu machen. Ähnlicher Meinung ist Klafki:

> *Lernen ... muss in seinem Kern entdeckendes bzw. nachentdeckendes und sinnhaft, verstehendes Lernen anhand exemplarischer Themen sein, ein Lernen, dem die reproduktive Übernahme von Kenntnissen und alles Trainieren, Üben, Wiederholen von Fertigkeiten eindeutig nachgeordnet oder besser: eingeordnet werden muss, also als notwendige, aber nur vom entdeckenden und/oder verstehenden Lernen her pädagogisch begründbare Momente.*[75]

1.2.2 Entdeckendes Lernen

Das nicht nur von Klafki geforderte *entdeckende Lernen*[76] setzt nun noch mehr als das Lernen überhaupt aktive Lernende voraus und damit eine Haltung, die Lernen fördert. *„Um solche Einstellungen durch Unterricht zu vermitteln, braucht man ein wenig mehr als die bloße Darbietung grundlegender Ideen. (...) aber es scheint, dass dabei ein für Entdeckungen zu begeisternder Sinn ein wichtiges Ingredienz ist für die Entdeckung von Regelhaftigkeiten bislang nicht erkannter Beziehungen und von Ähnlichkeiten zwischen Ideen mit einem sich daraus ergebenden Gefühl des Selbstvertrauens in die eigenen Fähigkeiten.“*[77] Entdeckendes Lernen fördert also genau die Haltungen, die für entdeckendes Lernen erforderlich sind. Ein solcher Unterricht, rechtzeitig begonnen, trägt, wenn er erfolgreich ist, sich selbst.

In diesem Zusammenhang ist Vorsicht bei der Unterrichtsplanung geboten. Wird der Unterricht stark lernzielorientiert ausgerichtet, dann kann entdeckendes Lernen erschwert werden:

> *Wer das Ziel verfolgt, dass Schüler (...) eigene Fragen und Vorschläge einbringen, dass sie möglichst selbstständige Wege zu einem von ihnen als sinnvoll erkannten Ziel erlernen, sei es auch über Irrtümer und Umwege, der kann das „zweckrationale Unterrichtskonzept“ allenfalls für begrenzte Teile des Unterrichts akzeptieren.*[78]

---

[71] zitiert nach [Tho00] S. 18
[72] Die Begriffe werden intuitiv benutzt, bevor sie vollständig analysiert werden.
[73] [Bru70] S. 25
[74] [Bru70] S. 42
[75] [Kla85] S. 85
[76] z. B. [Wag80]
[77] [Bru70] S. 33
[78] [Kla85] S. 85

Entdeckendes Lernen setzt aktive Beschäftigung mit dem Lerngegenstand voraus, und die kann – besonders anfangs – nicht nur im kognitiven Bereich liegen. Selbstständiges Arbeiten in einem Bereich, mit einem Gegenstand, für einen Zweck, hilft, Probleme erst einmal zu finden, einzuordnen und adäquate Fragen zu formulieren. *„So liegt ja auch den praktischen Übungen in den Schullaboratorien die Annahme zugrunde, dass etwas tun einem hilft, es zu verstehen."*[79] Wenn also in den Naturwissenschaften praktisches Experimentieren für unverzichtbar für einen erfolgreichen Unterricht gehalten wird[80]: Weshalb sollte im Informatikunterricht auf Phasen aktiver Schülerarbeit zugunsten von noch mehr deskriptivem und interpretierendem Handeln verzichtet werden? Die „Vergeisteswissenschaftlichung" der Informatik entspricht zwar weitgehend der „normalen" Arbeitsweise der Sek. II, nimmt dieser Schulform aber die Chance, auch eine andere Seite produktiven Tuns erlebbar zu machen. Die im Anhang dieser Arbeit vorgestellten Ergebnisse geben zumindest für die Schülerinnen und Schüler von Informatikkursen starke Hinweise, dass gerade diese ein entsprechendes Vorgehen auch wünschen.

> *Eine der am wenigsten erörterten Möglichkeiten, einen Schüler durch eine schwierige Unterrichtseinheit zu bringen, ist, ihn durch eine Gelegenheit, einmal zu zeigen, was in ihm steckt, anzuspornen, damit er die Freude entdeckt, erfolgreich und völlig aus sich heraus zu wirken.*[81]

Wenn die Experimentierphasen des entdeckenden Lernens teilweise dem analytischen Ordnen und Werten der Erfahrungen vorangehen, wenn also Vorerfahrungen gemacht werden, die erst später in neue oder vorhandene Strukturen einzuordnen sind, dann kann das für die Arbeit auf einem Gebiet eigentlich benötigte systematische Wissen nicht vorausgesetzt werden, denn das soll ja erst gewonnen werden. Wollen die Schülerinnen und Schüler trotzdem aktiv „forschen", dann müssen sie zumindest teilweise die nächsten Arbeitsschritte intuitiv ermitteln: sie müssen „raten".

> *„Was wir die Schüler zu erkennen lehren sollten ist wahrscheinlich sowohl, wann die Kosten des Nicht-Ratens zu hoch sind, als auch wann das Raten selbst zu teuer wird."*[82]

Wenn „*Raten*" bedeutet, mit der Arbeit zu beginnen, ohne die Lösung, das Ergebnis der Arbeit, zu kennen oder überhaupt zu wissen, ob es eine Lösung gibt, dann bildet im Bereich der Informatik der Prozess, zu einem Problem einen eigenen Algorithmus zu finden, wenigstens teilweise eine Form des Ratens. Natürlich gibt es systematische Verfahren, die bei der Lösungssuche helfen. Trotzdem bleibt ein wesentlicher Rest intuitiven Denkens erforderlich, da diese Verfahren das Ergebnis nicht determinieren. *„Solches Denken erfordert daher die Bereitwilligkeit, im Bemühen um Lösungen von Aufgaben, ehrliche Fehler zu machen."*[83] „Ehrliche" Fehler führen in Sackgassen, müssen gefunden, analysiert und korrigiert werden – und das erfordert Zeit. Dieser Zeitbedarf ist gerechtfertig, wenn der Lösungsprozess inklusive aller Fehler selbst Gegenstand des Unterrichts ist, wenn also die Inhalte, anhand derer Problemlösungsverfahren erprobt werden, den Methoden klar nachgeordnet sind.

> *Gerade die Möglichkeit, Fehler zu machen und daraus zu lernen, erfordert Zeit und Muße. Diese wird sich aber kaum einstellen, wenn Lehrer, Schülerinnen und Schüler in dem Bewusstsein leben, in dieser Zeit „nicht im Stoff voranzukommen".*[84]

---

[79] [Bru70] S. 41
[80] „Begreifen" kommt ja nicht ohne Grund „von den Händen".
[81] [Bru70] S. 59
[82] [Bru70] S. 72
[83] [Bru70] S. 73
[84] [Mod91] S. 22

Selbstständig arbeitende, intrinsisch motivierte Schülerinnen und Schüler lernen im Lernen das Lernen, und sie lernen das wesentlich besser, als wenn sie nur durch Zensuren motiviert werden. In diesem Fall besteht immer die Gefahr, *„dass das Lernen aufhört, sobald die Zensuren nicht mehr erteilt werden – nach der Abschlussprüfung."*[85] Aktive Auseinandersetzung mit dem Unterrichtsthema, möglichst nach eigenen Fragestellungen und mit selbst gewählten Methoden, mit der Möglichkeit Fehler zu machen, sie zu lokalisieren und zu korrigieren, fördert eine Haltung, die dem zu beobachtenden Trend des passiven „Unterhaltenwerdens" massiv entgegensteuert. Sie erzeugt „Frustrationstoleranz" und Durchhaltevermögen durch begründetes Selbstvertrauen, und sie verleiht der Bildung einen Eigenwert, der auch dann Bestand behält, wenn der Gebrauchswert des Gelernten nicht den Erwartungen entspricht.

1.2.3 Soziales Lernen

Lernen in der Schule findet immer im sozialen Verband statt. Man kann es unter dem Gesichtspunkt der beteiligten Akteure sehen. Darunter sind Lehrerinnen und Lehrer, Schülerinnen und Schüler und – im Bereich der Informatik – ggf. auch Computer und die daraus gebildeten Netze zu verstehen. Die letzteren, meist bezeichnet als *Informatiksysteme*, verschieben ein wenig den Gesichtspunkt auf eine eher technische Sicht der Kommunikation. Sigrid Schubert z. B. rechnet zum Bildungswert der Informatik u. a. Sozialkompetenz und versteht darunter *„Bewertungskriterien, Möglichkeiten und Grenzen, gesellschaftliche Auswirkungen, Verständnis für Sprachen, Mensch-Maschine-Kommunikation, Telekommunikation"*[86]. Das sind sicherlich alles Aspekte, die im Sozialen eine Rolle spielen; sicherlich stehen aber z. B. die hier gemeinten formalen Sprachen nicht im Zentrum der (menschlichen) Kommunikation in der Schule.

Man kann soziales Lernen auch auffassen als ein Lernen *miteinander*, *voneinander* und *füreinander*.

Das *Miteinander* ergibt sich zwangsläufig durch die schulische Organisation, wenn man sich auf die physische Anwesenheit beschränkt. Ob das Lernen aber wirklich gemeinsam erfolgt, also *im Miteinander*, hängt wesentlich vom Ablauf des Unterrichts ab und wird durch eine ausschließlich individuelle Leistungsbewertung in der Sek. II nicht gerade gefördert[87].

> *In jeder Kommunikation ist ein Inhalts- und ein Beziehungsaspekt zu erkennen. Die Inhaltsebene wird in der Schule weitgehend durch das Curriculum festgelegt. Auf der Beziehungsebene kommt die Art und Weise zum Tragen, wie die Information im Lehrer-Schüler-Verhältnis vermittelt wird.*[88]

Miteinander Lernen erfordert die Bereitschaft, aufeinander einzugehen, zuzuhören, zu argumentieren, eigene Ansichten darzulegen[89] und zu verteidigen. Es fördert die Einsicht in eigenes Können und eigene Grenzen, die entweder akzeptiert oder überwunden werden müssen. Es erfordert Rücksicht einerseits auf Langsamere, die ein Recht auf sinnvolle Teilhabe am Lernprozess haben, wie auf Schnellere, die ebenso

[85] [Bru70] S. 60
[86] [Schu99] S. 24
[87] Auch die unverhältnismäßige Bedeutung, die den Zensuren in der Vergangenheit z. B. bei der Studienplatzverteilung zugemessen wurde, verhinderte oft ein Klima der Zusammenarbeit.
[88] Watzlawick 1972, zitiert nach [Tho00] S. 4
[89] und natürlich erst einmal zu entwickeln

ein Recht auf die Erprobung und Erweiterung eigenen Könnens besitzen[90]. Das alles braucht Eigenschaften, die im Zeitalter der Ein-Kind-Familien weniger denn je als selbstverständlich vorausgesetzt werden können, die aber unter dem Stichwort „*Teamfähigkeit*" inzwischen oft höher als fachliche Kenntnisse eingeschätzt werden. Steffen Friedrich fasst das wie folgt zusammen:

> *Informatik hat in der Allgemeinbildung erst recht einen festen Platz, wenn man das Schulfach nicht nur als fachlichen Aspekt betrachtet. Im Rahmen des Unterrichts werden Arbeitsmethoden entwickelt und gefestigt, deren Bedeutung für den Schüler zu wenig beachtet wird. Das betrifft. z. B. handlungsorientiertes Lernen / experimentelles Arbeiten, Gruppenarbeit / Projekt als Arbeitsformen, Präzision / Genauigkeit im Arbeitsablauf, Arbeit mit computergestützten Lernumgebungen.*[91]

Beim Miteinander-Lernen muss es sich nicht unbedingt um eine sehr ernsthafte Veranstaltung handeln. Bei Jugendlichen hat das Miteinander immer auch einen spielerischen Zug, den Hischer betont: *„Die so genannte ‚Wissenschaftsorientierung' hat zu einer unerfreulichen kognitiven Überbetonung des Unterrichts insgesamt geführt."*[92] Er empfiehlt, Mathematik und Informatik teilweise als Spiel zu betreiben, denn *„Spielen öffnet Spiel-Räume"*[93], und Spielraum braucht der Mensch zur eigenen Entfaltung. In Spielen werden Fähigkeiten entwickelt und erprobt sowie Erfahrungen gewonnen, die auf ernsthafte Tätigkeiten vorbereiten. In diesem Sinne ist Schule ein Spiel, sie kann und sollte diesen Aspekt sehr viel stärker betonen als bisher, denn Spielen hat auch mit Spaß und Freude zu tun, die dem Lernen nur gut tun können.

Zum Miteinander kommt das *Voneinander*, wenn unterschiedliche Qualitäten der Beteiligten eingebracht werden[94]. Eine altbekannte Form davon ist das „*Einander-Helfen*", wobei der Helfende meist mehr lernt als der, dem geholfen wird. Bezieht man den Computer in die „sozialen" Beziehungen ein, dann kann dem Computer „*geholfen*" werden, indem helfende, weil mit Verständnis gesegnete Schülerinnen und Schüler ihm ihre Kenntnisse „*beibringen*", ihn also programmieren.[95] Die Rolle „*des Lernenden*" scheint mir in der Schule für den Computer sehr viel angemessener zu sein als die des „*geduldigen Lehrers*". Heranwachsende[96] geheimnissen in Computer Kenntnisse und Fähigkeiten herein, die diese weitgehend personifizieren[97]. Wir sollten diese irrationale Sicht nicht noch verstärken. Wenn Aufklärung bedeutet, Systeme durchsichtig und somit verständlich zu machen, dann kann Programmieren eben dazu verhelfen, indem es den Schülerinnen und Schülern Rechner als Maschinen zeigt, die so gesteuert werden, dass sie verständnislos Teile dessen automatisch verrichten, das die Programmierer selbst können und verstanden haben.[98]

In projektartigen Unterrichtsphasen mit unterschiedlicher Aufgabenverteilung ergibt sich das Voneinander, ohne eine Hierarchie zwischen den Betroffenen zu erzeugen. Auch wenn hier auf diese Arbeitsform nicht näher eingegangen werden kann[99], bleibt festzustellen, dass die dafür erforderliche Organisation der Zusammenarbeit einer-

---

[90] [Kla98c] S. 4: *„Lehrer guter Schulen richten ihr Augenmerk in mindestens gleichem Umfang auf die leistungsstärkeren wie auch auf die leistungsschwächeren Schüler."*
[91] [Fri95] S. 33
[92] [His94] S. 14
[93] gemeint als „spielerische Freiräume"
[94] s. a. "the peer-learning aspect" in [Dan00]
[95] s. auch [Schw96] Teil B
[96] und nicht nur die
[97] s. nächster Abschnitt
[98] nach [Bus87] S. 17: *„In erster Näherung lässt sich also sagen, dass ein Computer genau die intelligenten Operationen ersetzen kann, die der Mensch regelhaft darstellen und symbolisieren kann."*
[99] siehe dazu z. B. [Mod91] oder [Schu99]

seits erhebliche erzieherische Bedeutung hat, andererseits in der Schule nur selten zu finden ist. Entsprechend rechtfertigen viele Autoren das Schulfach Informatik u. a. damit, dass diese Arbeitsform informatikspezifisch ist: *„Die kooperative Arbeit, die soziales Lernen befördert, ist Grundbaustein der Didaktik der Informatik."*[100] *„Die kooperativen Arbeitsformen sind so informatikspezifisch, dass sie im Rahmen einer wissenschaftspropädeutischen Ausbildung unverzichtbar erscheinen."*[101]

Das *Füreinander-Lernen* fördert die von Klafki und Rolff geforderte *Solidaritätsfähigkeit*[102], die sich z. B. in *„der Entfaltung eines verantwortlichen Umgangs mit erworbenen Kompetenzen"*[103] manifestiert. Es setzt Kenntnisse der Möglichkeiten und Grenzen von Computersystemen voraus, die sich nicht nur fachlich ergeben, sondern ebenso aus ethisch-moralischen Überlegungen: *„Hierzu gehören nicht nur prinzipielle Grenzen des Computers, sondern auch menschliche Grenzen seines Einsatzes."*[104] Füreinander Lernen bedeutet auch, die erworbenen Kenntnisse und Fähigkeiten nicht nur für sich einzusetzen (z. B. für bessere Zensuren), sondern anderen zur Verfügung zu stellen (z. B. um das Gruppenprojekt erfolgreich abzuschließen), es erfordert Einsicht in das Privileg, lernen und sich persönlich weiterentwickeln zu können, in die Voraussetzungen, unter denen das geschieht, und in die Verpflichtungen, die sich daraus ergeben.

Veränderungen durch soziales Lernen zeigen sich nicht nur bei den Schülerinnen und Schülern, sondern auch bei den Unterrichtenden. So schreibt Berger in „*Informatische Weltbilder: Professionelle Konzeptionen von Mathematik-Informatik-Lehrern*": *„Informatiklehrer sind a posteriori innovativ. Die Beschäftigung mit und das Lehren von Informatik ‚macht' gewissermaßen innovativ."*[105]

1.2.4 Pragmatischer Konstruktivismus

Die bisher angestellten Überlegungen finden einen einheitlichen theoretischen Rahmen in der erkenntnistheoretischen[106] Denkschule des Konstruktivismus. Dieser basiert u. a. auf den Arbeiten Piagets[107] und nimmt eine Position ein, die der nichtklassischen Physik sehr ähnelt, indem er nicht mehr von einem erkennenden Subjekt ausgeht, das einer objektiven Wirklichkeit gegenüber steht, sondern feststellt, dass sich alles Wissen über eine – scheinbare – „*Wirklichkeit*" ausschließlich im erkennenden Subjekt findet, das sich ein Bild seiner (Um-)Welt konstruiert[108]. Der Konstruktivismus untersucht also *„die Art und Weise, wie wir Menschen unsere eigene Wirklichkeit erschaffen"*[109].

---

[100] [Hum00] S. 11
[101] [Schu99] S. 105
[102] [Kla85] S. 17
[103] [Bus87]
[104] [Koe95] S. 64
[105] [Ber98] S. 48
[106] oder lerntheoretischen – je nach Sichtweise
[107] und wohl auch auf denen Bruners: s. [Ben02] S. 5
[108] Christiane Floyd in „Das Mögliche ermöglichen" (1997): *„Der Konstruktivismus lehrt, dass unsere Erkenntnis durch Konstruktion zustande kommt, er macht damit keine Aussage über das Seiende."*
[109] Paul Watzlawick zitiert nach [BeB02]

> *Der Konstruktivismus ist keine Theorie des Seins, formuliert keine Aussagen über die Existenz der Dinge an sich, sondern ist eine Theorie der Genese des Wissens von den Dingen, eine genetische Erkenntnistheorie. Für den Konstruktivismus ist Wissen kein Abbild der externen Realität, sondern eine Funktion des Erkenntnisprozesses.*[110]

Ich kann hier aus Platzgründen nicht auf die erkenntnistheoretischen Folgen einer solchen Position eingehen, sondern will mich auf die offensichtlichen lerntheoretischen Folgerungen beschränken.

Dieter Wolff meint dazu:

> *Lernen wird als eine aktive Tätigkeit gesehen, die vom Lernenden selbstständig durchgeführt werden muss. Der Lernende konstruiert sich sein Wissen aus den angebotenen Informationen – Lernen wird als „kreativer Konstruktionsprozess" gesehen.*[111]

In dieser Sicht ist Wissen nicht direkt vermittelbar, sondern muss durch Aktivitäten der Lernenden erworben werden[112]. Diese entwickeln, ausgehend von ihrem Vorwissen[113], ein Modell der erfahrenen Wirklichkeit, das „passt", also zu keinen direkten Kollisionen mit der Erfahrung führt[114]. Da an die entwickelten Vorstellungen von der Welt nur die Forderung gestellt wird, zu *passen,* werden die von verschiedenen Lernenden konstruierten mentalen Wirklichkeiten nicht identisch sein, müssen sich noch nicht einmal ähneln. Sie müssen nur *gültig*[115] sein – jede auf ihre Art.

> *Constructivism claims each individual necessarily creates cognitive structures […]. Furthermore, it claims that each individual will perform the construction differently, depending on his or her pre-existing knowledge, learning style and personality traits. Hopefully, the construction is viable […]. Unfortunately, but perhaps inevitably, many users construct non-viable models.*[116]

Der Grad der erforderlichen Gültigkeit des Wissens hängt nun davon ab, mit welcher Intensität eine Auseinandersetzung mit dem Bereich der Welt erfolgt, über den dieses Wissen erforderlich ist. Bleibt der Kontakt oberflächlich, so genügen auch oberflächliche Konstrukte, um die Kontakte erfolgreich – jedenfalls nicht negativ – zu bewältigen. Je eingehender die Beschäftigung mit einem Bereich wird, desto häufiger werden Kollisionen des bisher konstruierten Modells dieses Bereichs mit der Erfahrung erfolgen, und desto genauer wird das nach diesen Kollisionen umkonstruierte Modell auf diesen Bereich passen.

Die konstruktivistische Lerntheorie sieht Lernen also als eine Art „Verschärfungsprozess", in dem ein anfangs ungenaues – aber nicht falsches – Bild der Wirklichkeit durch aktive Auseinandersetzung zunehmend präzisiert – also „schärfer" – wird.[117]

---

[110] Rolf Schulmeister, zitiert nach [BeB02]

[111] [Wol97]

[112] [Thi97] S. 12: *„Lernen ist nicht Übernahme von Wissen, sondern aktives Aufbauen von Wissensstrukturen, ein aktives Konstruieren."*

[113] [Wer98]*: „Einfach gesagt umfasst diese Auffassung, dass eine Person nicht von außen zu einer bestimmten Reaktion veranlasst bzw. determiniert werden kann, sondern dass immer die interne Struktur der Person bestimmt, wie sie sich mit Anregungen, die aus dem umgebenden Milieu kommen, auseinandersetzt. Jede Form der Beeinflussung […] muss sich damit auseinandersetzen, dass es keine direkten, instruktiven Interaktionsbeziehungen geben kann."*

[114] Diese Art der „Passung" entspricht der evolutionären Anpassung an die Umwelt.

[115] und nicht „wahr"

[116] [Ben02]

[117] Hier werden Einflüsse der evolutionären Erkenntnistheorie spürbar, etwa die Beschreibung des Weltbilds einer Amöbe von Konrad Lorenz.

> *Lernen heißt, mentale, kognitive Landkarten zu konstruieren, die immer mehr detailliert und verfeinert werden. Nicht sequentiell vom Einfachen zum Komplexen voranschreiten[118], sondern Gesamtstruktur konstruieren lassen, die im Laufe des individuellen Lernprozesses an Schärfe gewinnt, d. h. Gesamtheit vor Detail.*[119]

Eine Änderung in diesen „Landkarten" – neues Wissen – wird konstruiert, wenn Störungen auftreten, also das bisherige Modell zu Misserfolgen führt. Damit stören Probleme nicht den Lernprozess, sondern sie verursachen ihn: *„Lernschwierigkeiten und Probleme sind nicht möglichst schnell abzustellen, sondern bieten die Chance, die wesentlichen Fragen und damit das Thema tiefer zu verstehen."*[120]. Da Störungen von den Lernenden selbst als störend empfunden werden müssen, erfordert Lernen einen engen Bezug der Lernenden zum Thema. Im günstigen Fall kann dieser schon vorhanden sein, z. B. weil das Thema in Beziehung zur Erfahrungswelt der Lernenden steht. Anderenfalls muss er z. B. aus der praktischen Auseinandersetzung mit dem Thema entstehen[121]. Bleibt die Beziehung zwischen Lernenden und Thema indifferent, dann bleiben diese gegenüber Störungen gleichgültig und haben keinen Anlass, neues Wissen zu konstruieren. *„Ereignisse im Unterricht, die nicht als Perturbationen wirken, führen auch nicht zu strukturellen Umwandlungen subjektiver Systeme und können somit auch keine Lernprozesse in Gang setzen."*[122] Das verlagert einen Teil der Verantwortung für den Erfolg und/oder Misserfolg von Lernprozessen hin zu den Lernenden. Zwar müssen die Unterrichtenden für Unterrichtsinhalte motivieren, aber es wird auch von den Lernenden die Bereitschaft und die Fähigkeit erwartet, Interesse zu entwickeln. Bleibt Passivität und Desinteresse vorherrschend, dann sind Lernerfolge ausgeschlossen[123].

> *Man braucht in der Tat gar nicht sehr tief in das konstruktivistische Denken einzudringen, um sich darüber klar zu werden, dass diese Erkenntnistheorie zur einer alleinigen Verantwortlichkeit des denkenden Menschen für sein Denken, Wissen und Tun führt. Behavioristen schieben alle Verantwortung nach wie vor auf die Umwelt, Soziobiologen einen großen Teil auf die Gene; daher ist eine Lehre ungemütlich, die andeutet, dass wir unsere Lebenswelt uns selbst zu verdanken haben.*[124]

Was folgt nun daraus für die Lehrenden?

> *Unterrichten ist somit der Versuch der Anregung von komplexen affektiv-kognitiven Systemen, die nach ihrer eigenen Logik operieren. Sie sind selbstreferentiell, weil jede ihrer Handlungen auf ihre Struktur zurückwirkt und diese bestätigen oder verändern kann. Aufgrund der funktionalen Beziehungsstruktur zwischen Organismus und Umwelt werden ständig Wirklichkeitskonstruktionen als Routinen angewendet, teilweise neu entwickelt, überprüft, bestätigt, verworfen usw. Diese aktive Beziehungsgestaltung zwischen Subjekt und Umwelt ist der Ansatzpunkt jeder Beeinflussung.*[125]

---

[118] [DLP02]: *"Encourage students to make meaning by breaking wholes into parts. Avoid starting with the parts to build a 'whole.'"*
[119] [Thi97] S. 9
[120] [Thi97] S. 9
[121] [Boy00]: *"A central constructivist criticism of traditional formal teaching is that it is disembedded from the students' experience outside the classroom."*
[122] [Dah00]
[123] Diese einfache Einsicht entspricht natürlich auch der Erfahrung.
[124] Ernst von Glaserfeld 1995, zitiert nach [Dor99]
[125] [Wer98]

Die Unterrichtenden unterstützen den Konstruktionsprozess durch Anregungen, Hilfen und Kontrollen[126]. *„Dem Lehrer ist es unmöglich, seine Kenntnisse dem Lerner direkt weiterzugeben. Vielmehr hilft er dem Lerner durch sein Tun, durch Hinweise, Fragen und Informationen, selbst Wissen zu konstruieren.“*[127] Wissen entsteht, indem frühere Erfahrungen des Lernenden mit neuen Situationen in Beziehung gesetzt werden.[128] Soll der Prozess gesteuert ablaufen, dann müssen die Lehrenden natürlich Kenntnisse über dieses Vorwissen besitzen. Die Vorerfahrungen der Lernenden werden normalerweise bei einer weniger detaillierten Betrachtung des Lerngegenstands als in der aktuellen Unterrichtssituation gebildet worden sein. Sie sind deshalb in der neuen Sicht nicht mehr gültig, auf unterschiedliche Art unvollständig, aber nicht falsch. In diesem Zusammenhang sind dann einfache Tests, die zur Leistungsmessung meist ungeeignet sind, als Hilfsmittel zur Diagnostik des Vorwissens in einem neuen Licht zu sehen. Gemeint sind damit Aufgaben wie die Ermittlung der Resultate bestimmter Befehlfolgen[129]: z. B. ***read(A,B); read(B); write(A,B,B);***. Werden diese nicht zur Leistungsmessung eingesetzt, sondern um ungültige Vorstellung (hier: vom Variablenkonzept) offen zu legen, dann haben sie in diesem Zusammenhang durchaus ihren Sinn.[130]

> *Entscheidend für den konstruktiven Prozess des Wissenserwerbs sind bereits bestehende Wissensstrukturen; der Lernende konstruiert sein Wissen, indem er die Erfahrungen in Abhängigkeit von diesem Vorwissen und auf Grundlage bestehender Überzeugungen interpretiert.*[131]

Damit ist *„Lehren nicht die Vermittlung und Lernen nicht die Aneignung eines extern vorgegebenen ‚objektiven' Zielzustandes, sondern Lehren ist die Anregung des Subjekts, seine Konstruktionen von Wirklichkeit zu hinterfragen, zu überprüfen, weiterzuentwickeln, zu verwerfen, zu bestätigen etc. Das bedeutet auch, eine Vielfalt von Lernwegen zu ermöglichen, wie sie in Formen des offenen Unterrichts, des projektorientierten Unterrichts und des forschenden Lernens umgesetzt wird. Vielfalt bedeutet die Öffnung der Räume.“*[132] Die unterschiedlichen – und den Lehrenden nur unvollkommen bekannten – mentalen Konstrukte der Lernenden erfordern die Auseinandersetzung mit dem Lerngegenstand auf unterschiedliche Weise:

- einerseits, um dem unterschiedlichen Vorwissen gerecht zu werden,
- andererseits, um durch die Vernetzung des neu Gelernten mit unterschiedlichen Bereichen der Vorerfahrung ein möglichst valides Bild der Realität zu konstruieren.

> *Als erstes sollten daher den Lernenden die Gründe vermittelt werden, warum bestimmte Weisen des Handelns und Denkens als wünschenswert betrachtet werden. Daraus folgt notwendig die Erklärung der spezifischen Zusammenhänge, in denen das zu erwerbende Wissen angeblich funktionieren soll.*[133]

---

[126] [Hen96]: *“The goals of the teacher are to engage the learner in active participation, problem solving, interdisciplinary work, reflection and discussion.”*
[127] [Thi97] S. 8
[128] [Wer98]
[129] nach [Ben02] S. 7
[130] Analoges ist aus dem Physikunterricht für elementare Fragen aus der Mechanik bekannt: Hier können auf ähnliche Weise aus Erfahrung erworbene Vorstellungen der Aristotelischen Mechanik („Impetusprinzip“) der Newtonschen Mechanik gegenüber gestellt werden.
[131] [Bra97]
[132] [Wer98]
[133] Glasersfeld 1997, zitiert nach [Bra97]

Wissen wird über Handlungen gewonnen[134], indem sich die Lernenden aktiv mit dem zu Erlernenden auseinandersetzen. Dazu muss der Unterricht so angelegt sein, dass aktives Handeln für die Lernenden möglich wird. Verstehen wir Reden im Sinne Hartmut von Hentigs als „Probehandeln“[135], so sind projektartige Unterrichtsphasen, die zu gemeinsamem Planen und Agieren anregen, dafür besonders geeignet[136]. In diesen werden die Unterrichtenden überwiegend als Berater und Anreger tätig werden, die den Unterrichteten Rückmeldungen über die Qualität[137] ihrer Denkkonstrukte geben[138] und durch Rückfragen, Gegenbeispiele und Erweiterungen neue kognitive Konflikte verursachen.

> *Im Gegensatz zu gängigen „Eintrichterungstheorien“ wird eine konstruktivistische Didaktik das Lernen als einen Prozess der Selbstorganisation von Wissen verstehen. Das bedeutet, jeder Schüler wird neue Lerninhalte zunächst in Zusammenhang zu seinen Erlebnissen, seiner Weltsicht setzen. Dieser Prozess ist damit relativ, individuell und unvorhersagbar. Ziel der Lehrer muss sein, möglichst reichhaltige kommunikationsorientierte Umgebungen zu schaffen, welche die subjektiven Erfahrungsbereiche ansprechen und gleichzeitig neue „Rätsel“ beinhalten, die pragmatisch, interaktiv und kreativ zur Selbstorientierung einladen. Die Kunst des Lehrers besteht darin, zwischen der ursprünglichen Wirklichkeitskonstruktion des Lerners und derjenigen, die wissenschaftlich und gesellschaftlich als konsensfähig gilt, eine Kette von optimalen Diskrepanzen vorzusehen, die von den Lernern als Erwartungswiderspruch erlebt und durch Versuch und Irrtum produktiv überwunden werden.*[139]

Was ist nun pragmatisch am Konstruktivismus?

Weil Konstruktivisten auf „wahre“ Aussagen verzichten, stellen sie konsequenterweise auch ihre eigene Theorie nicht als absolut gültig dar, sondern verstehen sie als eine im Rahmen des Erforderlichen gültige Näherung. Der Konstruktivismus ist deshalb bemerkenswert undogmatisch. Auch wenn man seinen ontologischen Aussagen skeptisch gegenüber steht, können die sehr handfesten lerntheoretischen Folgerungen ziemlich problemlos auf den Unterricht angewandt werden. Vor allem aber können sie bestehende Lehr- und Lernformen ergänzen, ohne sie zu verdrängen: *“Note that Bruner seems to agree with the constructivist viewpoint that unfettered discovery is not helpful; he distinguishes between episodic empiricism, where the student accumulates unconnected facts, and cumulative constructionism, where the discovery is organized.”*[140] Der Konstruktivismus kann als umfassendes lerntheoretisches Modell fungieren, das aus unterschiedlichen Quellen, z. B. aus der Reformpädagogik stammende, bewährte Konzepte zusammenfasst. Diese Art des Unterrichtens entspricht im Fach Informatik auch weitgehend den Erwartungen und Fähigkeiten der Lernenden (s. Anhang).

Begreifen wir nun „pragmatisch“ im Sinne von „undogmatisch“ und „unter den gegebenen Umständen realisierbar“, dann versucht die *„pragmatisch-konstruktivistische Lernkultur eine Symbiose zwischen der traditionell lehrerzentrierten Instruktionspädagogik und einer lernerzentrierten Konstruktionspädagogik herzustellen. Je nach Schulfach, Wissensgebiet und Lehrplan sollen dabei eher instruktionelle und eher konstruktivistische Lehr-*

---

[134] [Dah00]
[135] und Denken als „Probereden“
[136] [Boy00]: *“Dialogue and the negotiation of meaning provide the basis for the individual to develop, test and refine their ideas.”*
[137] [Dor99]
[138] Außerdem werden sie ggf. deutlich machen müssen, dass Lernen auf Eigenaktivitäten der Lernenden basiert!
[139] [Mül98]
[140] [Ben02] S. 5

Lern-Phasen einander abwechseln[141] *und ergänzen.“*[142] Durch den Konstruktivismus geprägte Unterrichtseinheiten fügen sich in bestehenden Unterricht ein[143], sollen ihn *„durch Elemente und Projekte erweitern, die explorativen und selbstgesteuerten Wissenserwerb fördern. Dabei geht es letztendlich auch darum, dass Lernende Wissen so aufbauen, dass sie es möglichst dauerhaft anwenden können“*[144].

Die pragmatisch-konstruktivistische Sicht gestattet es demnach, bestehende Unterrichtskonzepte graduell zu verändern in eine Richtung, die

- dauerhafte Lernerfolge ermöglicht (statt der kurzfristigen Akkumulation von Fakten),
- Strukturen betont (statt Details),
- Voraussetzungen für selbstständiges lebenslanges Lernen[145] schafft,
- anwendungsorientiert arbeitet
- und die Lernenden als individuelle Subjekte in den Mittelpunkt des Lernprozesses stellt.

Sie kommt damit zu ähnlichen Ergebnissen wie Wilfried Herget – aus anderen Überlegungen – für den Mathematikunterricht: *„… Ein derart veränderter Unterricht ist eben eher prozessorientiert, weniger produktorientiert, ist mehr auf die Schülerinnen und Schüler hin ausgerichtet und weniger von Lehrerdominanz geprägt. Querverbindungen […] werden stärker herausgestellt, die fachliche Systematik tritt etwas zurück.“*[146]

Dass die Arbeit der Unterrichtenden damit nicht gerade einfacher wird, betont Mordechai Ben-Ari: *“The task of the teacher in the constructivist paradigm is significantly more difficult than in the classical one, because guidance must be based on the understanding of each student’s currently existing cognitive structures.”*[147] Andererseits erhalten damit Qualitäten einen höheren Stellenwert, an denen es im deutschen Schulwesen offensichtlich mangelt, wie sich gerade in der letzten Zeit[148] dramatisch gezeigt hat. Konstruktivistische Denkweisen zeigen hier einen Weg zu mehr Verständnis und mehr Anwendungsbezug[149]. Der pragmatische Aspekt ermöglicht einen gleitenden – und somit bei Misserfolg auch reversiblen – Übergang zu verändertem Lehrverhalten. Er zeigt eine klare Richtung, ohne Systembrüche zu erzwingen, lässt punktuelle Erprobungen und Erfahrungen zu, die bei Erfolg erweitert werden können. Pragmatischer Konstruktivismus erscheint mir deshalb als ein Weg, im Sinne Sigrid Schuberts *„durch eine Folge wohldurchdachter kleiner Schritte“*[150] den Informatikunterricht kontinuierlich zu verbessern.

---

[141] [Hen96]: *“Even so, this still leads to a change in the role of the teacher, where the teacher needs to create situations, where the student can work on useful problems, where the teacher provides counter-examples compelling reflection and reconsideration of solutions, and where the teacher is acting as mentor stimulating initiative and research rather then being a lecturer who transmits ready-made solutions.”*

[142] [Mül98]

[143] *„Instruktion und Konstruktion sind allenfalls in 'ideologischen' Auseinandersetzungen ein Gegensatz, in der Praxis dagegen eine sinnvolle Ergänzung“* (Mandl/Reinmann-Rothmeier 1996).

[144] [Ple98]

[145] [Dan00] *“… we must also equip them (the students) for continual learning subsequently”*

[146] [Her98] S. 24

[147] [Ben02]

[148] durch die PISA-Studien

[149] statt auf die Krise mit „mehr Pauken“ zu reagieren.

[150] [Schu99] S. 24

1.3 Folgerungen für die Informatikdidaktik

1.3.1 Zur Auswahl fachdidaktischer Themen

Schülerinnen und Schülern erleben Unterricht ausschließlich als Fachunterricht, dessen einzelne Stunden in ihrer Gesamtheit die angestrebte Allgemeinbildung erzielen sollen. Fachunterricht kann in diesem Sinne allgemein bildend wirksam werden, weil nach Klafki *„die Aufnahme und Aneignung von Inhalten stets verbunden ist mit der Formung, Entwicklung und Reifung von körperlichen, seelischen und geistigen Kräften."*[151] Diese Fachinhalte müssen dazu geeignet ausgewählt werden. „*Es ist Aufgabe der Fachdidaktik Informatik, die von der Fachwissenschaft gewonnenen Erkenntnisse – unter Einbeziehung von Erkenntnissen der Allgemeinen Didaktik – für die Schule (...) zu transformieren und aufzubereiten, um somit den Unterrichtenden eine ausreichende Vorbereitung für das Unterrichten allgemein bildender informatischer Inhalte vermitteln zu können."*[152] Bezeichnet man Unterrichtsstunden und ihre Komponenten als „*feinste Verästelungen*", die im Rahmen der Entscheidungs- und Bedingungsfelder des Unterrichts detailliert geplant werden, dann lautet der Anspruch der Berliner Didaktiker in einer Formulierung von Wolfgang Schulz:

> *Von den Wurzeln ihrer Voraussetzungen her bis in die feinsten Verästelungen ihrer Folgen hinein versucht die lehrende Intelligenz im System der Didaktik ihre Handlungssituation aufzuklären, um zu wissen und einer aufgeklärten Gesellschaft gegenüber verantworten zu können, was sie tut.*[153]

Die lehrende Intelligenz hat es dabei nicht ganz leicht, denn weder sind die Ergebnisse der Allgemeinen Didaktik eindeutig, noch besteht Konsens darüber, welche Inhalte der Fachwissenschaft allgemein bildenden Charakter haben[154]. *„Entsprechend existiert derzeit noch keine theoretische Basis der Informatikdidaktik, die zu einer annähernd geschlossenen Theorie führt."*[155]

Es gibt also keinen eindeutigen Weg von allgemeinen Zielen hin zu konkretem Unterricht. Fachdidaktiker müssen sich deshalb entscheiden, welche allgemein bildenden Ziele sie verfolgen wollen[156], und sie müssen dann unter den in der Schule unterrichtbaren Fachinhalten und -methoden diejenigen auswählen, mit denen sie diese Ziele zu erreichen glauben. Fachdidaktik hat immer mit Wertung und Entscheidung zwischen verschiedenen Alternativen zu tun. Sie ist „objektiv", also ohne den Einfluss persönlicher Gewichtungen, nicht möglich.[157] Fachdidaktische Entscheidungen müssen deshalb unter Angabe und vor dem Hintergrund ihrer Ziele begründet werden. Sie sind dann daran zu messen, ob sie diese Ziele erreichen. In diesem Sinne sind sie dann auch verifizierbar.[158]

---

[151] zitiert nach [Tho00] S. 2
[152] [Tho00] S. 10
[153] [Hei68] S. 17
[154] [Hey95] S. 48: *„Allgemeinbildungskonzepte bringen (...) selbst keinen Unterricht hervor und schon gar keinen Fachunterricht. Und aus einem Allgemeinbildungskonzept lässt sich, für sich genommen, keineswegs deduzieren, wie ein der Allgemeinbildung verpflichteter Fachunterricht im Detail auszusehen hätte."*
[155] [Tho00] Vorwort
[156] In dieser Arbeit ist das z. B. durch Auswahl entsprechender Literatur geschehen.
[157] Das gilt ganz allgemein für Schule.
[158] Es ist bequem, Didaktik nach Comenius als *Kunst* aufzufassen, weil man damit dem Begründungszwang weitgehend entgeht.

Ist ein Katalog allgemein bildender Ziele[159] ausgewählt, dann hat die Fachdidaktik zu zeigen, dass mindestens einige dieser Ziele durch das Lehren fachlicher Inhalte und Methoden erreicht werden können und dass diese Ziele entweder nur innerhalb des betrachteten Faches, zumindest aber dort wesentlich besser als in anderen zu verwirklichen sind. Die fehlende Eindeutigkeit erschwert allerdings diese Auswahl, und ob mit den ausgewählten Inhalten wirklich die angestrebten Ziele zu erreichen sind, muss empirisch geprüft werden. Dieser zeitaufwändige Prozess ist für die Informatik, die ihre Inhalte sehr schnell ändert, nicht annähernd abgeschlossen: *„Bis heute kann die Hochschulinformatik (...) nicht auf solides und fundiertes Vorwissen im Bereich der Informatik aus der Schule zurückgreifen."*[160]

Diese Situation ist eigentlich nicht zu verstehen, denn an den Unterrichtsschwerpunkten hat sich in den vergangenen Jahren erstaunlich wenig geändert[161], wie Wilfried Herget[162] zeigt. Wenn trotzdem in der Schulinformatik die *„nahezu beliebige Vielfalt der vorgeschlagenen Themen"*[163] nicht abreißt, zeigt das doch nur, dass die bisherigen entweder den didaktischen Ansprüchen nicht genügten, oder dass überhaupt nicht ernsthaft versucht wurde, didaktische Relevanz zu erreichen. Es ist – besonders im Rückblick – erstaunlich zu sehen, mit welchem Vertrauen die an der didaktischen Diskussion Beteiligten offensichtlich glauben, dass die jeweils neuen Themen länger bestehen werden als die alten. Es ist aber noch erstaunlicher, dass in vielen Fällen genau diejenigen, die die schnellen Themenwechsel beklagen, nach wenigen Sätzen schon wieder bei der Diskussion „neuer" Programmiersprachen o. Ä. angelangt sind.[164] Peter Hubwieser klagt zu Recht, dass *„die Vermittlung allgemein bildender Inhalte der Informatik nicht in dem Maße berücksichtigt wurde, wie es im Rückblick angebracht gewesen wäre."*[165] Wenn also nach den Erfahrungen der Vergangenheit von neuen Themen keine wesentlich größere Relevanz als von den alten zu erwarten ist, dann scheint es Erfolg versprechender, die alten in Ruhe zu erproben, zu verbessern und aufbauend auf Unterrichtserfahrungen im Sinne von Sigrid Schubert „evolutionär" zu ändern.[166] Insbesondere scheint es mir sinnvoll zu sein, Veränderungen nicht so sehr bei den Fachinhalten vorzunehmen, sondern im konstruktivistischen Sinne die im Laufe der Zeit sich ändernden Vorerfahrungen der Schülerinnen und Schüler zu berücksichtigen und zu nutzen, um mithilfe geeigneter Zwischenschritte den Weg von diesen zu den angestrebten Zielen des Unterrichts durchschaubarer und damit auch gangbarer zu machen als bisher.

---

[159] Marco Thomas ([Tho00] S. 10) zählt unter den *Zielsetzungen des Schulsystems* die *Ausbildung von Allgemeinbildung*, die allgemeine *Vorbereitung auf die Berufswelt* und die *Vermittlung einer Allgemeinen Studierfähigkeit* auf. Nach den genannten gängigen Kriterien für Allgemeinbildung lassen sich die letzten beiden Punkte allerdings als Unterpunkte der Allgemeinbildung auffassen.

[160] [Hum00] S. 1

[161] Wohl aber an ihrer Gewichtung.

[162] [Her94] S. 31

[163] [Lav98] S. 19

[164] z. B. ebenfalls [Lav98]

[165] [Hub97] S. 42

[166] [Schu99] S. 8: *„Es geht deshalb nicht so sehr darum, etwas ganz anderes im Informatikunterricht zu lehren, sondern auf den guten Lehrerfahrungen aufzubauen und Schwächen zu überwinden."*

1.3.2 Zum Aufgabenkatalog von Bussmann und Heymann

Die Ergebnisse der Allgemeinen Didaktik erzeugen zwar keinen Unterricht, sie liefern aber *Kriterien*, an denen sich Fachunterricht messen lässt.

> *Sie liefern (...) einen pädagogischen Orientierungsrahmen, durch den die Ansprüche unterschiedlicher Interessengruppen, etwa der Fachwissenschaftler, der Wirtschaft, des Staates, kritisierbar werden. Allgemeinbildungskonzepte werden der Idee der Allgemeinbildung nur gerecht, wenn sie helfen, eine Balance zu finden zwischen dem Recht des einzelnen auf seine Personwerdung, seinen individuellen Interessen und Bedürfnissen auf der einen Seite, und der allgemeinen Kultur gesellschaftlichen Anforderungen und Notwendigkeiten auf der anderen. Im Wechselspiel mit einschlägigen fachdidaktischen und fachlichen Überlegungen sollte sich also mittels eines hinlänglich ausgearbeiteten Allgemeinbildungskonzepts konkretisieren lassen, was für einen „allgemein bildenden Unterricht" in dem betreffenden Fach charakteristisch ist.*[167]

Damit ist der Bezug zur Fachdidaktik präzise beschrieben. Der aus diesen Überlegungen folgende Aufgabenkatalog von Bussmann und Heymann wurde schon genannt. Er wird in der fachdidaktischen Literatur weitgehend akzeptiert und eingehend in Bezug auf das Schulfach Informatik analysiert[168]. Hier sollen zu den einzelnen Punkten nur kurze Zusammenfassungen und einige mir wichtig erscheinende Anmerkungen gemacht werden.

Zur Vorbereitung auf künftige Lebenssituationen

Heymann versteht darunter *„Lebensvorbereitung im engen Sinne"*[169], also einen Begriff, der nicht so weit gemeint ist, dass er *alles* umfasst, was in der Schule geschieht oder geschehen könnte.[170] Peter Hubwieser übernimmt die Interpretation von Bussmann und Heymann[171] als Aufgabe

> *Qualifikationen [zu] vermitteln,*
>
> *(a) die zur Bewältigung realer und auf absehbare Zeit in unserer Gesellschaft verbreiteter Lebenssituationen beitragen,*
>
> *(b) die nicht auf die Ausübung eines bestimmten Berufes hin ausgerichtet sind,*
>
> *(c) von denen anzunehmen ist, dass sie nicht gleichsam automatisch, nebenher von jedem Heranwachsenden erworben werden und*
>
> *(d) die durch eine gewisse Universalität, also Anwendbarkeit in sehr verschiedenen Situationen gekennzeichnet sind.*

Er betont die Rolle der Automatisierung geistiger Tätigkeiten durch den Computer[172] und die unterschiedlichen Rollen[173], in denen Menschen dem Computer gegenüber treten. Da von den sechs angegebenen Rollen vier direkt computerbezogene Arbeitsplätze betreffen, kollidiert diese Interpretation allerdings heftig mit Heymanns Aussagen, der betont, dass eigentlich *„die große Mehrheit derjenigen [gemeint ist], die auf gänzlich andere Weise ihren Lebensunterhalt verdienen werden"*. Auch Heymanns

---

[167] [Hey95] S. 48
[168] z. B. [Hey95] S. 49, [Hub00] S. 57ff, [Tho00] S. 11, [Leh82] S26ff, [Bau96a] S. 169
[169] [Hey95] S. 48
[170] Also keinen „Freibrief" für alles.
[171] [Bus87] S. 5
[172] nach [Hop96], S. 8
[173] als Nutzer, Betroffener, Planer, Entwickler, ...

Forderung, *„alle Niveaus, keinesfalls nur das Gymnasium“* im Auge zu behalten, legt eine andere Auslegung nahe. Gabriele Lehmann geht in ihrem Aufsatz „*Ziele im Informatikunterricht*“[174] direkt auf den Aufgabenkatalog ein, sieht den Beitrag der Informatik in den Aspekten *„Verständnis für informationelle Prozesse, Einblick in gesellschaftlich relevante Anwendungen, Problemlösefähigkeiten, Befähigung zum Denken in Abläufen und Zusammenhängen“*[175] und beschreibt damit Aufgaben, die üblicherweise von der Informationstechnischen Grundbildung (ITG) zu leisten sind[176]. Die betrifft allerdings ***alle*** Schülerinnen und Schüler, so dass diese ITG-Aufgaben schon deshalb nicht in einen Informatikkurs der Oberstufe verlagert werden dürfen. Verstehen wir die Lebensvorbereitung im Sinne von Heymann, dann trägt ein Informatikkurs durch die Vermittlung *vertiefender* Einsichten in Arbeitsweise und Auswirkungen von Informatiksystemen natürlich ***auch*** zu diesem Aspekt bei, es kann aber nicht seine zentrale Aufgabe sein. Trotzdem sollte aber auch beim Unterrichten spezifischer Fachinhalte die Frage von Ludger Humbert beachtet werden: *„Wie viel Wissen braucht die ‚mündige Bürgerin', um ihre Rechte wahrnehmen zu können und um sachkundig an zentralen Diskussionen über die Gestaltung der Zukunft mitzuwirken?“*[177]

Die Aufgabe der Lebensvorbereitung kann nach Bussmann und Heymann missbraucht werden, wenn nur ***eine*** mögliche Zukunft prognostiziert wird, die dann als quasi naturgesetzlich festgeschrieben dargestellt wird und auf die vorbereitet werden muss. Dabei geht der Gesichtspunkt der Gestaltbarkeit der Zukunft z. B. durch Bildung der Agierenden oder politisches Engagement verloren und damit die Bereitschaft, Verantwortung für diese Gestaltung zu übernehmen und die Zukunft solidarisch zu gestalten.[178]

Eine Folge des Postulats der Lebensvorbereitung ist, dass die materiale hinter die formale Bildung zurücktritt[179], schon deshalb, weil in einer schnell veränderlichen Gesellschaft in der Schule erworbenes Wissen nicht mehr für ein ganzes Berufsleben, besser: noch nicht einmal für längere Abschnitte davon, ausreicht. Folglich müssen möglichst die Inhalte, anhand derer die Qualifikationen gewonnen werden, und natürlich die Qualifikationen selbst auf neue Situationen übertragbar sein. In diesem Zusammenhang wird die aus dem Konstruktivismus folgende Priorität der Lernmethoden vor den Inhalten wieder interessant.[180] Notwendig ist ein ***Transfer***, der zwar nach Bender nicht nachweisbar ist[181], trotzdem aber vorhanden sein kann. Je offener die Zukunft, auf die vorbereitet werden soll, aufgefasst wird, desto allgemeiner sind die darauf „vorbereitenden“ Qualifikationen, und desto schwerer wird es, deren „Vorbereitungstauglichkeit“ zu begründen. Da sehr allgemeine Qualifikationen an praktisch allen Inhalten erworben werden können, besteht die Gefahr, dass die Auswahl der Unterrichtsinhalte unter diesem Aspekt beliebig wird. Obwohl die Aufgabe so augenscheinlich sinnvoll erscheint, taugt sie doch nur eingeschränkt als Unterrichtskriterium.

---

[174] [Leh92] S. 27

[175] Leider ist sie bei der Präzisierung dieser Begriffe schon nach zwei Sätzen mitten in PROLOG.

[176] Dass Heymann Ähnliches meint, ergibt sich aus seinen Ausführungen zu den lebensvorbereitenden Aspekten der Mathematik in [Hey95] S. 50ff

[177] [Hum00] S. 2

[178] Derzeit ist die Warnung mehr als aktuell, wenn im Rahmen der Globalisierung soziale Rechte oder moralische Prinzipien als „nicht mehr haltbar“ dargestellt werden, weil die Entwicklung anderes erfordert.

[179] [Bus87] S. 6

[180] [Boy00]: “*A focus on the learning and reproduction of factual knowledge is thus not helpful. Both teachers and students will have to constantly relearn their subject skills and knowledge. The constructivist emphasis on learning problem-solving skills rather than factual knowledge thus seems appropriate.*“

[181] [Ben95] S. 13: *„Tatsächlich sind mir aus der allgemeinen pädagogisch-didaktischen Forschung keine breiten, dauerhaften Transfer-Erfolge bekannt.“*

Anzumerken ist, dass die auf niedrigem Niveau angesiedelten „lebensvorbereitenden“ Inhalte der ITG – im Sinne von Bedienerfertigkeiten – ein sehr zweischneidiges Schwert sind: Einerseits erhalten sie ihre Bedeutung durch die schnelle Verbreitung von Informatiksystemen, andererseits relativiert gerade diese Verbreitung ihre Relevanz für die Schule, weil sie teilweise schon flächendeckend an anderer Stelle erworben werden, bevor das Erlernen ihrer Beherrschung in Schulen wirklich verbreitet ist.[182] ITG-Inhalte beruhen meist auf Prognosen der zukünftigen Entwicklung – und die stimmen erfahrungsgemäß oft nicht[183]. Gerechtfertigt werden entsprechende ITG-Einheiten meist dadurch, dass über die Benutzerfertigkeiten hinaus die Schülerinnen und Schüler dazu geführt würden, Informatiksysteme und ihre Auswirkungen „kritisch zu beurteilen“ o. Ä. Da einerseits Kritik- und Beurteilungsfähigkeit eine intensive Beschäftigung mit der Materie und erhebliche Fachkenntnisse voraussetzt und andererseits für die ITG nur wenige Stunden bereit stehen, in denen bestenfalls etwas Wissen und Übung vermittelt werden können, erscheint mir der erhobene Anspruch kaum verwirklichbar zu sein. Es ist also kein Wunder, wenn die GI feststellt, dass die ITG „gescheitert“ sei.[184] In übertragenem Sinne gelten diese Aussagen teilweise auch für Inhalte des Informatikunterrichts, z. B. für die Gestaltung von „*Mensch-Maschine-Schnittstellen*“ bei Pascal-Programmen[185], die lange Zeit für wichtig gehalten wurden. Auch die Kritik, dass mit modernen grafischen Programmentwicklungssystemen Programme einfach „zusammengeklickt“ werden, zieht nur, wenn die Oberfläche schon den wesentlichen Teil des Programms darstellt.

Zur Stiftung kultureller Kohärenz

Kulturelle Kohärenz wird erzeugt durch die Tradierung gewisser schulischer Inhalte, u. a. um die Gesprächsfähigkeit zwischen den Generationen zu erhalten[186], und scheint ein geschlossenes Curriculum zu erzwingen, wie es früher weitgehend üblich war[187]. In einer schnelllebigen, sich rapide ändernden Gesellschaft ist eine solche Schule nicht mehr denkbar. Kulturelle Kohärenz kann deshalb nur die Bewahrung eines *gewissen Teils* schulischer Themen über die Zeit bedeuten, und sie weist auf die Bedeutung zumindest mittelfristig wichtiger Lehrinhalte hin[188]. Sie berührt damit ein zentrales Problem der Informatikdidaktik.

Die Einordnung informatischer Systeme in die Technik- und Kulturgeschichte, das Eingehen auf ihre Wurzeln aus Mathematik und aus anderen Wissenschaften macht die Entwicklung der Disziplin deutlich, liefert Anknüpfungspunkte an traditionelle schulische Themen und hebt so die Isolierung vom sonstigen schulischen Umfeld auf. An dieser Stelle sind auch die Ausführungen von Hoppe und Luther zum *„Bemühen um die Automatisierung geistiger Tätigkeiten“*[189] hilfreich, und Baumanns Auflistung der *„Ursprünge der Informatik“*[190] liefert reichlich Quellen. Da historische Gesichtspunkte aber meist nur ein Aspekt des Unterrichts unter vielen bleiben, muss für die restlichen Themen die Relevanz der gestellten Aufgabe ebenfalls bestimmt werden.

---

[182] das, obwohl S. Schubert noch 1999 feststellt: *„Tastatur und Maus sind als Eingabegeräte* [für Schüler!] *neu und gewöhnungsbedürftig.“* [Schu99] S. 56
[183] s. dazu [Mod92a] S. 191 über „Waschmaschinen“
[184] [GI01] S. 1
[185] s. dazu [Mod91] S. 36
[186] [Bus87] S. 9
[187] und damit wäre die Informatik „draußen“.
[188] Hubwiesers *Vereinheitlichung der Fachsprache* ist damit sicherlich nicht gemeint: [Hub00] S. 63
[189] [Hop96] S. 8
[190] [Bau96a] S. 50ff

Heymann überschreibt seine auf die Mathematik bezogenen Ausführungen zu diesem Punkt erweiternd als *„Stiftung kultureller Kohärenz durch Vermittlung zentraler Ideen“*[191] und weist damit der Informatikdidaktik einen Weg. Greifen wir auf Schwills „***fundamentale Ideen** der Informatik*“[192] zurück, anhand derer sich Unterrichtsinhalte bewerten lassen, dann haben wir damit offensichtlich eine Methode gefunden, die diesem Gedanken entspricht. Weil fundamentale Ideen sich *auf jedem Niveau* innerhalb eines Spiralcurriculums wieder finden sollten, sind sie auch und gerade für die Oberstufenkurse wichtig, weil sie Anknüpfungspunkte einerseits an vorhergehenden Unterricht, andererseits nach draußen zur Lebenswelt der Schülerinnen und Schüler herstellen[193], und nebenbei erzwingen sie durch das Zeitkriterium eine gewisse Konstanz der Unterrichtsinhalte.

Zur Weltorientierung

Zur Weltorientierung gehört exemplarisches Wissen über die Welt und über die erkannten Zusammenhänge. Da hier der materiale Begriff der Bildung überwiegt, fügen die einzelnen Schulfächer in ihrem Bereich als wichtig geltende Beispiele dem Gesamtbild bei. Inzwischen ist wohl unbestritten, dass hierzu auch ein informatischer Beitrag gehört. G. Lehmann betont die ***fächerübergreifende Arbeit*** im Informatikbereich, die dazu führt, dass die in anderen Fächern gebildeten Facetten zusammengefügt werden können. P. Hubwieser weist auf die Bedeutung des Informationsbegriffs[194] hin, der eine ähnlich tief greifende Funktion habe wie der Materie- (besser: Energie-) Begriff. Damit wird der eigenständige Beitrag der Informatik zum Aufbau eines Weltbildes deutlich.

Heymann diskutiert diese Aufgabe für die Mathematik und betont dabei ihren Modellierungsaspekt und ihren Anwendungsbezug:

> *Mathematikunterricht sollte vielfältige Erfahrungen ermöglichen, wie Mathematik zu Deutung und Modellierung, zum besseren Verständnis und zur Beherrschung primär nicht-mathematischer Phänomene herangezogen werden kann.*[195]

Ersetzen wir den Begriff „Mathematik“ durch „Informatik“, dann sind wir mit der ***Modellbildung*** mitten in einem zentralen Bereich der Informatik[196]. Da ein Weltbild selbst ein Modell der Welt ist, ist Einsicht in den Modellbildungsprozess wichtig für die kritische Einschätzung der eigenen Ansichten. Die ***Anwendung informatischer Methoden*** auf unterschiedliche Gebiete zeigt einerseits die Bedeutung von Modellen im Alltag, andererseits durch das aktive Modellieren ihr Zustandekommen, ihre Möglichkeiten und Grenzen und trägt so in Analogie zu einem etablierten Fach zur Weltorientierung bei. Auf diesen Aspekt wird noch einzugehen sein.

Bedenken wir, dass nach Bruner und im konstruktivistischen Sinne durch ein Fach Strukturen im Denken der Lernenden aufgebaut werden müssen, in die aktuelles Wissen eingeordnet wird, dann dienen übergeordnete Begriffe wie der der Information und ihrer Verarbeitung sicherlich diesem Ziel. Sie können den Unterrichteten als

---

[191] [Hey95] S. 51
[192] z. B. in [Schw96]
[193] und so im konstruktivistischen Sinne das Lernen erleichtern.
[194] nach Breier
[195] [Hey95] S. 52
[196] und der Mathematik. [His00] S. 6: *„Es wird damit deutlich, dass das das Thema ‚Modellbildung’ für den Mathematikunterricht zwingend eine außerhalb der Mathematik liegende und damit fächerübergreifende Perspektive hat: Im vorliegenden Kontext betreiben wir Modellbildung also i. d. R. außerhalb der Mathematik, davon nicht losgelöst dann innerhalb der Mathematik Strukturbildung.“*

Leitlinien der Entwicklung dienen und sind in jedem Fall einer ungeordneten Anhäufung von Details vorzuziehen. Andererseits reduziert die Beschränkung auf *einen* Aspekt natürlich die Weite eines Fachs – es sei denn, der Aspekt ist so allgemein gefasst, dass sich darunter praktisch alles einordnen lässt. In diesem Fall aber verlöre der Aspekt seine strukturbildende Eigenschaft.

Statt also *ein* „Paradigma" herauszustellen, sollte ein Fach über *ein Spektrum* von wenigen spezifischen Sichten auf die Welt verfügen, die es gestatten, die fachspezifischen Eigenheiten herauszustellen, die Fülle des Wissens zu ordnen und insbesondere neues, später zu erwerbendes Wissen einzuordnen. Die materiale Dimension der fachspezifischen Bildung sollte dann diese Sichten erzeugen und stärken; Fakten dienen als Exempel. Beschreibt man die Sichten durch ***fundamentale Ideen***, dann bilden diese das langfristig gültige Gerüst, das *durch* unterschiedliche Bausteine erzeugt wird, *in das* unterschiedliche Bausteine exemplarisch eingeordnet werden und *das* Ordnung in spätere Erfahrungen bringt. In diesem Sinne liegt kein so großer Unterschied zwischen der vorherigen Aufgabe und dieser. Beide erfordern Einsicht in Zusammenhänge, die anhand von Wissen erworben werden. Die Stiftung kultureller Kohärenz betont dabei den zeitlichen Aspekt, die Weltorientierung die Strukturbildung.

Zur Anleitung zum kritischen Vernunftgebrauch

Kritikfähigkeit ist als emanzipatorisches Element des Unterrichts grundlegend für mündige Bürgerinnen und Bürger und damit für die Demokratie westlicher Ausprägung. Zur Kritik gehört die Fähigkeit, selbst zu denken[197] und die Ergebnisse dieses Denkens zu artikulieren und gegenüber anderen zu vertreten. Dazu sind die Fähigkeit zur Erfassung einer Situation eines Systems, also ***Modellbildung***[198], folgerichtiges Denken und Diskursfähigkeit erforderlich. Da nicht jede Situation neu ist und nicht jede oder jeder permanent völlig neue Modelle entwickeln kann, gehört zum kritischen Denken die Möglichkeit, auf vorhandenes Wissen zurückzugreifen und vorhandene Erfahrungen auf die neue Situation zu übertragen, also wieder ***Transfer***. Weil Denken an bestimmten Inhalten gelernt wird, muss auch der Transfer auf andere Inhalte ausdrücklich geübt werden[199]. Es ist also wenig sinnvoll, wenn nur ein Beispiel pro Problemkreis im Unterricht auftaucht, weil dann die Gefahr besteht, dass die erworbenen Fähigkeiten nur mit diesem Beispiel verbunden bleiben.

Folgerichtiges Denken kann in Teilen durch Maschinen erfolgen, wenn Fakten und Schlussregeln geeignet aufbereitet werden. Es kann aber nur automatisiert werden, falls eindeutige Beziehungen vorherrschen. Wenn überwiegend diffuses Wissen vorliegt und Bewertungen sowie Erfahrungen den weiteren Gang der Argumentation bestimmen, dann muss schon ein menschliches Gehirn am Diskurs beteiligt sein[200]. Erfahrungen mit maschinellem Folgern, die Kommunikation über formale Sprachen mit Maschinen und das Modellieren realer Systeme führt zu Erfahrungen über Möglichkeiten und Grenzen der Automatisierbarkeit und damit zum Wert spezifisch menschlicher Eigenschaften. Folgen auf diese Erfahrungen Betrachtungen über die

---

[197] Daraus kann dann durchaus die Kritik an anderen und an z. B. gesellschaftlichen Zuständen folgen.

[198] [Mon98] S. 27: *„Andererseits gehört zum Allgemeinwissen eines mündigen Bürgers des 20. Jahrhunderts, dass er weiß, wie die Wissenschaften zur Erkenntnis kommen: Modellbildung muss in der Schule thematisiert und praktiziert werden."*

[199] [Hey95] S. 54

[200] Die Existenz von Expertensystemen schränkt diese Aussage nicht ein, weil ja auch dort zuerst die Erfahrungen von menschlichen Experten in Regelsysteme „gegossen" werden.

Grenzen des Einsatzes von Informatiksystemen, die durchaus nicht nur durch die Mathematik gegeben sind, sondern auch im ethisch-moralischem Bereich liegen oder einfach im Unvermögen, für einen Bereich computergeeignete Modelle zu erzeugen[201], dann fördert die auf Kenntnis beruhende Einsicht in die Problematik der Nutzung von Informatiksystemen sicherlich die Mündigkeit im allgemein bildenden Sinne.

Einsicht in die Möglichkeiten und die Grenzen von Computersystemen ergeben sich m. E. nicht so sehr aus der Betrachtung der Rollen, in denen Menschen diesen Systemen gegenüber stehen[202], sondern aus den Rollen, die Computer in dieser Gesellschaft spielen. Informatiksysteme als *Rechner*, *Datenverarbeiter*, *Roboter*, *Symbolverarbeiter*, *Medium* und *Kommunikationspartner*, *Planer*, *Entscheider* oder *künstliche Intelligenz*[203] übernehmen ehemals menschliche Aufgaben, und damit wird auch ein Teil der menschlichen Verantwortung delegiert. Aus Betrachtungen dieser Rollen ergeben sich einige fachliche Einsichten; viel wichtiger aber erscheint mir, dass sich daraus fachlich fundierte Fragen entwickeln, die einer politischen Beantwortung harren. Auch wenn im Unterricht diese Fragen nur ansatzweise beantwortet werden, so liegt der eigentliche Wert des Unterrichts darin, dass die Schülerinnen und Schüler lernen, sich von den Verantwortlichen nicht mit Nebensächlichkeiten abspeisen zu lassen, sondern die entscheidenden Punkte erkennen.

Zur Entfaltung von Verantwortungsbereitschaft und zur Einübung in Verständigung und Kooperation

Verantwortungsbereitschaft kann man nicht so einfach lehren wie materiale Bildungsaspekte, man muss sie erwerben. Da es sich um eine Haltung handelt, kann die Schule dazu beitragen, diese Haltung zu fördern, indem sie den Schülerinnen und Schülern zeigt, dass Handlungen Folgen haben, von denen viele durchaus nicht vorausgesehen und damit beabsichtigt sein müssen. Sie sollte Beispiele dafür bringen, dass Entwicklungen nicht zwangsläufig ablaufen, sondern dass es Alternativen gibt, die Zukunft also durch Übernahme von Verantwortung beeinflussbar bleibt. *„Gebildet ist, wer (...) (sach-) kompetent ist und darüber hinaus von seiner Sachkompetenz verantwortungsvoll Gebrauch macht.“*[204] Sie kann den Schülerinnen und Schülern durch soziales Lernen Erfahrungen verschaffen, die deren Bereitschaft steigern, diese Haltung zu entwickeln[205]. Die Entfaltung von Verantwortungsbereitschaft hängt deshalb kaum von Unterrichtsinhalten ab, sie ist eine Frage der Unterrichtskultur[206]. Zur Bedeutung des sozialen Lernens mit Zielen wie Kommunikationsfähigkeit und Teamfähigkeit, die in der Diskussion um Schlüsselqualifikationen einen zentralen Platz einnehmen[207], habe ich mich schon vorher geäußert.

---

[201] [Mod91] S. 67
[202] wie Hubwieser meint: s. [Hub00] S. 63
[203] [Mod92a] S. 196ff
[204] [Bus87] S. 13
[205] Bemerkenswerterweise findet man zu dieser und vergleichbaren Aufgaben kaum Äußerungen in der fachdidaktischen Literatur.
[206] [Hey95] S. 55
[207] [Hey95] S. 50

Zur Stärkung des Schüler-Ichs

Bei P. Hubwieser und G. Lehmann finden sich erstaunliche Aussagen darüber, dass der Informatikunterricht den Schülerinnen und Schülern das Gefühl vermitteln soll, *„Computer zu beherrschen, statt von ihnen beherrscht zu werden“*[208] bzw. *„sich der Überlegenheit gegenüber dem Computer bewusst zu werden“*[209]. Meiner Erfahrung nach sind unsere Unterrichteten weit entfernt davon, Computer als gleichberechtigt, geschweige denn als überlegen anzusehen, und auch die Computer sind noch nicht annähernd so entwickelt, dass sich diese Frage stellt. Die Frage nach der Beherrschung *durch Computer* lenkt m. E. eher davon ab, dass Herrschaft nicht durch technische Systeme erfolgt, sondern durch diejenigen, die diese einsetzen, sowie politische Verhältnisse, die dieses zulassen.

Der Wert des Informatikunterrichts in allgemein bildenden Schulen ergibt sich m. E. zu einem großen Teil aus den Defiziten anderer Fächer, die stoffüberfrachtet den Schülerinnen und Schülern keinen Raum mehr lassen, um deren eigenen Stärken und Schwächen zu erproben, um zu probieren, ob und wie sie eigene Ideen entwickeln[210], konkretisieren und realisieren können, und ob sie so etwas gerne tun[211]. Die Schule bewegt sich viel zu viel entlang ausgetretener, gut erprobter Wege. Sie unterrichtet die *Ergebnisse* der Ideen anderer, meist außergewöhnlicher Koryphäen, neben denen sich die Schülerinnen und Schüler – und nicht nur die – klein vorkommen müssen. Sie zeigt nur unvollkommen die Wege zu diesen Ergebnissen, die Umwege und Irrungen, die dabei auftraten. Sie stellt eine „perfekte Denkwelt“ vor, mit „eleganten“ Schlüssen und ausgefeilten Argumenten, ohne zu zeigen, dass diese Eleganz erst das Endprodukt, die aufpolierte Version einer Rohfassung ist, die unter Mühen gewonnen wurde. Der direkte Weg zu den Endergebnissen, der Umwege und Schwierigkeiten versteckt, verhindert oft (Denk-)Kollisionen mit den Vorerfahrungen der Lernenden und damit Anlässe für das Umkonstruieren und Erweitern des bisher Gelernten, also neues Lernen. Erst die Kenntnis des Kärrnerwegs kann junge Leute ermutigen, sich ebenfalls auf den Weg zu machen in der Hoffnung, dass einige ihrer Ergebnisse dann auch irgendwann den Weg in die „Hochglanzwissenschaft“ finden werden. Natürlich findet materiale Bildung an den Glanzpunkten von Wissenschaft und Kunst statt, aber selbstverständlich darf nicht der Eindruck erweckt werden, dass Glanzpunkte der einzige und erste Ertrag von Wissenschaft sind.

Persönlichkeitsstärkend[212] ist also die Erfahrung, selbstständig und erfolgreich ein Problem bearbeiten und lösen zu können. *„Wenn man anerkennt, dass eines der allgemeinen Ziele einer humanen und demokratischen Erziehung (...) die Befähigung zur Selbstbestimmung sein muss, dann muss man zugleich Selbsttätigkeit als notwendiges pädagogisches Prinzip anerkennen.“*[213] Soll dieses sinnvoll geschehen, dann muss das bearbeitete Problem so gewählt werden, dass von den Bearbeitern selbst, hier von den Schülerinnen und Schülern, die Arbeit als lohnend empfunden wird. Dazu müssen sie an der Auswahl der Problemstellung mindestens beteiligt sein – ein typisches Merkmal für ***Projektunterricht***. Fast ausgeschlossen werden kann die reine Aufgabenstellung durch den Lehrer, wenn sie nicht nur Übungszwecken dient. Sie wird auch von den Schülerinnen und Schülern nicht gerade geschätzt (s. Anhang). Selbst gewählte

---

[208] [Hub00] S. 64
[209] [Leh92] S. 30
[210] [Hey95] S. 49: *„ ... es gilt, selbst zu denken und der Kraft des eigenen Verstandes zu vertrauen.“*
[211] [Hey95] S. 50: *„Schülerinnen und Schülern muss in der Schule Raum gewährt werden, ihre eigenen Bedürfnisse und Möglichkeiten zu entfalten, ihre spezifischen Stärken zu entdecken und zu entwickeln.*
[212] ab jetzt weitgehend nach [Mod91] S. 29
[213] [Kla98b] S. 2

Problemstellungen bedeuten natürlich, dass verschiedene Lernende auch an verschiedenen Problemen arbeiten werden (selbst wenn es sich um unterschiedliche Teile des gleichen Projekts handelt). Damit müssen die Schülerinnen und Schüler sich zumindest teilweise die für die Lösung benötigten Informationen selbst beschaffen, denn sie können nicht erwarten, dass der Lehrer gleichzeitig in verschiedenen Arbeitsgruppen so präsent ist, wie er es bei einheitlicher Aufgabenstellung wäre. *„Gerade, wenn Unterricht zur Entwicklung der selbstständigen Lernfähigkeit beitragen soll, dann darf er nicht durchgehend so organisiert sein, dass alle Kinder einer Klasse ständig unter direkter Leitung des Lehrers zur gleichen Zeit immer die gleichen Aufgaben in der gleichen Weise und im gleichen Lerntempo bearbeiten sollen."*[214] Schülerinnen und Schüler sind also als Problemlöser sehr viel autonomer als im normalen Unterricht, der Lernprozess ist sowohl im Lerntempo wie in der Art des Lernens stärker individualisiert und bereitet lebenslanges Lernen[215] effektiv vor.

Wesentlich erscheint mir, dass ein zuerst als unüberschaubar erscheinendes Problem angegangen werden kann, ohne vorher zu wissen, wie die endgültige Lösung aussehen wird. Im Gegensatz zu anderen Fächern lehrt die Informatik Methoden, um allgemein Probleme zu lösen, aber nur selten, wie ein bestimmtes Problem behandelt wird. Machen Schülerinnen und Schüler mehrfach die Erfahrung, dass die erlernten Lösungswege für sie gangbar sind, dann sollten sie so viel begründetes Selbstbewusstsein gewinnen, dass sie auch auf anderen Gebieten aktiv werden.

Wichtig ist die Erfahrung, dass Fehler zum normalen Lösungsverfahren dazugehören, dass aus ihnen gelernt werden kann[216]. Dieser Prozess des „Debugging" ist in der Informatik effizient möglich, weil schon erste, noch fehlerhafte Lösungsversuche realisiert und auf dem Rechner getestet werden können. Schülerinnen und Schüler lernen, dass zur erfolgreichen Problembewältigung sowohl Ideen wie sorgfältige Arbeit erforderlich sind. Keines von beiden genügt – von Ausnahmen abgesehen – alleine. Die Erfahrung, dass sich Arbeit lohnt, dass Durchhaltevermögen und Sorgfalt zum Ziel führen, auch ohne von anderen, z. B. dem Lehrer, ständig geleitet zu werden, können andere Fächer so kaum vermitteln. Der Versuch, durch frühes Eingreifen des Lehrers Fehler zu vermeiden, wäre deshalb auch unter diesem Gesichtspunkt fehl am Platz.[217]

Die Behandlung komplexerer Probleme kann von einzelnen Schülerinnen und Schülern jedenfalls in der Unterrichtszeit nicht mehr geleistet werden. Die schon deshalb notwendige Arbeitsteilung erfordert die Fähigkeit, im Team zu arbeiten, Absprachen zu treffen und einzuhalten, dasselbe Problem unter verschiedenen Aspekten zu untersuchen, Vorschläge anderer zur Kenntnis zu nehmen, ihnen also erst einmal zuzuhören (obwohl sicherlich eigene Ideen vorliegen), Kompromisse zu schließen – kurz Teamfähigkeit. *"Projects have, in our opinion, a potential of being an excellent form for covering the whole spectrum from theory to skill including social competence and personal development for computer science students (...)."*[218] Informatik fördert in solchen Unterrichtsphasen damit nicht nur die Kommunikation mit dem Rechner, sondern vor allem das Gespräch und die Zusammenarbeit mit anderen Lernenden.

---

[214] [Kla98b] S. 5

[215] [Ste01]: *"Computer science education, moreover, must seek to prepare students for lifelong learning that will enable them to move beyond today's technology to meet the challenges of the future."*

[216] besonders aus konstruktivistischer Sicht.

[217] [Ben02] S. 15: *"You must provide as much opportunity as possible for individual reflection (for example, analysis of errors) and social interaction (for example, group labs)."*

[218] [Dan00]

Die Abbildung von Teilen der Welt auf Programme im Rechner erfordert in erheblichem Maße Abstraktionsfähigkeiten. Die Reduktion der Realität auf die einerseits für das behandelte Problem vermuteten wesentlichen Aspekte, andererseits auf noch bewältigbare Größenordnungen, also Modellbildung, bedingt immer Verluste an Präzision, so dass die gefundenen Lösungen höchst kritisch gewürdigt werden müssen. In unserem Fall ist also Kritik an der eigenen Arbeit erforderlich, und die Fähigkeit zur Selbstkritik kann kaum hoch genug eingeschätzt werden.

Wenn diese Aspekte den von mir so gesehenen Stellenwert haben, dann ist es unverzichtbar für den Informatikunterricht, die entsprechende Zeit und geeignete Anreize und Hilfen für selbstständiges Lernen zur Verfügung zu stellen, und diese Zeit fehlt dann natürlich fürs „Stofflernen“. Eine wesentliche Folge dieser Überlegungen ist deshalb, bei didaktischen Planungen den erforderlichen Zeitbedarf angemessen, d. h. höchst kritisch zu berücksichtigen.[219] Ein wesentlicher Teil der informatikdidaktischen Literatur tut das m. E. nicht und landet deshalb bei Vorschlägen, deren „Vorbereitungsschritte“ schon den Zeitrahmen ausschöpfen, so dass für die eigentlichen Inhalte nicht mehr angemessen Zeit zur Verfügung steht.

Unter dem Aspekt der Motivation zur selbstständigen Arbeit gewinnen aktuelle Unterrichtsinhalte, die wegen der Beständigkeit des Fachs eigentlich mit Vorsicht zu betrachten sind, einen neuen Wert. Jugendliche sind natürlich an den neuesten Entwicklungen interessiert. Können diese gleichwertige „traditionelle“ Inhalte ersetzen und sind alle Beteiligte – u. a. auch die Unterrichtenden – willens und in der Lage, den Austausch sinnvoll vorzunehmen, dann – aber auch nur dann – spricht nichts dagegen, im Unterricht „aktuell“ zu bleiben.

1.3.3 Kriterien für Unterrichtsinhalte

Die vorangehenden Überlegungen ergeben einen relativ kurzen Katalog von vorerst allgemeinen[220] Kriterien, die zur Auswahl von Informatik-Unterrichtseinheiten herangezogen werden können.

Die Inhalte des Informatikunterrichts müssen

- ***formale Bildung ermöglichen***
  Da formale Qualifikationen vorrangig sind, müssen die gewünschten Qualifikationen anhand der Inhalte erwerbbar sein. Eine nur durch fachimmanente Überlegungen scheinbar erforderliche Ausweitung der Inhalte über diesen Zweck hinaus ist zu vermeiden.
- ***anhand fundamentaler Ideen ausgewählt werden***
  Da einerseits das konstruktive Lernen der Schülerinnen und Schüler effizient in Strukturen erfolgt und andererseits diese Strukturen erst aufgebaut werden müssen, sollten die Inhalte danach ausgewählt werden, ob sie fundamentale Ideen des Fachs verdeutlichen. Es reicht dabei nicht, dass diese Ideen nur den Unterrichtenden bei der Planung ihres Unterrichts deutlich sind, sondern auch und gerade die Schülerinnen und Schüler müssen im Bewusstsein dieser Ordnung neue Inhalte in das entstehende fachliche Raster einbauen.

[219] Aussagekräftig werden solche Schätzungen immer dann, wenn der Unterricht von anderen als den Entwicklern gehalten wird, wenn sich also neue Inhalte verbreiten.
[220] Der Katalog muss natürlich noch für die einzelnen Fachthemen konkretisiert werden.

- ***einen Bezug zu Schlüsselproblemen haben***
  Weil Informatiksysteme eine immer größere Bedeutung im politischen, wirtschaftlichen, gesellschaftlichen und privaten Bereich gewinnen, weil einerseits Problemlösungen in diesen Bereichen mit ihrer Hilfe versucht werden und andererseits Probleme durch sie erst entstehen, weil sie neue Chancen eröffnen und den Zugang zu Informationsressourcen bestimmen, kann und muss der Informatikunterricht Klafkis Forderung an die Allgemeinbildung gerecht werden und einen vertiefenden Zugang zu diesen Themen liefern. Er kann das auch problemlos, weil sich die Bezüge einerseits zwanglos aus den fachlichen Fragen ergeben und sich andererseits fachliche Fragestellungen direkt aus diesen Bezügen gewinnen lassen.
- ***Transfer einüben***
  Die fundamentalen Ideen müssen an unterschiedlichen Beispielen erprobt werden, denn erst in der Übersicht kann sich ja das Fundamentale dieser Ideen zeigen. Die Breite dieser Basis muss ausreichend sein, um den Transfer von erlernten Qualifikationen aktiv zu üben.
- ***Modellbildung ermöglichen***
  Informatikmethoden müssen auf unterschiedliche Gebiete angewandt werden, um einerseits den Modellbildungsprozess deutlich zu machen, andererseits die Beschäftigung mit Folgen der Anwendung zu ermöglichen. Die Gebiete sollten so gewählt werden, dass sich Anknüpfungspunkte zur Lebens- und Erfahrungswelt der Lernenden ergeben, so dass neues Wissen auf diese aufbauen kann.
- ***Projektunterricht unterstützen***
  Die Unterrichtsmethode der Wahl ist projektartiges Arbeiten[221], in dem einerseits Formen des sozialen Lernens geübt werden, andererseits phasenweise selbstständiges Arbeiten der Schülerinnen und Schüler möglich ist.
- ***als Bausteine eines offenen Kanons dienen***
  Die erforderliche relative Konstanz der Unterrichtsinhalte legt es nahe, einen Kanon von Inhalten festzulegen, der die gewünschten Eigenschaften hat. Dabei sollte das Ziel des Unterrichtens dieser Inhalte deutlich sein, damit bei Bedarf einzelne Themen durch aktuellere Ausprägungen ersetzt werden können, die dem gleichen Ziel dienen[222].

[221] Die Abschwächung erfolgt, weil es vielleicht etwas übertrieben ist, grundsätzlich „richtigen“ Projektunterricht zu fordern.

[222] In [Mod91] hatte ich das vor zehn Jahren so formuliert:

Der Informatikunterricht an einer Schule ist gerechtfertigt, wenn

1. *Problemlösungsmethoden unterrichtet werden, die die Schülerinnen und Schüler in die Lage versetzen, selbstständig eigene Lösungen auf einem geeigneten Niveau zu finden, zu testen und zu verbessern.*
2. *die behandelten Probleme geeignet sind, gesellschaftspolitische und moralische Aspekte des Computereinsatzes fachlich begründet zu diskutieren.*
3. *die behandelten Probleme zumindest phasenweise so bearbeitet werden, dass soziales Lernen und Arbeiten gefördert wird. Projektarbeit ist dafür eine geeignete Form.*
4. *die für die praktische Arbeit notwendigen Kenntnisse und Fertigkeiten einerseits solide unterrichtet werden, andererseits aber nicht so in den Vordergrund treten, dass sie den Zweck, aus dem sie unterrichtet werden, verdrängen. Inhalte sind den Methoden klar nachgeordnet.*

2. Spezielle fachdidaktische Fragen

Wenn die Allgemeine Didaktik der Fachdidaktik auch einen Rahmen setzt, dann fehlen immer noch allgemeinere fachdidaktische Vereinbarungen, vor deren Hintergrund Teilbereiche informatischer Bildung zu bewerten, zu gewichten und in Unterricht umzusetzen sind. Die im ersten Abschnitt gefundenen sehr allgemeinen Kriterien zur Auswahl von Unterrichtsinhalten ermöglichen es, aus dem umfangreichen Katalog möglicher Themen geeignete auszuwählen. Von diesen ist dann aber noch nicht klar, mit welchem Ziel und mit welcher Akzentuierung sie im Unterricht behandelt werden sollen. Um diese Fragen zu klären, benötigt man eine etwas detailliertere Vorstellung davon, was genau denn z. B. unter einem offenen Lernzielkatalog, fundamentalen Ideen oder der Rolle des Programmierkurses im Unterricht zu verstehen ist. Diesen Fragen widmet sich der zweite Abschnitt.

2.1 Offener Lernzielkatalog und ortsnahe Curriculumentwicklung

Die Informatikdidaktik wird die Frage zu klären haben, wie ein sich schnell weiterentwickelndes Fach wie die Informatik[223] in einem sehr trägen System wie dem öffentlichen Schulwesen langfristig existieren kann. Diese Frage stellt sich verschärft in der aktuellen Situation[224], wird aber auch bei verbesserter Lehrerausbildung virulent bleiben. Schulfächer brauchen Standards, die üblicherweise durch die Universität gesetzt werden[225], und damit Beständigkeit. Die Unterrichtenden brauchen Zeit, um Erfahrungen zu machen und auf dieser Grundlage ihren Unterricht zu verbessern. Das Fach braucht Veränderungen, um seine motivierende Wirkung bei den Unterrichteten[226] zu erhalten und neuen Entwicklungen zu folgen, aber diese Veränderungen brauchen, wenn sie zu Standards werden sollen, wiederum sehr viel Zeit. Es muss also geklärt werden, wie in einem relativ konstanten Rahmen aktuelle Veränderungen möglich sind, ohne den Bezug zur „informatischen Umwelt“ in den Schulen zu verlieren.

> *The rapid evolution of computer science requires an ongoing review of the corresponding curriculum. Given the pace of change in our discipline, the process of updating the curriculum once a decade has become unworkable. The professional associations in this discipline must establish an ongoing review process that allows individual components of the curriculum recommendations to be updated on a recurring basis.*[227]

Anregungen zu diesem Problemkreis können in Klafkis Ausführungen zur Handlungsforschung und zur schulnahen Curriculumentwicklung[228] gefunden werden. Voraussetzung für diese Konzepte ist ein *offener Lernzielkatalog*, der durch das Primat formaler Bildung im Bereich der Schulinformatik ohnehin nahe gelegt wird. „Offen“

---

[223] [Tuc96]: *“The distance between the foundations of computing and its research and application frontiers is considerably shorter than in many other fields. As a result, the curriculum in computer science and engineering faces constant evolutionary pressure to integrate new critical developments.”*

[224] unter den erschwerten Bedingungen mit leeren öffentlichen Kassen und überwiegend unzureichend ausgebildeten Unterrichtenden.

[225] im fachlichen Bereich z. B. durch die weitgehend standardisierte Grundausbildung, die einen Kanon von Inhalten bereitstellt, der Fachlehrerinnen und –lehrern dann zur Verfügung steht.

[226] und nicht zuletzt auch bei den Unterrichtenden

[227] [Ste01]

[228] [Kla76] S.123ff

bedeutet in diesem Zusammenhang, dass es offen gehalten werden kann, an welchen materialen Inhalten die gewünschte formale Bildung erworben wird. Das darf aber nicht als Beliebigkeit missverstanden werden: Im Gegenteil kann ein offenes Konzept nur dann Bestand haben, wenn die Anforderungen an die formale Bildung so präzise formuliert sind, dass im Detail entschieden werden kann, welche materialen Inhalte diesen Anforderungen gerecht werden. Da in der kategorialen Bildung materiale und formale Momente eine Einheit bilden, also formale Bildung stets an Inhalte gebunden ist, muss entschieden werden können, welche der vielen möglichen Inhalte die jeweiligen Anforderungen erfüllen. Solche Präzisierung erfolgt am deutlichsten am Beispiel. Ist in diesen Beispielen der Bezug der Fachinhalte zu den formalen Zielen des Unterrichts hinreichend deutlich, ist also die Stellung des Fachinhalts in der Unterrichtsfolge klar umrissen, dann kann „offen" entschieden werden, welche alternativen Inhalte die gleiche Rolle übernehmen können. Diese Entscheidung kann dann in weiten Bereichen auch „vor Ort" fallen, also durch die Unterrichtenden und/oder die Fachkonferenzen der Schulen, ohne an den Zielen des Unterrichts und der Ausrichtung des Faches Wesentliches zu ändern.

Ein offener Lernzielkatalog setzt also Präzision bei den formalen Zielen des Unterrichts voraus. Er muss im Detail Inhalte enthalten, die es erlauben, diese Ziele zu erreichen[229]. Die Inhalte bestimmen aber nicht den Unterricht, sondern sind das Medium, in dem und anhand dessen Unterricht stattfindet. Die einen bestimmten Unterrichtsgang zu einem bestimmten Zeitpunkt bildende Folge von Inhalten ist dann als eine aktuelle Manifestierung eines allgemeineren Konzepts zu verstehen, das zu einem anderen Zeitpunkt in anderer Ausprägung, aber gleicher Intention verwirklicht wird. Ist die Stellung der Inhalte im Unterrichtsgang klar definiert, dann brauchen alternative Inhalte, also austauschbare Komponenten, zum Zeitpunkt der Verabschiedung des Konzepts nicht genannt, noch nicht einmal bekannt zu sein. Es genügt, ihre Funktion zu beschreiben. Ein entsprechend konzipierter Unterricht verhindert m. E. die unerträgliche Situation, bei jedem Wechsel auch eher unbedeutender Inhalte neu in eine Grundsatzdiskussion über die Ziele des Faches einzutreten, die dann natürlich die kontinuierliche Arbeit und Verbesserung des schon Erreichten verhindert. Sigrid Schubert formuliert das so:

> *„Informatik sollte nicht ständig neu begründet werden, aber die Entwicklung zum erfolgreichen Schulfach ist fortzusetzen. Dazu sind gute Erfahrungen zu bewahren. Aus erkanntem Mangel folgt die Suche nach Elementen, die den Lehrgegenstand bereichern, ohne ihn zu überladen."*[230]

Im Schulinformatikbereich übersteigt die für die Verbreitung mancher neuer Inhalte erforderliche Zeit oft die Dauer, in der diese Inhalte für den Unterricht relevant sind[231]. Dadurch entfällt für diese Teilbereiche auch der übliche Weg, über didaktische Forschung, Schulversuche, Entwicklung und Erprobung von Materialien, Revision der Rahmenrichtlinien, Lehrerfortbildung usw. für die Verbreitung dieser Inhalte zu sorgen. Benötigt wird deshalb der offene Lernzielkatalog, um *ortsnah Curriculumentwicklung* in dem Sinne zu ermöglichen, dass vorhandene Inhalte gegen andere *von den Unterrichtenden selbst* ausgetauscht werden. Praktisch bedeutet dieses für die Lehrerinnen und Lehrer, dass sie für ihren eigenen Unterricht nicht nur Unterrichtsvorbereitung und -analyse im fachlich bekannten Rahmen und im bekannten

---

[229] Die in vielen Rahmenrichtlinien zu findende „Offenheit" in Form von sehr allgemein gehaltenen Formulierungen, die den Fachunterricht ggf. der Beliebigkeit ausliefern, sind das genaue Gegenteil davon.
[230] [Schu99] S. 6
[231] Genau dieses hat ja zur starken Zersplitterung des Faches geführt.

Umfang[232] zu betreiben haben, sondern dass eine Art Handlungsforschung[233] von ihnen erwartet wird, obwohl sie meist für diese Aufgabe nicht hinreichend qualifiziert sind. Nach Klafki führt diese Arbeit unter diesen Bedingungen *„bei allen Beteiligten notwendigerweise [zu] Enttäuschungen."* [234] Speziell für den Informatikunterricht hält er die Anforderungen an Lehrer „derzeit" nicht für erfüllbar.[235] Aber stimmt das unter allen Bedingungen?

Man findet die „schwierigen" Anteile des Informatikunterrichts eher bei den theoretischen oder abstrakten Inhalten des Faches als im schnell veränderlichen Bereich der Programmiersprachen und anderer Tools – und die Theorieteile ändern sich nun wirklich nicht täglich. Wenn die schnelle Änderung trotzdem überall beklagt wird, dann hat das seine Ursache m. E. auch darin, dass die „aktuellen" Inhalte des Unterrichts einen unverhältnismäßig hohen Anteil am Unterricht beanspruchen – und das liegt vermutlich an der überwiegend schlechten und/oder weitgehend fehlenden fachlichen Ausbildung der Unterrichtenden, denen die Theorieteile damit ziemlich fern liegen[236]. Entsprechend selten werden sie dann angemessen unterrichtet.[237] Rechnet man also zum offenen Lernzielkatalog einen angemessenen Teil Theorie und Grundlagen der Informatik, dann verbleiben bei den schnell veränderlichen Teilen eher Themen, die keinen überhöhten intellektuellen Anspruch erheben. Als Beispiel mag die Simulation digitaler Grundbausteine dienen: Die Frage, wie aus „Schaltern" programmierbare Systeme entstehen, gehört zum „Urgestein" der Schulinformatik. Auch die Simulation dieser Bauteile durch Softwarekomponenten ist ziemlich alt. Jeweils neu ist nur die aktuelle Ausprägung dieser Simulation, z. B. derzeit als ein exzellenter Anwendungsfall für OOP. Die Einarbeitung in eine neue Programmiersprache ist ab und zu bewältigbar – wenn man andere gut kennt. Die Benutzung einer integrierten Entwicklungsumgebung ist ohne großen Aufwand möglich – wenn man auch sonst intensiv mit Computern arbeitet. Der Umgang mit neuen Tools wie etwa einem SQL-Server erfordert etwas Zeit, aber keine neuen Konzepte[238]. Die *Erfahrungen* mit diesen Werkzeugen, die unbedingt nötig sind, um die Schülerinnen und Schüler im Unterricht angemessen zu unterstützen, erfordern allerdings erhebliche Arbeit und auch Arbeitszeit. Sie sind „nebenbei" nicht ständig neu zu erwerben.

Betreibt man schulnahe Curriculumentwicklung im genannten Sinne, dann reduziert sich die Handlungs*forschung* weitgehend auf den Erwerb von Erfahrungen mit neuen Werkzeugen, das Erlernen neuer technischer Details und die damit meist verbundene Materialienentwicklung. Wirklich neue Konzepte[239] tauchen eher selten auf – wie in anderen Fächern auch. Solche Änderungen im Detail und besonders die Mate-

---

[232] [Hei68] S. S22: *Zwei Situationen sind es vor allem, die die Fähigkeit des einzelnen Lehrers zu Reflexionen alltäglich herausfordern: Nach dem Unterricht muss dessen Analyse ihm helfen, klüger als vorher zu werden.(...) In der Analyse wird er später auch die Konstruktion prüfen, um die Ergebnisse der Prüfung wiederum in die Planung eingehen zu lassen.*

[233] [Kla85] S. 99

[234] [Kla76] S. 123

[235] [Kla85] S. 135 Da sich die Situation zwischenzeitlich nicht grundlegend geändert hat, bleibt diese Aussage gültig.

[236] Deutlich wurde dieses z. B. innerhalb der Lehrerfortbildung, wenn sich bei einem „Auffrischungskurs in theoretischer Informatik" sich die schon länger unterrichtenden Kolleginnen und Kollegen auffallend intensiv immer noch mit dem beliebten „Blumenautomaten" beschäftigten. Es entstand der Eindruck, dass trotz vorhandener Unterrichtspraxis die „Theorie" bei ihnen fast ausschließlich aus diesem Beispiel bestand.

[237] [Schu99] S. 22: *„Es ist nicht zu erwarten, dass die angehenden Lehrer das, was sie selbst nicht umfassend verstanden haben, an ihre zukünftigen Schüler vermitteln können."*

[238] Relationale Datenbanken und Abfragesprachen gehören schon lange zum Standard. Neu ist nur deren (kostenfreie) Verfügbarkeit für Schulen, z. B. mit MySQL.

[239] wie die „objektorientierte Programmierung"

rialentwicklung kann effizient durch ortsnahe Lehrerfortbildung[240] unterstützt werden, am besten unter Beteiligung der Universitäten, denn diese können auf Grund ihres Sachverstandes eine Sichtung und Bewertung neuer Entwicklungen vornehmen, die für Praktiker oft schon zeitlich nicht möglich ist[241] – und zwar in Kenntnis des offenen Lernzielkatalogs. In diesem Rahmen können die Universitäten dann auch „richtige" Handlungsforschung betreiben, *„Forschung beim Unterrichten unter Einmischung"*[242]. Gefordert ist deshalb immer noch eine *„schulnahe Curriculumentwicklung, gekoppelt mit neuen Formen der praktischen Lehrerfortbildung, und zwar mit einem unterrichtsnahen in-service-training"*[243]. Wird auf diese Weise den Praktikern die Sichtung neuer Entwicklungen und die ggf. daraus folgende Materialentwicklung zumindest teilweise abgenommen, dann scheint es mir durchaus möglich, „ortsnah" aktuelle Curricula zusammenzustellen.

2.2 Zu den fundamentalen Ideen der Informatik

Fundamentale Ideen strukturieren ein Fach sowohl wissenschaftlich wie erkenntnistheoretisch[244]. Sie stellen *„ ‚Schnittstellen' zwischen der (...) informatischen Fachsprache und Fachkultur auf der einen Seite und der außer[informatischen] Kultur unserer Gesellschaft auf der anderen Seite dar"*[245] und tragen so zur kulturellen Kohärenz bei[246]. Sie *„schaffen Beziehungsnetze, prägen [fachlichen] Details eine verbindende Struktur auf und unterstützen so den nichtspezifischen Transfer"*[247]. Fundamentale Ideen sind also außerordentlich hilfreich – aber was sind „fundamentale Ideen"?

2.2.1 Zur Wirkung fundamentaler Ideen

Begriffe wie „fundamental" und „strukturbildend" werden oft in einem etwas diffusen Zusammenhang benutzt. Jeder hat eine vage Vorstellung davon, aber eine exakte Definition fehlt meist – schon bei Bruner[248]. Gehen wir deshalb von der Funktion fundamentaler Ideen aus: In einer Menge ungeordneter Details können Strukturen erzeugt werden, indem man sie anhand unterschiedlicher Ideen ordnet. Ideen induzieren also Ordnung, und verschiedene Ideen erzeugen unterschiedliche Ordnungen. Ideen zeigen Zusammenhänge und verdeutlichen Beziehungen *unter einem bestimmten Aspekt*, sie reduzieren damit die Komplexität durch Weglassen der für diesen Aspekt unwesentlichen Details. In diesem Sinne reduzieren sie die Wirklichkeit auf ihr „Wesen", ihren eigentlichen Sinn[249] und ermöglichen durch diese Reduktion

---

[240] die auch den dringend erforderlichen Erfahrungsaustausch zwischen den oft sehr alleine arbeitenden Unterrichtenden ermöglicht.

[241] Deshalb wird diese Aufgabe oft den neuesten Ausgaben der Computerzeitschriften übertragen, die bei aller fachlichen Sachkenntnis und manchmal anzuerkennendem Bemühen pädagogisch wirklich nicht qualifiziert sind.

[242] [Kla76] S. 121

[243] [Kla76] S. 128

[244] J. Bruner, zitiert nach [Schw95]

[245] [Hey95]

[246] [Tuc96]: *"We believe it is important to implement a similar curriculum for computer science that identifies the "great ideas" in the computing and how these ideas affect the world in which we live."*

[247] [Schw96]

[248] nach [Schw96]

[249] [Bro69]

Lernen. Die Ideen eines Faches verdeutlichen damit die mit diesem Fach verbundenen Sichten auf die Welt[250], sie unterscheiden das Fach von anderen fachlichen Sichten und heben seine speziellen Ansätze und Gewichtungen, aber auch Beschränkungen hervor. Damit machen sie die grundlegenden Fragestellungen und Möglichkeiten eines Fachs auch Nichtfachleuten zugänglich, ermöglichen den Diskurs zwischen Spezialisten und Laien. Sie bilden in diesem Sinne die Grundlage für demokratische Entscheidungsprozesse. Wählen wir als *Anwendungsfälle* fundamentaler Ideen Fragen, die nach Klafki einen Bezug zu Schlüsselproblemen haben, dann ermöglichen wir damit eine ggf. durchaus kontroverse, aber im notwendigen Rahmen fachlich fundierte politische Diskussion eben dieser Probleme in unserer Gesellschaft und tragen so zur rationalen demokratische Willensbildung bei.

Die Ordnungswirkung fachlicher Ideen wird in der Schule ergänzt durch andere ihrer Wirkungen. Zum Unterrichten werden Filter benötigt, die in der Masse möglicher Unterrichtsinhalte erst einmal die wenigen zeitlich realisierbaren Kandidaten für exemplarisches Lernen und Lehren identifizieren helfen[251]. Fundamentale Ideen wirken hier in dreierlei Weise: Einerseits können sie *bei den Unterrichtenden* in ihrer Gesamtheit genau diesen Filter bilden, andererseits sollen sie *bei den Unterrichteten* durch die Bearbeitung solcher Beispiele sowohl selbst wiedererstehen – durch Rekonstruktion der speziellen fachlichen Sichten durch die Schülerinnen und Schüler – als auch hilfreich das Lernen fördern, indem sie es gestatten, eben diese neuen Inhalte in teilweise vorhandene, aber noch wachsende Strukturen einzuordnen und so wiederum zu deren Wachsen beizutragen. Diese unterschiedlichen Rollen machen es schwierig, ein klares Anforderungsprofil für fundamentale Ideen zu entwickeln. Die Unterrichtenden sollten über ein breites Basiswissen verfügen, das von den fachlichen Ideen strukturiert wird, und eben diese Ideen sollten sie beim Unterrichten leiten. Dieses Unterrichten ist aber nur dann wirksam, wenn die meist nur *implizit* im Unterricht z. B. als „Leitlinien“ vorhandenen Ideen in den Köpfen der Unterrichteten – notwendigerweise ohne dieses breite Wissen – neu entstehen. Ich meine deshalb, dass die fundamentalen Ideen des Faches an geeigneten Stellen auch *explizit* thematisiert werden müssen, schon um den Unterrichteten die Zielrichtung des Lernprozesses zu verdeutlichen[252]. Da diese Ideen den speziellen Beitrag eines Faches zur Welterkenntnis bestimmen, ist es gerade in der Sekundarstufe II angebracht, anhand dieses Themas die Stellung des Fachs innerhalb des Kanons der Wissenschaften zu diskutieren – nachdem genügend Kenntnisse erworben worden sind, um diese Diskussion seitens der Schülerinnen und Schüler sinnvoll zu führen[253].

---

[250] Davon kann es durchaus mehrere geben, aber nicht viele.

[251] Denn die Ideen selbst bilden nicht die Inhalte [Hey95]: *Die zentralen Ideen dürfen nicht mit den Grundbegriffen des Fachs verwechselt werden.*

[252] *“Make Thinking Processes Explicit”* Johnson 1992, zitiert nach [Schu01] S. 8

[253] [Schu01] S. 14: *„Der spezifische Beitrag des Informatikunterrichts zur Allgemeinbildung ist, ob nun gewollt oder nicht, gebunden an eine bestimmte Sichtweise auf die typischen Probleme einer technisierten Gesellschaft und auf mögliche Lösungen dafür. Man kann sogar vermuten, dass im Informatikunterricht implizit und unvermeidbar Ansätze für ein bestimmtes Weltbild vermittelt werden.“*

2.2.2 Zu Schwills Kriterien für fundamentale Ideen

Beachten wir die strukturgebende Eigenschaft der fundamentalen Ideen, ihre Stellung beim Lernprozess und vor allem auch die Notwendigkeit, sie durch Rekonstruktion in den Köpfen der Unterrichteten neu entstehen zu lassen, dann können wir uns jetzt der „Fundamentalität" selbst widmen. Nach Andreas Schwill ist *„eine fundamentale Idee der Informatik ein Denk-, Handlungs-, Beschreibungs- oder Erklärungsschema, das vier Kriterien erfüllt: das Horizontalkriterium, das Vertikalkriterium, das Sinnkriterium und das Zeitkriterium."*[254] Für diese Kriterien gilt:

- das Horizontalkriterium: *Die Idee ist in verschiedenen Bereichen der Informatik vielfältig anwendbar oder erkennbar.*
- das Vertikalkriterium: *Die Idee kann auf jedem intellektuellen Niveau aufgezeigt und vermittelt werden.*
- das Zeitkriterium: *Die Idee ist in der historischen Entwicklung der Informatik deutlich wahrnehmbar und bleibt längerfristig relevant.*
- das Sinnkriterium: *Die Idee besitzt eine Verankerung im Alltagsdenken und eine lebensweltliche Bedeutung.*

Als „Masterideen" identifiziert er *Algorithmisierung*, *strukturierte Zerlegung* und *Sprache*, und aus diesen leitet er einen Katalog weiterer fundamentaler Ideen her, der mehr als fünfzig Elemente umfasst. Die schiere Zahl alleine schon scheint mir der „Fundamentalitätseigenschaft" all dieser Elemente zu widersprechen. Nach Wilhelm von *Ockham*[255] *„liegt das Universale nur in der Seele und darum nicht in der Sache"*. Wenn wir Sachverhalte untersuchen, dann muss sich das Universale (hier: das Fundamentale) in diesen zeigen, und dazu darf keine Inflation von fachlichen Inhalten nötig sein, weil in der Menge der Details die Klarheit und Übersichtlichkeit allgemeiner Prinzipien verloren geht. Da der genannte Ideenkatalog aber anhand der so einleuchtenden Schwillschen Kriterien gefunden wurde, müssen diese wohl etwas schärfer gefasst oder zumindest schärfer interpretiert werden, um „die Spreu vom Weizen" zu trennen. Ich werde sie deshalb eingehender untersuchen.

Peter Bender nennt als Kriterien für universelle Ideen der Mathematik die Kriterien *Weite* (logische Allgemeinheit), *Fülle* (vielfältige Verkörperung inner- und außerhalb der Disziplin) und *Sinn* (Verankerung in der Lebenswelt)[256]. Die fundamentalen Ideen der Informatik, insbesondere die Masterideen, hält er für inhaltliche Verkörperungen universeller mathematischer Ideen, die im Gegensatz zu den durch die Mathematiklehrenden nur „*wirksam werdende*n" universellen Ideen allerdings *„selbst die Inhalte eines ernst gemeinten Informatikunterrichts sein [müssten]"* – und damit *„sei die allgemein bildende Schule erheblich überfordert"*[257]. Er macht es sich damit m. E. ziemlich einfach, weil einerseits das „Wirksamwerden" der universellen Ideen wohl nur schwer nachzuprüfen ist, und er andererseits pauschal den doch sehr konstruktiven Schwillschen Ideen die Realisierbarkeit abspricht – und sich somit einer inhaltlichen Diskussion entzieht. Insbesondere übersieht er[258], dass – selbst wenn die Mathematik wirklich die Informatik umfassen würde – dieses für die Unterrichteten unerheblich wäre, solange die prinzipiell vorhandenen Möglichkeiten nicht wirklich realisiert werden,

[254] [Schw93a]
[255] ca. 1285 – ca. 1350
[256] Schreiber, zitiert nach [Ben95] S. 12
[257] [Ben95] S. 13
[258] in der weiteren Argumentation

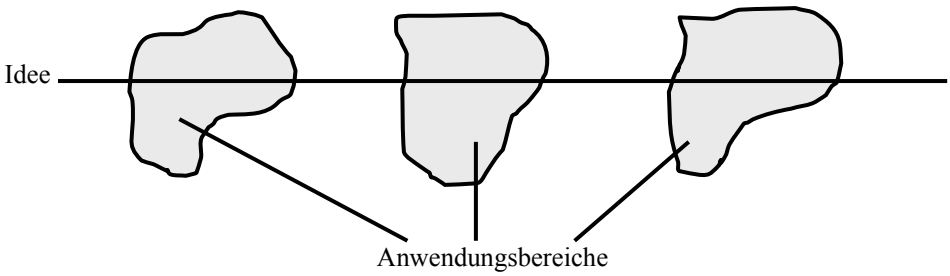
also die spezifischen Stärken des Schulfaches Informatik auch wirklich im Mathematikunterricht zu erfahren sind.[259]

Sigrid Schubert und Wilfried Herget sehen Schwills Kriterien für fundamentale Ideen wesentlich positiver, halten die Diskussion darum aber offensichtlich noch nicht für abgeschlossen, *„da er einen sehr eingeschränkten Bereich der Informatik[260] in die Auswahl einbezieht. Aspekte der technischen und angewandten Informatik[261] fehlen darin.“*[262] Auch nach Herget *„decken die drei Hauptideen noch nicht das ganze Spektrum ab, und insbesondere der dritte Punkt [die Sprache] unterscheidet sich vom Wesen her doch sehr von den ersten beiden.“*[263]

Bedenken wir, dass jedenfalls in dieser Arbeit fundamentale Ideen nur als *ein Aspekt* zur Auswahl von Unterrichtsinhalten beitragen, dann brauchen wir von diesen nicht gleich alles zu fordern, was *insgesamt* für diesen Ausleseprozess von Bedeutung ist. Wir müssen aber fordern, dass sie zur fachlichen Beurteilung beitragen – und dazu dürfen sie nicht allzu abgehoben vom Konkreten sein. Wir müssen auch fordern, dass sie den Lernprozess unterstützen, und dazu dürfen sie nicht in erster Linie auf die Bedürfnisse der Unterrichtenden zugeschnitten sein, sondern müssen den Lernenden dienen. Die verfügen nun weder über ein enzyklopädisches fachliches Wissen, noch sollen sie dieses erwerben. Deshalb tritt die vorhandenes Wissen *ordnende* Eigenschaft der fundamentalen Ideen hinter ihre den Wissenserwerb fördernde *strukturierende* zurück, und weil für diesen zweiten Aspekt Übersichtlichkeit notwendig ist, muss die Anzahl fundamentaler Ideen gering gehalten werden. Die Kriterien zu ihrer Auswahl müssen dieses gewährleisten. Darüber hinaus müssen die Kriterien so gewählt werden, dass auch bei einer beschränkten Menge von fachlichem Wissen im Unterricht die Bedeutung der Ideen deutlich wird[264].

Zum Horizontalkriterium

Eine Idee muss in verschiedenen Bereichen der Informatik vielfältig anwendbar oder erkennbar sein, um als fundamental zu gelten. Das wird nach Schwill wie folgt veranschaulicht[265]:

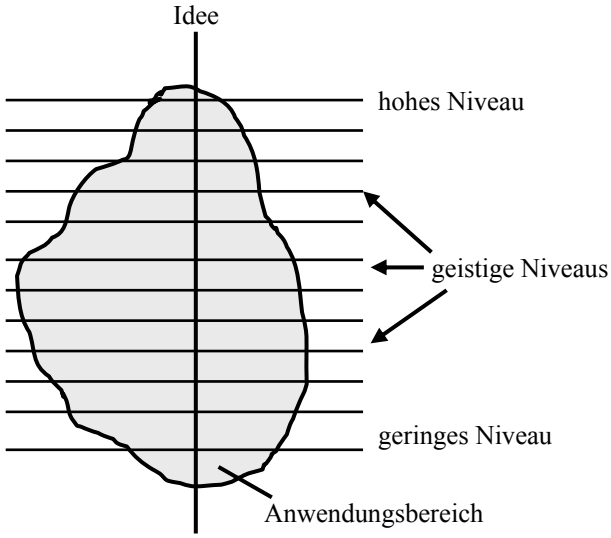


Das Kriterium sortiert reines Spezialwissen aus, ist offensichtlich sinnvoll. Es ist aber eigentlich ein Kriterium für „erfahrene Informatiker“, da es breites Fachwissen voraussetzt, aus dem die gemeinsame Idee durch Abstraktion gewonnen werden kann. Im Unterricht wird den Schülerinnen und Schülern diese Eigenschaft einer Idee nur

[259] An der Diskussion, ob die Informatikinhalte in den Mathematikunterricht integrierbar sind – oder nicht –, möchte ich mich hier nicht beteiligen, weil ich das für ein eher organisatorisches Problem halte. Wenn Elemente des Informatikunterrichts für die Schülerinnen und Schüler von Bedeutung sind, dann ist entscheidend, **dass** sie diesen Unterricht erhalten, und nicht, **unter welchem Etikett** das geschieht: s. [Mod95b]
[260] den der Software-Entwicklung
[261] M. Thomas sieht das ähnlich: [Tho00] S. 17
[262] [Schu99] S. 95
[263] [Her94] S. 36
[264] zumindest werden kann.
[265] Definitionen und Bilder nach [Schw96]

dann deutlich werden, wenn wirklich unterschiedliche Anwendungsbereiche vorkommen, in denen die Idee relevant ist – also mindestens zwei[266]. Und auch dann wird die übergreifende Gültigkeit wohl nur erkannt werden, wenn sie explizit thematisiert wird. Da Anwendungsgebiete, die durch gemeinsame Ideen gekennzeichnet sind, meist auch ähnliche Problemlösungsmethoden erfordern, zumindest ermöglichen, eröffnet die ernsthafte Umsetzung der Folgerungen aus diesem Kriterium einen Unterricht, in dem anhand dieser Beispiele der *spezifische Transfer* erlernter Methoden aktiv geübt werden kann und muss. Daraus kann dann anhand der erworbenen Haltungen und Arbeitsmethoden auch nichtspezifischer Transfer folgen – jedenfalls ist das zu hoffen.

Zum Vertikalkriterium



Eine Idee muss auf jedem intellektuellen Niveau aufgezeigt und vermittelt werden können, um als fundamental zu gelten. Das wird nach Schwill im nebenstehenden Bild veranschaulicht. Das Kriterium ist noch mehr als das vorherige ein Kriterium für Lehrer, denn die Heranwachsenden überblicken nicht ihren Bildungsgang als Ganzes[267], sondern *erfahren* ihre momentane Position in diesem. Das Kriterium ist Grundlage des Spiralcurriculums. Für das Schulfach Informatik ermöglicht es einen Unterrichtsgang, der an unterschiedlichen Stellen sinnvoll abgebrochen werden kann, also sinnvollen Unterricht auch für die Schülerinnen und Schüler, die nur einen Teil der Kursfolge durchlaufen[268]. Voraussetzung dafür ist natürlich, dass der Unterricht wirklich an diesem Kriterium ausgerichtet ist. Innerhalb eines Bildungsgangs – z. B. des gymnasialen – ermöglicht es den Unterrichteten die Erfahrung der zunehmenden Vertiefung und Fundierung ihrer Kenntnisse. Es rechtfertigt einen phänomenologischen ersten Zugang zu einem Problembereich durch enaktiv „handfeste“ und ikonisch „betrachtende“ Erfahrungen zur Vorbereitung späterer Formalisierung.

Zum Zeitkriterium

Eine Idee muss in der historischen Entwicklung der Informatik deutlich wahrnehmbar und längerfristig relevant sein, um als fundamental zu gelten. Nach Schwill können durch Betrachtung der geschichtlichen Entwicklung fundamentale Ideen identifiziert werden[269]. Das Kriterium sichert die Kontinuität des Unterrichts und den Wert der Kenntnisse und Erfahrungen der Unterrichtenden und verhindert fachliche „Moden“. Offensichtlich wird es im Bereich der Informatik kaum beachtet.

---

[266] Weil nur ein kleiner Teil der Unterrichteten später Informatik – oder verwandte Fächer – studieren wird, bliebe den meisten sonst die Bedeutung dieses Kriteriums für immer verschlossen.

[267] schon weil sie ihn noch nicht vollständig durchlaufen haben

[268] was ja derzeit für die Mehrzahl gilt.

[269] [DLP02]: *“How do I apply constructivism in my classroom? You can define or find "essential concepts" in different ways. You might refer to the list of standards your professional group publishes. Or, you can organize your constructivist work by exploring significant historical events or seminal works from multiple perspectives.”*

Zum Sinnkriterium

Eine Idee muss eine Verankerung im Alltagsdenken und eine lebensweltliche Bedeutung haben, um als fundamental zu gelten. Schwill interpretiert das so, dass der Kontext dieser Ideen vortheoretisch sein muss und erst innerhalb des Faches zu einem (Fach-)Begriff präzisiert wird. Mir ist das zu wenig. Da „Sinn" ein sehr weit gefasster Begriff ist, kann dieses Kriterium auch so interpretiert werden, dass die Zahl der fundamentalen Ideen überschaubar bleibt[270]. Wenn wir bedenken, dass fundamentale Ideen die spezifische Sicht eines Faches auf die Welt definieren – also „die Idee" des Fachs – dann kann eine Idee – in meinem Verständnis – nur als „fundamental" gelten, wenn sie wirklich zum Fundament des Fachs gehört, **wenn sie für das Verständnis des Faches notwendig ist**. Erst dann ist auch der Aufwand gerechtfertigt, der zur Rekonstruktion dieser Idee bei den Unterrichteten erforderlich ist. Bilden also die Schülerinnen und Schüler so verstandene Ideen heraus, **dann verstehen sie das Fach** – **und darin liegt der Sinn des Unterrichts**. Ein Satz solcher notwendigen Ideen muss nicht nur orthogonale Elemente enthalten. Im Gegenteil: Weil fundamentale Ideen vortheoretisch sind, nicht wissenschaftlich präzisiert, werden sie zwar unterschiedliche Aspekte des Faches sichtbar machen, das Fach aus unterschiedlicher Sicht beleuchten; sie werden aber auch „Überlappungsbereiche" haben, in denen es möglich ist, diesen Bereich der einen oder der anderen Idee zuzuordnen.

Die Einführung eines „Notwendigkeits-Kriteriums" erübrigt sich, wenn wir das Sinnkriterium entsprechend erweitern:

> *Eine Idee muss eine Verankerung im Alltagsdenken und eine lebensweltliche Bedeutung haben,* ***und sie muss für das Verständnis des Faches notwendig sein****, um als fundamental zu gelten.*

Der Unterricht muss dann so angelegt werden, dass sich diese – wenigen – fundamentalen Ideen bei den Schülerinnen und Schülern bilden können, er muss Kenntnisse und Erfahrungen vermitteln, die anhand dieser Ideen zu ordnen sind, und er muss diese Ideen zu einem geeigneten Zeitpunkt explizit thematisieren, um die spezifischen Möglichkeiten und Beschränkungen der Informatik in Abgrenzung gegen andere Disziplinen erkennbar zu machen.

Akzeptiert man diese Änderung am Sinnkriterium, dann hat das gravierende Folgen: In der Schwillschen Form sind die Kriterien weitgehend objektiv anwendbar, d. h. mit einer Durchmusterung vorhandener Materialien und Daten lässt sich feststellen, ob das Horizontal-, Vertikal- und Zeitkriterium jeweils gelten. Auch Schwills Sinnkriterium wird von ihm – übrigens sehr knapp – nur eingesetzt, um die Eigenschaft einer Idee, vortheoretisch[271] zu sein, objektiv zu überprüfen. Diese Eigenschaft ist in der Tat sehr wichtig, weil es die Kommunikation mit der Welt außerhalb des Faches sichert. In der Objektivität der Kriterien liegt m. E. aber auch ihre Schwäche. Der Verzicht auf Wertung, also auf das Annehmen einer *speziellen* Sicht, billigt *jeder* die „objektiven" Kriterien erfüllenden Idee die Fundamentalitätseigenschaft zu. Weil es unterschiedliche Sichten auf die Welt gibt, kann man aber auch unterschiedliche Sätze von – jeweils wenigen – Ideen finden, die zur Beschreibung einer Sicht notwendig sind. Diese Sichten entsprechen den in 1.3.1 genannten Alternativen, zwischen denen Fachdidaktiker eine Entscheidung treffen müssen. Schwill verzichtet auf diese Entscheidung, und deshalb enthält sein Katalog (fast) *alle* in Frage kommenden Ideen,

---

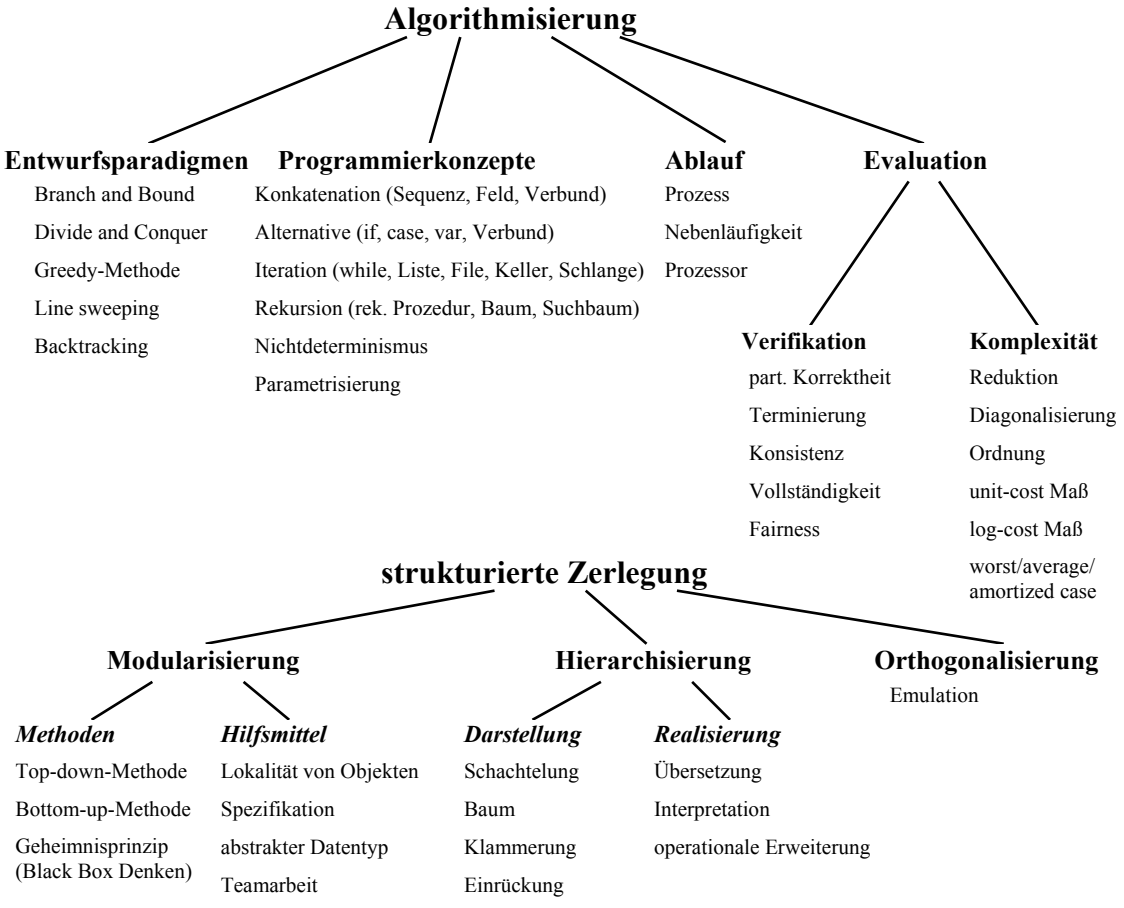
[270] Dadurch wird auch eine „Inflation der Kriterien" vermieden, mit der die „Inflation der Ideen" verhindert werden soll – in Analogie zu „Ockhams Rasiermesser": *Man sollte ohne Notwendigkeit die Zahl der unbewiesenen Annahmen nicht vermehren;* zitiert nach [Con94] S. 417

[271] Das können wir als „an Laien vermittelbar" übersetzen.

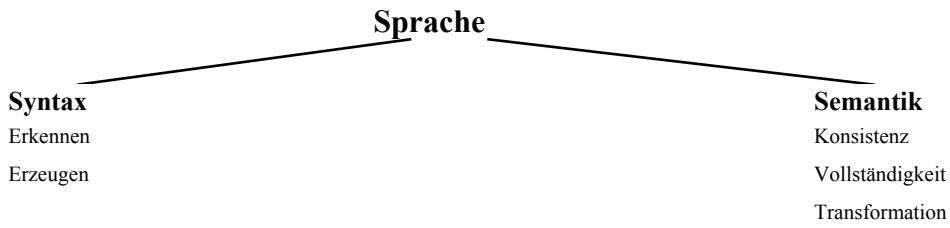
also viel zu viele. Unterrichtende müssen sich zwischen vielen möglichen Zielen für eines (oder wenige) entscheiden, sie müssen eine Position beziehen, die klar ist. Diese Positionierung beinhaltet Einschränkungen, weil andere Positionen damit ausgeschlossen sind, eben nicht eingenommen werden. Daraus folgt eine Einschränkung der Zahl der diese Position beschreibenden Ideen, und **diese sind dann für diese Position fundamental**. Sie beschreiben den *Sinn* des Faches aus dieser Sicht. ***Notwendig* ist also das, was die eingenommene Position zutreffend und knapp beschreibt**, die „Notwendigkeits-Eigenschaft" von Ideen ist ein Ergebnis einer – teilweise subjektiven – Bewertung. Die oben genannte Erweiterung des Sinnkriteriums entspricht m. E. der Bedeutung des Begriffs „Sinn", der immer mit Interpretation und Wertung, meist mit einem Zweck, also mit subjektiver Auslegung und Zielsetzung verbunden ist. Das erweiterte Sinnkriterium befördert Schwills Kriterienkatalog aus dem Bereich (natur-)wissenschaftlicher Objektivität heraus, direkt hinein in die (geistes- und sozial-)wissenschaftliche Hermeneutik, und da sich hier u. a. die Didaktik tummelt, gehört er dort auch hin.

2.2.3 Zu Schwills Masterideen

Schwill nennt als Masterideen die *Algorithmisierung*, die *strukturierte Zerlegung* und die *Sprache*[272]. Offensichtlich gehört der Begriff „Sprache" zu einer anderen Kategorie als die ersten beiden. Auch bei der Aufgliederung in Unterideen zeigen sich Unterschiede. Während die ersten Masterideen zu einer Kaskade nachgeordneter Ideen führen, bleibt der „Sprachbaum" recht mager.



[272] Hubwieser vergleicht diese in seinem informationsorientierten Ansatz mit *Verarbeitung*, *Darstellung* und *Verteilung* von Informationen: [Hub00] S. 81



Schwill bezeichnet alle in diesen drei Bäumen auftauchenden Ideen ausdrücklich als fundamental[273], sie müssen also seinen Kriterien genügen. Ob aber das „log-cost-Maß“, „Line sweeping“ und „partielle Korrektheit“ wirklich eine Verankerung im Alltagsdenken und eine lebensweltliche Bedeutung haben, mag dahingestellt sein[274].

Untersuchen wir die drei Ideenbäume etwas genauer:

Algorithmisierung und strukturierte Zerlegung führen direkt zu sehr handfesten Unterideen, sie sind konkret anwendbar, am informatischen Handeln orientiert. Sie sind aber nicht klar gegen einander abgegrenzt[275], weil einerseits strukturierte Zerlegung sicher als Teil des Algorithmisierungsprozesses aufgefasst werden kann, und anderseits z. B. die algorithmischen Grundstrukturen Anfang oder Ende[276] des Zerlegungsprozesses definieren. Innerhalb der Bäume können wir die einzelnen Ebenen als Präzisierungen der vorangegangenen auffassen, als zunehmende fachliche Durchdringung der übergeordneten Ideen. Ebenso wie *Weite*, *Fülle* und *Sinn* als Kriterien nicht spezifisch für die Auswahl mathematischer Ideen sind, sondern nicht nur in Hinsicht auf die Mathematik noch einer erheblichen Präzisierung bedürfen, können *Algorithmisierung* und *strukturierte Zerlegung* als fundamentale Ideen nicht als spezifisch informatisch angesehen werden. Jede Naturwissenschaft und natürlich auch und gerade die Mathematik nimmt entsprechende Ideen für sich zu Recht in Anspruch. Schwills „Unterideen“ präzisieren nun die Masterideen in Hinsicht auf die Informatik, stellen das spezifisch Informatische an ihnen heraus. Fundamentalen Ideen scheint es eigen zu sein, dass sie nicht besonders fachspezifisch formuliert werden können, besser: dass die Fundamentalitätseigenschaft sich in dem Maße verliert, wie sie fachlich präzisiert werden[277], und dass sich das Fachliche verliert, wenn die Fundamentalität zunimmt[278]. Schwills Ideenbäume erläutern in diesem Sinne, wie die Masterideen zu verstehen sind, wenn man sie auf die Informatik anwendet. Wenden wir sie auf ein anderes Fach an, dann erhalten wir andere Ideenbäume, und die werden wir auch erhalten, wenn wir das gleiche Fach „anders verstehen“, also eine andere Sicht einnehmen (s. o.).

Weil Unterideen die jeweiligen übergeordneten Ideen fachlich präzisieren, gewinnen sie ihre Bedeutung u. a. auch durch diese Funktion. So aufgefasst muss eine ziemlich spezielle Idee wie „partielle Korrektheit“ nicht mehr einen allzu engen Kontakt zur Erfahrungswelt haben, weil sie diesen Aspekt teilweise von den Ideen der „Verifikation“ und „Evaluation“ erbt. Sie ist also als Unteridee weniger fundamental als z. B. die Masterideen und genügt deshalb auch weniger den Kriterien für Fundamentalität. Sie ist aber wesentlich präziser in Hinsicht auf die informatische Ausprägung und deshalb notwendig zur Erläuterung des Gemeinten.

---

[273] [Schw96] Teil C letzte Seite

[274] Zumindest erscheint mir das strittig und einer Erklärung bedürftig, und somit ist es mit der „objektiven Anwendbarkeit“ dieses Kriteriums auch in der bisherigen Form nicht weit her.

[275] und brauchen das auch nicht zu sein (s. o.)

[276] je nach „Arbeitsrichtung“

[277] siehe Schwills *Unterideen*

[278] siehe Benders *universelle Ideen*

Betrachten wir einmal zum Vergleich die *Algorithmisierung* und die *strukturierte Zerlegung*, angewandt auf die Physik: Ebenso wie in der Informatik können wir die betrachteten Systeme modellieren, die erkannten Gesetzmäßigkeiten formal beschreiben und so z. B. Voraussagen über die zeitliche Entwicklung des Systems machen – natürlich jeweils mit den fachspezifischen Methoden und Werkzeugen. Wir können die Systeme zerlegen, unter verschiedenen Aspekten vereinfachen[279], unterschiedliche Module und Hierarchien herausarbeiten, und eine ganze Weile lang kommen wir dann auch in der Physik zu immer einfacheren Teilsystemen, die komplexes Verhalten im Rahmen des Modells „erklären" – z. B. von der Thermodynamik über die elementare Atomistik zur klassischen Mechanik[280]. Bei diesen Prozessen werden Ideen der Physik deutlich, die beschreiben, wie beobachtete Systeme „strukturiert zerlegt" werden können, um zu fachlich beherrschbaren Teilproblemen zu kommen – aber ganz anders als in der Informatik. Bei der Fortsetzung des Prozesses in der Physik erreichen wir dann jedoch eine neue Ebene der Wirklichkeitsbeschreibung, die alles andere als eine weitere Vereinfachung darstellt – z. B. in der Quantenmechanik oder in der nichtlinearen Dynamik[281]. Hier zeigen sich entscheidende Unterschiede: Während die Informatik – solange sie nicht auf ein reales System angewandt wird und damit ggf. dessen Kontext übernimmt – als Strukturwissenschaft betrachtet sich mit abstrakten Konstrukten beschäftigt, deren Komplexität beim Zerlegungsprozess abnehmen *muss*, wenn beherrschbare Systeme das Ziel der Entwicklung sind, misst sich die Physik als Naturwissenschaft an der – angenommenen – Realität, die sich in immer neuen Wahrnehmungsebenen auf immer neue Weise offenbart. Im üblichen Bild des Modellbildungsprozesses als Kreislauf, in dem wiederholt Modelle gebildet, auf mathematische und/oder informatische Konstrukte abgebildet, getestet und neu angepasst werden[282], konzentriert sich eine so aufgefasste „physikalische" strukturierte Zerlegung eher auf die Konstruktion und die Veränderung von geeigneten Modellen, also ggf. auf eine *Folge von Modellen* und deren jeweilige Gültigkeitsbereiche, während die Informatik im engeren Sinne diese Idee bei der Realisierung *des Modells* wirksam werden lässt. Diese Unterschiede werden dann in den tiefer liegenden Ebenen der entsprechenden Ideenbäume zu Tage treten.

Fundamentale Ideen erfordern also eine gewisse Allgemeinheit, die einer fachlichen Erläuterung bedarf, die besagt, was genau *aus der Sicht des Faches* unter diesen Ideen zu verstehen ist. Nun wird ein Fach an allgemein bildenden Schulen nie in seiner vollen Breite und immer mit unterschiedlicher Vertiefung unterrichtet. Es ist also zu fragen, welche der Äste und welche Verzweigungstiefe für eine hinreichend genaue Rekonstruktion der angestrebten informatorischen Weltsicht bei den Lernenden notwendig sind. Es erscheint mir unstrittig, dass Algorithmisierung und strukturierte Zerlegung zu den fundamentalen Ideen der Informatik gehören, es ist aber nach den oben angestellten Überlegungen offensichtlich, dass diese Masterideen noch zu allgemein sind, um das Fach hinreichend zu beschreiben. Notwendig sind sicherlich Vorstellungen von den *Ergebnissen der Algorithmisierung*[283] und den *Wegen zu diesen*. Die Lernenden sollten z. B. wissen, dass durch *Reihung*, *Alternative* und *Iteration* Algorithmen so formuliert werden können, dass Computer sie abarbeiten können. Zur Formulierung können aber auch *Reihung*, *Alternative* und *Rekursion* herangezogen werden. Es ist sicher nicht unbedingt notwendig, beide Möglichkeiten auf-

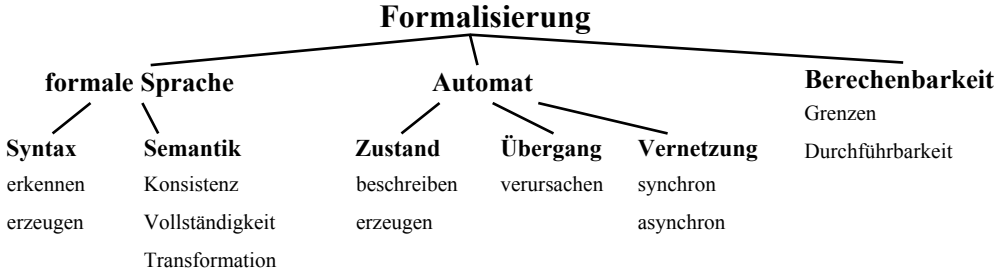
[279] z. B. durch die Wahl der Observablen
[280] oder umgekehrt
[281] z. B. in [Con94], [Man91], [Schr91], [Cra88], [Arg94], [Pri81], [Gle90]
[282] z. B. in [Hen98] S. 9
[283] also z. B. den „Programmen"

zuführen, und keine der beiden ist qualitativ besser[284]; beide sind aber möglich und stellen so jeweils eine Präzisierung der fundamentalen Ideen in diesem Bereich dar – je nach Ziel des Unterrichts. Entsprechend ist es notwendig, Entwurfsmethoden zu konkretisieren – durch mindestes ein Verfahren. Welches davon gewählt wird, sollte den Lehrenden überlassen bleiben. Die Evaluation von Algorithmen gehört in den Unterricht. Ob dieses mit formalen Methoden – und welchen – und/oder Tests – und welchen – erfolgt, hängt von vielen Faktoren ab[285].

Sprache als Masteridee gehört zu einer anderen Kategorie als die ersten beiden. Der Begriff passt schlecht zu den anderen, eher handlungsorientierten, er wirkt künstlich angefügt. Die „Sprachidee" bedarf sicherlich einer Interpretation. Schwill liefert die durch Beispiele, die eine Gemeinsamkeit haben: Es handelt sich (fast) ausschließlich um formale Sprachen und Verfahren, die dann wiederum sehr konkret sind. Ich denke, dass die *Sprach*-Idee durch die Masteridee ***Formalisierung***[286] ersetzt werden sollte. Diese ist nicht auf Sprachen beschränkt, trifft aber den m. E. von Schwill gemeinten Bereich besser, umfasst ihn und passt auch besser zu den beiden anderen. Verstehen wir unter *Formalisierung* die Idee, bekannte oder neu gewonnene Verfahren so zu beschreiben, dass sie nach festen Regeln ohne weiteren menschlichen – also denkenden – Eingriff ablaufen können, dann beschreibt die Idee das Bemühen, von den Verständnis voraussetzenden Tätigkeiten die von einem formalen System durchführbaren Bereiche abzutrennen, um sich dann von menschlicher Seite auf den „interessanten, nicht formalisierbaren Rest" zu konzentrieren – wenn es den gibt[287]. Formalisierung umfasst damit die ***Automatisierung*** und den Begriff der ***Maschine***. Damit genügt die Idee offensichtlich dem Horizontalkriterium, denn diese Begriffe tauchen praktisch in allen Bereichen der Informatik auf. Beschreiben wir Maschinen durch ***Zustände*** und deren Übergänge, dann haben wir eine Idee, die auf jeder Komplexitätsebene anwendbar ist, vom Blumenautomaten und einfachen Addierer über OOP-Systeme bis zu Spezialthemen der theoretischen Informatik – also ist das Vertikalkriterium erfüllt. Das Zeitkriterium gilt ebenfalls, denn die Untersuchung der Möglichkeiten und Grenzen formaler Systeme gehört mit den Arbeiten Turings, Gödels, Churchs und anderer einerseits zu den Wurzeln der Informatik, während andererseits die theoretische Durchdringung komplexer Systeme aktuelle Forschungsthemen bietet. Bleibt das Sinnkriterium in der von mir gewählten Form. Die automatische Bearbeitung von Aufgaben, die ehemals menschliches Verständnis voraussetzten, gehört nun in der Tat zur lebensweltlichen Bedeutung. Sie findet sich im Alltagsdenken, beherrscht es schon teilweise und weiter zunehmend, sei es bei der Arbeit, in der Freizeit[288] oder als „Betroffener". Ist nun Formalisierung für das Verständnis des Faches *notwendig*? Ich meine ja, denn die Überführung immer weiterer Bereiche in eine Form, die formalen Methoden zugänglich ist, beschreibt zentral das, was Informatik für die Gesellschaft bedeutet. Die automatische Bearbeitung erleichtert vieles, bietet neue Möglichkeiten und schafft Raum für neue Aufgaben. Sie nimmt aber auch Einflussmöglichkeiten, erfordert Vereinheitlichung und erschwert Individualität. Komplexe formale Systeme können den Eindruck von Verständnis erzeugen, sie spiegeln Personalität und ersetzen so – z. B. bei den Computerspielen

[284] aber natürlich ist – je nach Anwendungsfall – oft eine vorzuziehen, weil „eleganter".
[285] z. B. von der Kursart: Grund- oder Leistungskurs
[286] aufgefasst als „nach Regeln ohne Verständnis bearbeitbar"
[287] Als Beispiel aus den Naturwissenschaften kann die Entstehung von komplexen Organismen dienen, die einerseits – ehemals ausschließlich – durch transzendente Überlegungen erklärt werden können, jetzt aber auch zumindest teilweise durch Prozesse wie Selbstorganisation und Evolution.
[288] Die Bedeutung von Computern als Spielpartner bei Jugendlichen – und zunehmend auch bei Erwachsenen – ist erschreckend.

oder in Lehrsystemen – menschliche Kommunikation. Einsicht in die Möglichkeiten formaler Systeme kann die Gefahren der Entfremdung im sozialen und mentalen Bereich zeigen, die eine Entsprechung zur Entfremdung von der Natur in der modernen Industriegesellschaft bildet, sie kann auch die Notwendigkeit von Grenzen für den Einsatz solcher Systeme verdeutlichen, die ähnlich wie in der nichtlinearen Dynamik bei genügend großer Komplexität kaum noch beherrschbar sind.



Die Idee der Formalisierung ist wiederum nicht orthogonal zu den anderen beiden Masterideen. Sie beschreibt aber einen anderen wesentlichen Aspekt und erweitert das Spektrum stark in Richtung auf die theoretische und technische Informatik. Im Bereich der Schule führt sie zu dauerhaften Inhalten und betont die Verantwortung bei der Behandlung von Schlüsselproblemen, die sich direkt aus den technischen oder erkenntnistheoretischen Anwendungen der formalen Konzepte ergeben.

Da die Ideenbäume nicht mit den Inhalten verwechselt werden dürfen, anhand derer die Ideen zu verdeutlichen sind, muss auch hier ausdrücklich darauf hingewiesen werden, dass der Baum keine Übersicht über die Inhalte der theoretischen Informatik an Schulen liefern soll. Ähnlich wie die anderen beiden Bäume lassen sich aber auch hier den angegebenen Ideen leicht Inhalte zuordnen. Die Ideenbäume sollten deshalb – in der etwas modifizierten Form – geeignet sein, Unterrichtsinhalte zu klassifizieren und damit geeignete Unterrichtsgänge auszuwählen. Schwills Ideenbäume zur Algorithmisierung und strukturierten Zerlegung sowie meiner zur Formalisierung umfassen also m. E. unterschiedliche Sätze fundamentaler Ideen, die von den Unterrichtenden jeweils geeignet zu wählen sind. Es wäre sicherlich ein Kunstfehler, sich bei dieser Auswahl auf einen Ast oder einen Baum zu beschränken, also z. B. einen reinen Programmierkurs (im Sinne von „Codierkurs“) einzurichten. Die Bäume können als Hilfen angesehen werden, die den Lehrenden unterschiedliche Möglichkeiten aufzeigen, je nach Vertiefungsgrad übergeordnete Ideen unterschiedlich detailliert zu behandeln. Im Groben kann gefordert werden, keinen der Äste im Unterricht zu vergessen; in welchem Ausmaß dieses jeweils geschieht, muss den Unterrichtenden – natürlich in Übereinstimmung mit den Rahmenrichtlinien – überlassen bleiben.

Der Vorwurf der „beengten Sicht“, der z. B. von Schubert und Herget gegen Schwills Masterideen erhoben wird, ist, wenn man seine Beispiele betrachtet, wohl nicht zutreffend. Informatikanwendungen sind notwendig mit den Masterideen *Algorithmisierung* und *strukturierte Zerlegung* verbunden. Schwills *Sprach*-Idee z. B. umfasst durchaus die theoretische und technische Informatik, sie legt dort aber keinen Schwerpunkt. Ersetzen wir sie durch die *Formalisierungs*-Idee, dann liefern formale Sprachen, zugeordnete Automaten, Anwendungen bei Schaltwerken reichlich gut erprobte schulische Standardthemen aus diesem Bereich. Netzwerke unterschiedlicher Klassen, gekoppelte Automaten und deren Anwendungen liefern Zugang zu aktuellen und auch für die Unterrichteten spannenden Themen.

2.3 Zu den vorhandenen Didaktikansätzen

In der Informatikdidaktik findet man eine Reihe von unterschiedlichen Konzepten, die sich in der „Orientierung“ der jeweiligen Ansätze zeigen. Untersuchungen dazu finden sich z. B. bei Wilfried Herget[289], der herausstellt, dass sich bei allen vordergründigen Unterschieden die Ziele relativ wenig von einander unterscheiden[290], auf hoher Ebene sogar weitgehend mit denen des Mathematikunterrichts übereinstimmen. Die unterschiedlichen Orientierungen der Informatikdidaktik ergaben sich in der Vergangenheit teilweise aus den vorhandenen technischen Möglichkeiten der Schule[291], teilweise auch aus dem jeweiligen Kenntnisstand der Lehrenden. Die Folge aus Hardware-, Algorithmen-, Anwendungs-, Benutzer-, Informations- und anderen Orientierungen spiegelt so auch die technische Entwicklung der Geräte und teilweise die fachliche Entwicklung der Lehrenden wider, und das beklagte Verharren der Schule beim algorithmenorientierten Ansatz ist eine logische Folge der fehlenden fachspezifischen Aus- und Weiterbildung der Lehrkräfte. Die dann ebenfalls logische Folge der sinkenden Teilnehmerzahlen an Informatikkursen ergibt sich aus den wachsenden Unterschieden zwischen „erfahrener“ Computerwelt und der Schulwirklichkeit. (Dabei muss deutlich unterschieden werden zwischen einem „Verharren aus Einsicht“ – also einer Betonung der auch aktuell gültigen und hilfreichen Grundlagen des Faches – und einem „Verharren aus Mangel und/oder Resignation“.) Wesentlichen Einfluss auf die Diskussion hat m. E. auch die jeweilige bildungspolitische Situation genommen. So spielte in den Anfangsjahren des Faches die Abgrenzung gegen die Mathematik eine große Rolle – was zu einer grotesken „Verteufelung“ der so reizvollen mathematischen Inhalte führte –, während später mit dem Aufkommen der Standardsoftware teilweise eine Abgrenzung gegen die informationstechnische Grundbildung, teilweise ein Aufgehen in dieser für notwendig befunden wurde – mit der Folge der „Verteufelung“ des Programmierens. Mit dem z. B. von der GI festgestellten Scheitern der ITG[292] werden aktuell wieder originär informatische Gesichtspunkte stärker gewichtet.

Der Ablauf der Diskussion ist für das Verständnis der aktuellen Situation wichtig. Der Dauerstreit z. B. um die Stellung des Programmierens in der Kursfolge kann nur verstanden werden, wenn man berücksichtigt, dass offensichtlich nicht geklärt ist, aus welchem Grund in Schulen überhaupt programmiert werden soll – und das hängt wiederum mit fehlender Klarheit über die Rolle des Informatikunterrichts im Spektrum der Schulfächer zusammen. Die Ziele eines Schulfaches werden nun u. a. von den unterschiedlichen gesellschaftlichen Gruppen beeinflusst, die meist aufgrund der aktuellen Situation und auch ihrer sehr spezifischen Sichtweise urteilen und nicht auf der Basis langfristiger erziehungswissenschaftlicher Überlegungen. Man kann den Wechsel bei den Ansätzen der Informatikdidaktik deshalb zwar überspitzt, aber doch recht gut mit dem jeweils aktuellen Bedarf auf dem IT-Arbeitsmarkt und der Ausstattung der Schulen erklären. Schon früh wurden Computer „irgendwie“ für wichtig gehalten, und weil Schulen keine hatten, sollten sie wenigstens „irgendwas“ in dieser

---

[289] [Her94]

[290] Wie richtig er damit liegt, zeigt sich, wenn man die in den Schulbüchern und Veröffentlichungen jeweils bearbeiteten Probleme und „Aufgaben“ vergleicht. Da tut sich nämlich kaum etwas: Vom Zeichnen von Funktionsgraphen über die Sortierung von Telefonnummern bis zu den allseits beliebten Bundesjugendspielen und der Organisation der Schulbibliothek (den „Projekten“) findet sich alles mit verblüffender Konstanz wieder – nur jeweils mit einem neuen Tool bearbeitet.

[291] So ist „Hardwareorientierung“ zwanglos zu übersetzen als *Fehlen von Schulcomputern*, aber *Vorhandensein von digitalen Bausteinsätzen*.

[292] [GI01]

Richtung unternehmen, also digitale Schaltungen entwerfen: Es folgt der „hardwareorientierte Ansatz“. Mit dem Aufkommen der Schulcomputer wurde der direkte Umgang mit diesen Systemen möglich, und Anfang der 80-er Jahre war Programmieren die einzige Möglichkeit, die Geräte sinnvoll zu benutzen: Es folgt der „algorithmenorientierte Ansatz“, anfangs, weil nur wenige Geräte bezahlbar waren, als Fach für „besondere“ (wenige) Schülerinnen und Schüler, also als Oberstufenkurs. Standardsoftwarepakete sind wesentlich einfacher zu benutzen als Programmentwicklungssysteme, lösen viele Aufgaben und fanden so schnell Verbreitung: Es folgt der „anwendungsorientierte Ansatz“, verbunden durch fallende Computerpreise mit einer Verbreiterung der Basis in Richtung „ITG für alle (kommenden Arbeitskräfte)“ und in Richtung Mittelstufe. Den derzeitigen „informationsorientierten Ansatz“ kann man u. a. mit dem Aufkommen des Internets, seiner prognostizierten wirtschaftlichen und gesellschaftlichen Bedeutung und der damit aufziehenden globalen Informationsgesellschaft begründen. Wenn diese Beschreibung auch eine stark verkürzte Sicht darstellt[293], so ist sie m. E. doch nicht ganz falsch, und wenn das so ist, dann zeigt sich, dass außerhalb des Bildungsbereichs ein originäres Interesse am Bildungswert bestimmter Informatikinhalte oder sogar am Fach selbst gar nicht besteht, sondern nur ein diffuses Interesse bei Eltern und Arbeitgebern festzustellen ist, den Umgang mit Computern „in brauchbarer Form“ irgendwie sicherzustellen[294]. Dazu meinte man zuerst – als seinerzeit einzige Alternative – das Schulfach Informatik zu brauchen, danach wurde dieses mit der ITG anders gesehen, und nun scheint wieder die Informatik an Ansehen zu gewinnen.

Dass die Informatikdidaktik wenigstens teilweise wirklich durch diese Trends beeinflusst wurde, zeigt sich auch in der aktuellen Diskussion: Es ist m. E. ungeklärt, ob Informatik nun auch für eine Breiten-Grundbildung sorgen soll[295] (im Sinne der ITG) oder nur ein Spezialistenfach für wenige ist. Die damit verbundenen Entscheidungen zwischen

Pflichtfach ←→ Wahlfach
Oberflächenwissen ←→ vertieftem Wissen
Anwenderkenntnissen ←→ Fachkenntnissen

sind nicht gefallen, und so findet man in der didaktischen Literatur erhebliche Widersprüche:

- Einerseits wird eine Sammlung außerordentlich anspruchsvoller Ziele formuliert („*Auswirkungen reflektieren und beurteilen*[296]“, „*Fähigkeiten zur Auswahl und der problembezogenen selbstständigen Nutzung geeigneter Systeme sowie deren Anpassung*“, „*Chancen und Risiken der Anwendung von Informatiksystemen erkennen und zum verantwortungsvollen Einsatz von Informatiksystemen bereit sein*“, „*Bewertung von Informatiksystemen*“[297]), die erhebliche Erfahrungen und Fachkenntnisse bei den Unterrichteten voraussetzt,

  andererseits lassen es die vorgelegten Beispiele und der angesetzte Zeitbedarf zweifelhaft erscheinen, dass diese Ziele erreicht werden können[298].

[293] und ein wenig lästerlich ist
[294] Die Forderung, das Fach nach den aktuellen Gegebenheiten auf dem Arbeitsmarkt und den momentanen Forderungen gesellschaftlicher Gruppen auszurichten, ist schon deshalb unsinnig, weil sich deren Empfehlungen sehr schnell ändern, teilweise ins genaue Gegenteil.
[295] Dass die ITG „gescheitert“ ist, kann ja wohl nicht ernsthaft als Lösung dieser Frage angesehen werden.
[296] „Aufgaben und Ziele der Grundbildung“, Nordrhein-Westfalen 1990
[297] [Fri95]
[298] z. B. in [Bau96a] S. 342, 361

- Einerseits wird aktive Schülerarbeit und Projektunterricht[299] als grundlegend für das Fach herausgestellt[300],

  andererseits wird die für Projektunterricht wesentliche Produkt- und Ergebnisorientierung, die im Informatikunterricht weitgehend durch Programmierung realisiert wird, erschwert oder verhindert, indem ein in den Unterricht integrierter Programmierkurs als schädlich oder überflüssig hingestellt wird, zumindest aber nicht genügend Zeit bekommt, obwohl anderseits verschiedene Programmierparadigmen kennen gelernt und beurteilt werden sollen.

- Einerseits wird der schnelle Wechsel bei den Informatikinhalten beklagt und die Besinnung auf Grundlagen gefordert[301],

  andererseits wird das Fach weitgehend durch die starke Anwendungsorientierung begründet.

- Einerseits wird ein Pflichtfach Informatik gefordert, „*weil die notwendige Einsatzbereitschaft der Schülerinnen und Schüler nur in einem Pflichtfach erwartet werden kann*“[302] (m. E. eine merkwürdige Auffassung von Schule!),

  andererseits werden die motivierenden Momente des Faches, die zu eben dieser Einsatzbereitschaft führen, zugunsten formaler Methoden (bei Hubwieser alleine in Klasse 9 praktisch alle in der Informatik gebräuchlichen Diagrammtypen gleichzeitig[303]) herabgestuft[304]. Eine wenig überraschende Konsequenz ist, dass *„die am Versuch beteiligten Schüler* (...) *beängstigend wenig Interesse erkennen* [ließen]“[305] – und dann braucht man eben ein Pflichtfach.

Ich halte eine fachliche „Orientierung“ des Informatikunterrichts – gleich welcher Art – für kontraproduktiv[306]. Obwohl eine einheitliche Ausrichtung intellektuell reizvoll ist, werden die pädagogischen Möglichkeiten des Faches unnötig beschränkt und die Fachinhalte erhalten eine Gewichtung, die diesen nicht zukommt[307]. Ein Fach sollte den durch allgemeindidaktische Überlegungen gesteckten Rahmen vollständig ausfüllen und nicht versuchen, ihn aus fachlichen Überlegungen weiter einzuschränken[308]. Vor allem aber sollte es nicht Ausrichtungen vornehmen, die mit lerntheoretischen Ergebnissen kollidieren. So widerspricht die Orientierung an *einem* fachlichen Oberbegriff, z. B. dem der Information, klar dem konstruktivistisch gedachten Aufbau von Denkstrukturen, die von *vielfältigem* bestehendem Wissen ausgehen, das der Erfahrungswelt der Lernenden entlehnt ist. Natürlich kann eine informationsorientierte Sicht das *Ziel* des Unterrichts sein; sie darf aber nicht den *Unterrichtsgang* prä-

---

[299] [Hum00] S. 11
[300] praktisch in allen didaktischen Arbeiten
[301] [Hum00] S. 11
[302] [Hub00] S. 70 und [GI01] S. VI
[303] in [Hub97]: ER-, Objekt-, Zustandsübergangsdiagramme, Datenflussmodelle, Aktionsgraphen, ...
[304] z. B. in [Bur94] zu Lehmanns Arbeiten: „*Die Veröffentlichung von Musterprojekten [...] hatte eher abschreckenden Charakter: Die Formalisierung durch Formulare u. a. konnte auch die letzte Kreativität der Schüler vertreiben.*“
[305] [Leh92] S. 30
[306] Unterricht sollte zielgruppen- hier: schülerorientiert sein.
[307] [Kla85] S. 35: „*Fachdidaktische Entscheidungen können folglich nicht aus den einzelnen Fachwissenschaften abgeleitet werden, da Schulfächer* [...] *und Fachwissenschaften nicht notwendig kongruent sind.* [...] *Die Fachwissenschaften entwickeln als solche keine hineichenden Auswahlkriterien, wiewohl didaktische Entscheidungen selbstverständlich nicht ohne Bezug auf die jeweilige Bezugswissenschaft gefällt werden können.*“
[308] [His91] S. 13: *„Eine didaktische Standortbestimmung kann – für welches Fach auch immer – weder allein aus Sicht dieses Unterrichtsfachs noch aus der Sicht einer ggf. existierenden Mutterwissenschaft vorgenommen werden. Vielmehr ist es erforderlich, auch übergreifende Standorte zu erklimmen, um damit den Bildungsauftrag der Schule – für die Gesellschaft und das Individuum – in den Blich nehmen zu können.“*

gen und schon gar nicht als *Startpunkt* gewählt werden[309]. Weiterhin bezweifele ich in vielen Fällen, dass die übergeordnete Sichtweise, die den Entwicklern eines Didaktikansatzes zur Verfügung steht, von den Lernenden aus den Unterrichtsinhalten wieder erschließbar, also rekonstruierbar ist. Insofern wäre dann die Orientierung sinnlos, weil sie die Adressaten des Unterrichts nicht erreicht.

Weil die existierenden Didaktikansätze außerordentlich unterschiedlich sind, lässt sich ein einheitliches Schema zu ihrer Bewertung in Rahmen dieser Arbeit kaum finden. Sie anhand der hier entwickelten Maßstäbe zu messen, wäre unfair, da sie andere Zielsetzungen haben. Ich beschränke mich deshalb auf eine punktuelle Kritik, die von den in den jeweiligen Arbeiten selbst zugrunde gelegten Kriterien ausgeht.

Am Beispiel von Peter Hubwiesers Didaktik[310] ist der ungute Einfluss der (hier: Informations-)Orientierung nachzuvollziehen: Nach erfreulichen Aussagen zu Motivation und Kreativitätsförderung[311] und einer einleuchtenden Begründung für die Bedeutung des Informationsbegriffs im Informatikunterricht, der Entwicklung eines Grundschemas der Informationsverarbeitung[312] und einer Darlegung der Wichtigkeit langfristig relevanter Inhalte erfolgt eine Einschränkung auf die zuletzt genannten Punkte, die m. E. unzulässig ist, weil sie *alleine fachimmanent* begründet ist: Hubwieser rechnet Programmiersprachenkenntnisse nicht zum Kanon allgemein bildender Inhalte – obwohl er sie für einen *„unverzichtbaren Baustein zum Verständnis elektronischer Informations- und Kommunikationssysteme“*[313] hält – und vermeidet die Erstellung konkreter Programme, wo immer es geht. Er folgert: *„Es liegt auf der Hand, dass Software-Entwicklungsmethoden für den Schulunterricht umso wertvoller sind, je allgemeiner, sprach- und plattformunabhängiger sie sind.“*[314] Er verzichtet weitgehend auf die Implementierung der entwickelten Methoden und so m. E. auf den aus allgemeindidaktischer Sicht entscheidenden Grund für ein Schulfach Informatik[315] (s. 2.4.1). Um die Programmierung zu umgehen und planendes Handeln zu unterstützen, benutzt er diverse Arten von Diagrammen bei der Modellierung und Darstellung von Informationen. Er bleibt aber auf eine Begründung schuldig, weshalb denn ER-Diagramme, Datenflussmodelle oder Transitionsgraphen allgemein bildend sein sollen[316]. Dass die Beschränkung auf den Umgang mit diesen Graphen in der beschriebenen 9. Klassenstufe motivations- und kreativitätsfördernd sein soll, widerspricht all meiner Berufserfahrung. Die als Kompromiss angebotene Pascal-Implementierung von Automatenmodellen in einem völlig starren Schema, aufgegliedert in Etappen, die nach einzelnen Kontrollstrukturen geordnet sind, erschwert durch ihre Einseitigkeit die Nutzung des Programmentwicklungssystems als Werkzeug zur Realisierung eigener Ideen. Die Informationsorientierung weitgehend ohne Implementation der Modelle führt bei Hubwieser zu vorbereitendem Lernen – das schon in anderen Fächern nicht funktioniert und die Motivation töten kann. Die z. B. in der Klasse 6 eingeführten, an Java angelehnten formalen Objektbeschreibungen bleiben zu diesem Zeitpunkt folgenlos und werden wohl vergessen sein, wenn laut Lehrplan nach zwei Schuljahren weiterfüh-

---

[309] Jedenfalls solange nicht, wie Informationsorientierung nicht zur Erfahrungswelt der Lernenden gehört.
[310] [Hub00]
[311] [Hub00] S. 17: *„Die Kreativität der Schüler ist jedenfalls Voraussetzung für die Neukonstruktion von Wissen. Deshalb gehört ihre Förderung zu den wichtigsten Zielen des Unterrichts.“*
[312] Darstellung, Verarbeitung, Transport und Interpretation von Informationen
[313] [Hub00] S. 169
[314] [Hub97] S. 44
[315] Das wirklich Neue am Fach besteht ja darin, dass die Ideen der Unterrichteten standardmäßig realisiert werden können, nicht nur „besprochen“ werden. Wird hierauf verzichtet, dann wird das Spektrum der Schule nicht entscheidend erweitert.
[316] es sei denn, das „Nicht-Programmieren“ wäre ein Wert für sich

rende Einheiten auftauchen. Ich hielte es für besser, wenn Unterrichtsinhalte den Unterrichteten *zum aktuellen Zeitpunkt* hilfreich und sinnvoll erscheinen[317] und ihren Sinn nicht nur aus der Fortsetzung zu einem späteren Zeitpunkt erhalten. Hubwieser zahlt m. E. für seinen informationszentrierten Ansatz einen hohen Preis: Seine Beispiele passen nicht mehr so recht zu seinen sehr ansprechenden, offensichtlich aus Erfahrung gewonnenen Praxistipps vom Anfang seines Buches[318], und er läuft m. E. Gefahr, die Schülerinnen und Schüler zu verlieren[319], denn sonst wäre seine Forderung nach Pflichtunterricht, weil „*die Schüler nur in einem solchen Fach zu derartigen Anstrengungen bereit sind*“, kaum zu erklären[320]. Die informationszentrierte Sicht auf die Informatik ist natürlich ein interessanter und wohlbegründeter Ansatz; sie ist aber nicht die einzig mögliche, und die daraus gezogenen Folgerungen wären gegen andere, z. B. allgemeinpädagogische, abzuwägen. Geschieht das nicht, dann kann der Unterricht einseitig werden.[321]

Ähnliches lässt sich zur Didaktik Rüdiger Baumanns[322] sagen. In einer enormen Fleißarbeit hat Baumann eine erste Didaktik geschrieben, in der er einen informationswissenschaftlichen Ansatz herausarbeitet, der allerdings – weil er auf weitgehend nicht vorhandene Voraussetzungen aufsetzt[323] – in der Schule wenig Bedeutung erlangt hat[324]. Baumann beginnt mit den Bezugswissenschaften und einer Handlungstheorie, die er formal zu formulieren versucht[325] und die er als grundlegend für die Informatik bezeichnet. Leider aber findet sich in den folgenden Überlegungen zur Informatikdidaktik keinerlei Bezug mehr zur „Handlungstheorie“. Ähnliche Brüche finden sich häufiger, so z. B. wenn eine Gliederung[326], angelehnt an Kategorien der allgemeinen Didaktik, ein ganzes Arbeitsprogramm enthält, das aber nirgends angegangen wird[327]. Wenn so etwas bei einem ersten Versuch, eine geschlossene Fachdidaktik vorzulegen, auch verständlich ist, so muss doch kritisiert werden, dass Baumann immer wieder Ansprüche erhebt, die er zwar selbst nicht einlöst, trotzdem aber als Grundlage zur Kritik an bestehendem Unterricht wählt. So konstatiert er, dass *„ein Informatikunterricht, der nicht ausgiebig und intensiv Themen der KI-Forschung und der Kognitionswissenschaft aufgreift, sein (Bildungs-)Ziel verfehlt.“*[328] Danach kommen einige Seiten Aufzählungen – z. B. ganze neun Zeilen zu Diagnostiksystemen –, auf die ein bemerkenswertes Resümee folgt: *„Das im Unterricht entwickelte Minimalsystem kann natürlich keine zutreffende Vorstellung davon vermitteln, was derzeit machbar ist, und erst recht keine davon, was in Zukunft erreichbar sein wird. Daher sind Aussagen über prinzipielle Grenzen dessen, was Computer vermögen, in der Schule höchst problematisch. Es kann sich eigentlich nur um einen Beginn des Nachdenkens über Fragen dieser Art handeln.“* Wenn also einerseits das Thema essentiell für das Fach sein soll und andererseits in

---

[317] und sie sollten natürlich auch sinnvoll sein!

[318] [Hub00] S. 18: „*Kreativität kann in einer fest vorgeplanten Umgebung nicht gedeihen, ein gewisser chaotischer Bereich ist unabdingbar*.“

[319] In [Hub97] S. 45 weist er selbst auf die „*leidvollen Erfahrungen*“ mit abstrakten Konzepten hin.

[320] Die *anders* begründete Forderung nach einem Pflichtfach – z. B. durch die Bedeutung von Informatikkenntnissen als vierter Kulturtechnik – wird von dieser Kritik natürlich nicht betroffen.

[321] In diesem Zusammenhang zeigt sich auch der Sinn von Bergers Frage: „*Ist dann eine Professionalisierung in Hinsicht auf Starrheit (wie in der Mathematik) überhaupt wünschenswert. Ist Informatik nicht gerade wegen ihrer ‚Unprofessionalität’ eine Wohltat für das Schulsystem?*“ [Ber98]

[322] [Bau90], [Bau96a]

[323] z. B. ein durchgehendes Pflichtfach Informatik

[324] [Hum00] S. 20

[325] [Bau96a] S. 33: „*Er realisiert damit die Handlung H = (P,Z,T*) = opt{Ti}*“

[326] auf S. 45

[327] z. B. „*Erarbeitung der Zusammenhänge zwischen Informatikunterricht und allgemeinen Bildungszielen und Erörterung der Sinnfrage im Informatikunterricht*“.

[328] [Bau96a] S. 341

der Schule kaum behandelbar: welche Hilfe stellen diese Überlegungen dar? Baumann neigt zu Formulierungen, die bestehenden Unterricht in sehr scharfer Form abqualifizieren, ohne praktikable Alternativen zu nennen. Er will den Unterricht nicht weiterentwickeln, sondern (jeweils neu) radikal ändern, und er erhebt Ansprüche, die er leider durch seine Unterrichtsbeispiele nicht einlöst.[329] Ich halte dieses Vorgehen für gefährlich, weil dadurch die Nutzung bestehender Unterrichtserfahrungen, also Kontinuität, verhindert werden kann. Schlimmer noch: Die Entwertung des meist autodidaktisch erworbenen Wissens der Unterrichtenden kann diese entmutigen, auf ihrem Weg fortzuschreiten. Unbeschadet davon sind Baumanns Arbeiten immer eine Quelle interessanter Ideen und schlagen oft eine Bresche in neue Bereiche, die ohne seine Vorarbeiten nicht – oder wesentlich schwieriger – zugänglich wären. Leider propagiert er eher einen Unterricht, der extrem „kopfzentriert" ist und die Bedeutung von enaktiven Unterrichtsphasen m. E. völlig unterschätzt[330].

Einen ganz anderen Weg als die beiden genannten Didaktiker zeigen die Ergebnisse der „fachdidaktischen Gespräche der TU Dresden"[331], die aktuell zusammen mit Hubwiesers informationszentrierter Didaktik offensichtlich in die „Empfehlungen für ein Gesamtkonzept zur informatischen Bildung an allgemein bildenden Schulen"[332] der Gesellschaft für Informatik eingeflossen sind. Der Unterschied besteht darin, dass einerseits aktuelle Inhalte aus den allgemeindidaktischen Zielen abgeleitet werden, andererseits bestehende Ansätze zusammengeführt und fortgesetzt werden. Die von Steffen Friedrich initiierten Treffen führten zu vier Leitlinien, mit deren Hilfe *Sachkompetenz*, *Methodenkompetenz* und *Handlungskompetenz* bei den Unterrichteten erreicht werden sollen. Die angestrebte informatische Bildung wird handfest definiert und soll durch Unterrichtsbeispiele unterfüttert werden. Gelingt dieses überzeugend, dann kann aus der Initiative ein konkretes, in Schulen umsetzbares Modell für Informatikunterricht entstehen – wie es die GI jetzt ja auch anstrebt. Die Leitlinien lauten:

Umgang mit Informationen:

Verständnis für informationelle Prozesse entwickeln und wesentliche Merkmale von Informationen erkennen; Codierung als Prinzip der Informationsverarbeitung erkennen; Methoden zur Strukturierung und Darbietung von Informationen kennen und anwenden; einen Einblick in die Prinzipien der Organisation von Wissen erhalten; Methoden zur Beschaffung von Informationen kennen und nutzen; Kommunikationsnetze als Bestandteil soziotechnischer Systeme erkennen und nutzen; Datenschutz und informationelle Selbstbestimmung als Grundrecht verstehen und zum verantwortungsvollen Umgang mit Informationen bereit sein.

Wirkprinzipien von Informatiksystemen:

Aufbau und die Funktionsweise von Informatiksystemen in ihrer Einheit von Hardware und Software kennen; die Fähigkeit erwerben, sich in die Nutzung von Informatiksystemen einzuarbeiten; einen Einblick in die historische Entwicklung von Informatiksystemen erhalten; Chancen und Risiken der Anwendung von Informatiksystemen erkennen und zum verantwortungsvollen Einsatz von Informatiksystemen bereit sein.

Problemlösen mit Informatiksystemen:

Probleme erkennen, die mit Informatiksystemen gelöst werden können; einen Einblick in gesellschaftlich bedeutsame Anwendungen der Informations- und Kommunikationstechnik erhalten; erkennen, dass Modellbildung ein zentrales Element des Problemlösens mit In-

---

[329] Auch in seinen Schulbüchern nicht, die *nach* der Herausgabe der Didaktik erschienen sind. [Bau92], [Bau93]
[330] z. B. in [Bau96a] S. 305 zur technischen Informatik
[331] [Fri95], [Fri96]
[332] [GI01]

formatiksystemen ist; problemadäquate Softwarewerkzeuge (Standardsoftware, Softwaretools, Programmiersprachen) zur Lösung von Problemen auswählen und anwenden; Problemlösungen hinsichtlich ihrer Relevanz, Korrektheit und Effizienz beurteilen.

Arbeiten mit Modellen:

Erkennen, dass die Arbeit mit Informatiksystemen ein Arbeiten mit Modellen ist; erkennen, dass jedes Modell ein Abbild der Realität ist; Methoden zur Modellbildung kennen und anwenden; Phasen der Modellbildung anwenden können, insbesondere die Notwendigkeit der Modellkritik einsehen.

Die Ergebnisse der Arbeitsgruppe fußen auf Klafkis *epochaltypischen Schlüsselproblemen* und den Arbeiten von Heymann und Bussmann. Sie landen entsprechend bei Empfehlungen, die nicht alleine fachspezifisch begründet sind, sondern den pädagogischen Rahmen berücksichtigen und Raum für unterschiedliche (An-)Sichten lassen. Als Arbeitsprinzip wird exemplarischer Unterricht gefordert, und Unterrichtsinhalte werden nach den Kriterien „*handlungsorientiert; Bezug zur Praxis, zu Lebensnähe der Schüler; realer Anwendungsbezug; arbeitsteilig bearbeitbar; auf den Rechnern realisierbar; Möglichkeiten zur selbstständigen kreativ-forschenden Erarbeitung der Problemlösung; interdisziplinär; Vorkenntnisse berücksichtigend; Notwendigkeit für neue informatische Fachinhalte minimierend*“ ausgewählt, entsprechen damit weitgehend den Folgerungen aus der konstruktivistischen Sicht. Aus dem doppelten Ausgangspunkt der Arbeitsgruppe, der fachlichen und der schulischen Sicht, kann sich allerdings m. E. auch ein Schwachpunkt ergeben: Es kann zu Kollisionen führen, wenn eine klare Rangordnung fehlt – z. B. wenn die Fachinhalte nicht aus allgemeinen Zielen folgen, sondern weitgehend unabhängig gewählt werden.

Friedrich nennt als Beiträge der Informatik zur Allgemeinbildung „*Kenntnis über die Grundlagen der computergestützten Kommunikation; Sensibilisieren für Informationssicherheit; Wissen und exemplarisches Können über Programmierbarkeit (Modellierung, Algorithmisierung, Formalisierung); Transparenz von Grundfunktionen der Hardware; Beherrschen von komplexen Zusammenhängen (durch Modellierung komplexer Systeme); Bewertung von Informatiksystemen*“ und *„Wissen über Unentscheidbarkeit (als Phänomen)“*. Damit ist er nahe genug an konkretem Unterricht, ohne diesen im Detail festzulegen. Er liefert Orientierung[333] ohne zu gängeln. Es ist zu hoffen, dass die Arbeitsgruppe bei der Entwicklung der Unterrichtsbeispiele nicht in die gleiche Falle tappt wie andere Ansätze, also Ziele definiert, die nur bei vertiefter informatischer Bildung erreichbar sind, und gleichzeitig Unterricht zu beschreiben, der einer Breitenbildung dienen soll. Die im GI-Papier erfolgte Trennung nach Schulstufen – und damit nach Pflicht- und Wahlunterricht – bietet hier einen vernünftigen Ansatz[334].

[333] obwohl ich das Ziel der „Bewertung von Informatiksystemen“ schon für *sehr* hochgestochen halte!

[334] Allerdings halte ich die Übernahme von Hubwiesers Begründung für Pflichtunterricht (s. Seite 55) für bedenklich.

2.4 Zum Programmieren und der OOP

„Informatikunterricht soll kein Programmierkurs sein. Warum eigentlich nicht?“[335] Dieses Zitat von Sigrid Schubert ist in einem Umfeld, das ziemlich einhellig Programmierkurse ablehnt, einigermaßen erfrischend. Da Frau Schubert nicht gerade den leichtesten Weg zum Informatikunterricht empfiehlt, z. B. den Theorieanteil stark betont, kann die Äußerung nicht einfach übergangen werden. Es ist deshalb zu untersuchen, was unter einem „*Programmierkurs*“ eigentlich verstanden werden soll.

2.4.1 Zum Programmierkurs in der Kursfolge

Die didaktische Diskussion über die Stellung der Programmiersprachen usw. setzt m. E. fast immer eine Stufe zu spät ein. Bevor über Art und Umfang eines Programmierkurses und über die benutzten Sprachen entschieden werden kann, muss erst einmal die Aufgabe dieses Kurses im Unterricht geklärt werden. Einigkeit herrscht weitgehend darüber, dass der Informatikunterricht die Kreativität der Schülerinnen und Schüler stärken, soziales Lernen in projektartigen Unterrichtsphasen ermöglichen und eigenständiges Modellieren fördern soll. Ziehen wir weiter die von mir gefundenen Kriterien zur Auswahl von Unterrichtseinheiten heran, dann lassen sich viele davon nur erfüllen, wenn die Unterrichteten phasenweise selbstständig Probleme analysieren, modellieren und die gefundenen Modelle implementieren und testen, wobei noch nichts über die Art des Werkzeugs gesagt ist – bis auf eines: Eigenständiges Arbeiten erfordert fundierte Kenntnisse der Möglichkeiten des Werkzeuggebrauchs, und die müssen irgendwo erworben werden, bestimmt nicht nebenbei. *Wenn also selbstständiges Arbeiten gefordert wird, dann impliziert das einen sorgfältig geplanten Unterricht, in dem die dafür erforderlichen Qualifikationen erworben werden können*, *und der kostet viel Zeit.* Benutzen wir unterschiedliche Werkzeuge, dann müssen wir auch mehrfach die Zeit zum Erlernen des jeweiligen Werkzeuggebrauchs aufwenden – und dieser Aufwand will sorgfältig begründet sein. Verwenden wir die Unterrichtszeit so weitgehend zur Vermittlung von Inhalten, dass von den Schülerinnen und Schülern die Fähigkeit, z. B. die Programmiersprache als Werkzeug zu gebrauchen, nicht mehr erworben wird, dann können wir das Fach auch nicht mehr mit der Möglichkeit zu Projektunterricht, Teamarbeit und individualisiertem Lernen rechtfertigen.

Spezialkenntnisse über das benutzte Werkzeug sind außerordentlich kurzlebig und zählen sicher nicht zu den Fachinhalten, mit denen ein Schulfach Informatik begründbar ist[336]. Sie gewinnen ihre Bedeutung erst aus dem Kontext, in dem sie eingesetzt werden: der eigenständigen Schülerarbeit und den Erfahrungen, die daraus gewonnen werden. Kenntnisse über den Gebrauch der Werkzeuge sind kein Selbstzweck, und deshalb wird auch keine *systematische* Einführung in den Werkzeuggebrauch verlangt, sondern eine *erforderliche*, die sich meist aus der bearbeiteten Problemstellung ergibt. Die dafür benötigten Kenntnisse müssen sorgfältig vermittelt und geübt werden. Geeignet für den Informatikunterricht erscheinen mir nach wie vor Programmentwicklungssysteme, die eine universelle Programmiersprache enthalten und die durch die Bereitstellung geeigneter Bibliotheken so erweitert werden können, dass ein Wechsel zu anderen Werkzeugen weitgehend überflüssig wird.

---

[335] [Schu99] S. 6

[336] In diesem Sinne hat Peter Hubwieser völlig Recht.

Unter der reinen „Programmierung“ versteht man die Kodierung eines Algorithmus' in einer Programmiersprache, also die Formulierung des etwa in Struktogrammform beschriebenen Verfahrens in einer Sprache, die dann von einem Compiler in ein ausführbares Programm übersetzt werden kann. Die Entwicklung des Algorithmus selbst ist (fast) programmiersprachenfrei, allerdings hat der Programmentwickler natürlich die Möglichkeiten der „Zielsprache“ im Hinterkopf, so dass er meist Verfahren wählt, die in der benutzten Programmiersprache gut zu verwirklichen sind. Die reine Kodierung ist als Unterrichtsthema völlig ungeeignet. Unter einem Programmierkurs verstehen wir deshalb die eingestreuten Unterrichtsphasen, in denen Probleme gelöst werden, die sich sehr direkt auf bestimmte Strukturen der Programmiersprache abbilden lassen. Ein so verstandener Programmierkurs kann zwar, darf aber nicht den Informatikunterricht bilden[337].

Andererseits müssen die Lernenden sicher programmieren können, wenn sie ihre Ideen selbstständig am Computer verwirklichen sollen. Da ihre Modelle sich nicht durch Kurzprogramme, wie sie im Programmierkurs üblich sind, realisieren lassen, müssen sie ihr Werkzeug wenigstens unter den dafür erforderlichen Aspekten beherrschen – und das sind nicht wenige. Für den problemorientierten Unterricht wäre es also wünschenswert, Schülerinnen und Schüler zu haben, die in etwa wissen, welche Möglichkeiten das Programmentwicklungssystem zu Verfügung stellt, damit sie nicht laufend durch Detailprobleme für eigene Ideen blockiert sind. Es wäre trotzdem keine gute Idee, einen Programmierkurs dem eigentlichen Informatikunterricht vorzuschalten. Der größere Teil der Informatikschülerinnen und -schüler nimmt nicht an der gesamten Kursfolge teil, sondern steigt nach einem oder zwei Kursen (z. B. nach der 11. Klasse) aus. Würde deren Unterrichtszeit überwiegend von einem Programmierkurs eingenommen, dessen Rechtfertigung erst durch die Teilnahme an den Kursen höherer Semester erfolgt, dann wäre der Unterricht dieser Schülerinnen und Schüler vertan. Folglich muss ein Programmierkurs über die gesamte Kursfolge so verteilt werden, dass einerseits die Sprachstrukturen, die zur Bearbeitung bestimmter Problemklassen erforderlich sind, sicher beherrscht werden, andererseits die zu dieser Übung erforderliche Zeit in einem ausgewogenen Verhältnis zur Gesamtunterrichtszeit steht. Sinnvollerweise folgen die benötigten Sprachstrukturen aus einer umfassenderen Problemstellung, so dass Schüler und Lehrer jeweils wissen, weshalb gerade diese Programmierübungsphase eingeschoben werden muss. Ein Programmierkurs kann und soll deshalb nicht an der Systematik der Programmiersprache ausgerichtet sein. Er wird die zur Verfügung stehenden Sprachmittel nur unvollständig ausschöpfen. Er enthält nur die *benötigten*, nicht die *möglichen* Teile; diese werden allerdings sorgfältig unterrichtet.

2.4.2 Zur Programmiersprache

Ulrich Hoppe und Wolfram Luther haben einen Standpunkt formuliert, der meinen Vorstellungen entspricht: „*Ebenso wie das elementare Rechnen die ‚Primärerfahrung' der Mathematik ist, gilt dies entsprechend für das Programmieren als Primärerfahrung der Informatik.“* und *„Wenn man also (...) die Informatik als Bestandteil der Allgemeinbildung sieht, so kommt dem Programmieren eine zentrale Bedeutung zu.“*[338] Ähnlich sieht das Jürgen Burkert: *„Will Informatik seinen konstruktiven Charakter nicht aufgeben – und das würde Informatik zum Sandkastenspiel reduzieren –, müssen der Analyse des Problems und*

[337] Leider wird unter *Programmieren* oft ein solcher Kurs verstanden, auf den dann – zu Recht – „eingeprügelt“ wird.
[338] [Hop96]

seiner Zergliederung Lösungsentwürfe und deren Realisation mit dem Computer folgen."[339] Programmieren ist also wichtig, um die konstruktiven Komponenten des Faches zu erhalten, den Lernenden eigene Gestaltungsmöglichkeiten einzuräumen und ihnen – unabhängig von der Bewertung durch die Unterrichtenden – eigene Testmöglichkeiten für die Qualität ihrer Vorstellungen zu geben[340], anhand derer sie die Gültigkeit ihrer für das bearbeitete Problemfeld entwickelten Denkkonstrukte überprüfen können. Entsprechend wichtig ist das Medium, in dem diese Arbeit abläuft – die Programmiersprache.

Diskussionen über die Wahl der benutzten Programmiersprache *„sollten nicht die Diskussion um Inhalte des Informatikunterrichts ersetzen"*[341]. Trotzdem beherrscht gerade diese Diskussion seit Jahr(zehnt)en die didaktischen Veröffentlichungen. Sigrid Schubert meint: *„Es sollte davon abgegangen werden, die Programmiersprachen gegeneinander zu stellen. Die Schüler benötigen verschiedene Sprachwerkzeuge, um deren Vorzüge und Nachteile in Abhängigkeit von dem zu lösenden Problem zu verstehen. Offen ist, in welchem Umfang man über alternative Sprachkonzepte aufklären muss, um exemplarisches Verständnis zu entwickeln."* So weit, so gut. Die Konsequenzen sind aber auch offensichtlich: *„Auf Fertigkeiten wird man weitgehend verzichten müssen. Die Einsichten dominieren. Die motivierende Wirkung, die aus der Erprobung eigener Lösungen und dem Experimentieren am Computer resultiert, darf nicht verloren gehen."*[342] An dieser Stelle wird die Problematik sehr deutlich: Einerseits werden – aus fachimmanenten Gründen – verschiedene Sprachkonzepte gefordert, andererseits wird der daraus folgende offene Widerspruch zwischen „*Kennen lernen verschiedener Sprachen*" und „*Werkzeugbeherrschung*" nicht gelöst. Ohne eine konkrete Lösung, die den Zeitbedarf berücksichtigt[343], sind solche Empfehlungen für den Praktiker aber wertlos. Etwas später schreibt Frau Schubert: *„Mehrsprachigkeit gehört zum Informatikunterricht. Die verschiedenen Paradigmen sind prinzipiell gleich leistungsstark, eignen sich aber für Aufgabenklassen unterschiedlich gut. Diese Erkenntnis und die Fähigkeit zum begründeten Vergleich besitzen einen höheren Bildungswert als Vertiefungen in einem einzelnen Paradigma."*[344] Man muss fragen, wie die Fähigkeit zum „*begründeten Vergleich"* erworben wird, wenn aus Zeitgründen auf Fertigkeiten verzichtet wird (s. o.), und wo dann der Bildungswert dieser Vergleichsfähigkeit liegt. Ich meine, dass hier die Programmiersprachen bei all ihrer Bedeutung einen unangemessen hohen Stellenwert bekommen, dass sie als Selbstzweck und nicht als Medium des Lernens betrachtet werden, und dass das pädagogische Ziel des Programmierens hinter aller Fachsystematik nicht mehr gesehen wird. Der Aufwand für einen Wechsel des Werkzeugs muss immer im Kontext betrachtet werden. Im Leistungskurs oder in der Prüfungskursfolge steht die erforderliche Zeit ggf. zur Verfügung, in kürzeren Einheiten meistens nicht. Und auch wenn die Zeit vorhanden ist: Unterschiedliche Sprachkonzepte können, müssen aber nicht in unterschiedlichen Umgebungen erprobt werden. Es ist durchaus möglich, z. B. die Besonderheiten des deklarativen Programmierens zu „erfahren", indem man im Rahmen des Theorieteils einen Mini-Prolog-Interpreter selbst schreibt und dabei die neuen Konzepte (und deren Beschränkungen) kennen lernt.

---

339 [Bur94]

340 [Schu99] S. 6: *„Der persönlichkeitsbildende Wert der Informatik ist ihre spezifische Weise, Modelle aus Strukturen zu entwickeln und die experimentelle Manipulation mit diesen Modellen, die dem Lernenden seine Denkfehler aufzeigen."*

341 [Neu98]

342 [Schu99] S. 11

343 Etwas später werden für das Thema *„Problemlösen mit Algorithmen und Datenstrukturen und rekursive Arbeitsweise"* vier Stunden von insgesamt dreißig angesetzt!

344 [Schu99] S. 36

Die Wahl der Sprache ist sicherlich wichtig[345], ihre Bedeutung sollte aber auch nicht überschätzt werden. Wir wollen festhalten, dass die Programmiersprache ein Werkzeug ist, ihre Beherrschung also nicht als Selbstzweck, sondern als Mittel zur Erreichung anderer Ziele angesehen werden muss. Wie bei anderen Werkzeugen auch ist es ziemlich gleichgültig, welches von mehreren *geeigneten* ausgewählt wird. Die Wahl der Programmiersprache ist auch zeitabhängig, denn die auf Schulrechnern lauffähigen (und bezahlbaren) Programmiersprachen, besonders aber die Mächtigkeit ihrer Sprachmittel, ändern sich laufend. Selbst wenn die jeweils neueste Version wirklich für die Schule wichtige Verbesserungen enthalten sollte, so wäre auch dann ein ständiger Wechsel des Systems nicht zu vertreten. Man sollte sich gut überlegen, welche der zu einem Zeitpunkt geeigneten Sprachversionen man wählt – und dann solange wie vertretbar bei dieser Version bleiben, denn die Qualität des Unterrichts hängt auch davon ab, wie genau die Unterrichtenden das benutzte System kennen. Für die Wahl der Sprache muss die *Eignung in einer gegebenen Situation* entscheidend sein, und deshalb sind nicht nur Spracheigenschaften wichtig, sondern auch die Vorkenntnisse der Unterrichtenden, die Ausstattung der Schule und – ggf. – der Aufwand für einen Systemwechsel. Für entscheidender als die Sprache selbst halte ich sowieso die Entwicklungsumgebung, in die die Sprache integriert ist[346], sowie deren Hilfen und Testmöglichkeiten, denn in dieser Umgebung arbeiten die Lernenden, sie ist entscheidend für die Akzeptanz.

Objektorientierte Sprachen erleichtern heute die Integration zusätzlicher Werkzeuge, z. B. für den Zugriff auf Datenbanken oder zusätzliche Geräte wie Roboter o. Ä.[347] Da die Syntax dieser Sprachen mit der „Punktnotation“ zum Aufruf von Objekteigenschaften weitgehend einheitlich ist, bestehen die Unterschiede auf der in Schulen benötigten Komplexitätsebene zum großen Teil in der Notation der Kontrollstrukturen – spielen also kaum eine Rolle[348]. OOP hat sich aus guten Gründen weitgehend durchgesetzt. (Auf die Unterschiede zu traditionellen Programmiersprachen, insbesondere die Modellierungsmöglichkeiten, will ich hier aus Platzgründen nicht näher eingehen.) Der Hauptvorteil liegt m. E. darin, dass im Zusammenspiel mit integrierten Entwicklungsumgebungen die Bedeutung der „Benutzerschnittstelle“, also der Programmoberfläche, auf das angemessene geringe Maß heruntergestuft wurde. Oberflächen werden schnell „zusammengeklickt“, und dann bleibt Zeit für die eigentliche Problemlösung. Sprachen wie Delphi, Java (mit IDE) und andere ersetzen endlose Folgen von Read-Write-Anweisungen oder entsprechende Befehlsfolgen, die Oberflächen erzeugen sollen, durch einige wenige Ressourcendefinitionen, die z. B. bei Delphi im Programmtext kaum noch zu finden sind. Sie schaffen Raum für die Implementierung abstrakter Konzepte, von Datenstrukturen und anspruchsvollen Algorithmen, deren Ergebnisse nur noch an Oberflächenelemente gekoppelt werden müssen. OOP ermöglicht so wirklich inhaltsreicheren Informatikunterricht, stellt eine qualitative Verbesserung dar – und das nicht so sehr durch die neuen Sprachkonzepte[349], sondern durch die verbesserten Entwicklungswerkzeuge.

OOP-Systeme sind ein interessanter Anwendungsfall konstruktivistischer Lernvorstellungen. Einerseits ermöglichen sie relativ einfach die Entwicklung komplexer Programme, indem sie zahlreiche Details „verstecken“, andererseits verhindert dieses

[345] Ab jetzt weitgehend nach [Mod91] S. 57ff
[346] Idealerweise nimmt man eine IDE mit auswechselbaren Sprachen.
[347] Sie ersetzen so die früher gebräuchlichen Lernumgebungen mit entsprechenden Spracherweiterungen.
[348] Für die Lernenden ist jede Syntax neu – und damit austauschbar. Dass es sehr viel mehr auf *Erfahrungen im Problemlösen mit Computern* als auf spezielle Sprachkenntnisse ankommt, zeigen z. B. die im Anhang aufgeführten Ergebnisse.
[349] Die standen in den letzten Pascal-Versionen ja auch schon zur Verfügung.

Verstecken ggf. die Entwicklung eines gültigen Computermodells bei den Lernenden.[350] Ben-Ari spricht von einem „objektorientierten Paradoxon“:

> *The abstraction inherent in OOP is essential as a way of forgetting detail, and software development would be impossible without abstraction, but it seems to me that there must be an object-oriented paradox: how is it possible to forget detail that you never knew or even imagined? If students find it difficult to construct a viable model of variables and parameters, why should we believe that they can construct a viable model of an object such as a window object? Advocates of an objects-first approach seem to be rejecting Piaget's view that abstraction (or accommodation) follows assimilation.*[351]

Er hat natürlich Recht damit, dass die Lernenden in jedem Fall ein Computermodell entwickeln werden: *“Even if no effort is made to present a view of what is going on 'inside' the learners will form their own.”*[352], und auch mit der Notwendigkeit eines solchen Modells stimme ich überein: *“The lack of an effective, even if flawed, model of a computer can be a serious obstacle to teaching computer science if we accept the claim (...) that prior knowledge, even in the form of misconceptions, is essential to the construction of new knowledge.“*[353] Trotzdem ist das m. E. kein Einwand gegen OOP-Sprachen. Ben-Ari sagt nämlich nichts über die Art des zugrunde zu legenden Computermodells.[354] Folgen wir seiner Argumentation, dass es falsch ist, unbekannte Eigenschaften zu verstecken (s. o.), dann könnte Lernen immer nur auf der elementarsten Ebene beginnen. Auch die Benutzung z. B. einfacher imperativer Sprachen versteckt ja „fast alle“ Details und müsste folgerichtig z. B. zugunsten einer vorgelagerten „Bit-und-Byte-Hardwareebene“ verboten werden. Ben-Ari übersieht m. E., dass die Modelle der Lernenden aus konstruktivistischer Sicht nicht vollständig, sondern nur gültig sein müssen, und zwar gültig für die Abstraktionsebene, auf der gearbeitet wird. Wie auf anderen Gebieten[355] auch wird ausgehend von einem „Urmodell“, das eher intuitiv aus der Erfahrungswelt der Lernenden entstanden ist, zu mehr und mehr gültigen Modellen gewechselt werden müssen. Die Erfahrungswelt ist aber fast nie auf einer elementaren Ebene angeordnet, sondern entspricht meist einer Sicht von Benutzern, die z. B. technische Geräte (wie Computer) als Black-Boxes einfach benutzen. Von dieser Ebene aus kann zu „tiefer liegenden“ Fragen übergegangen werden – wie z. B. im Physikunterricht. Es können aber auch Systematiken heraus gearbeitet werden – wie z. B. in der Taxonomie der Biologie – oder Kausalketten – wie z. B. in der Ökologie oder Ökonomie. Je nach Arbeitsrichtung werden die vorhandenen, meist überwiegend ungültigen Modellvorstellungen in unterschiedlicher Hinsicht „verschärft“. Dementsprechend ist es auch erlaubt, die Interaktionen von Softwareobjekten in einem aus Erfahrungen gewonnenen Modell zu beschreiben, das tiefer liegende Eigenschaften ignoriert, dafür aber eben gültig z. B. für das Eventhandling ist. Ben-Aris Window-Objekte werden dann zwar nicht auf der Implementationsebene korrekt modelliert, dafür aber auf der Interaktionsebene. Solange diese die vorrangige Rolle spielt, ist sie auch ausreichend gültig.

---

[350] [Boy00] *„Ben-Ari (...) advocates, in particular, making explicit clear models that enables students to construct more accurate mental representations of the behaviour of the computer. This, in turn, will provide them with a cognitive basis for understanding and debugging the behaviour of programs.“*

[351] [Ben02] S. 11

[352] [Ben02] S. 8

[353] [Ben02] S. 9

[354] [Hen96] *“This wrong focus leads him to propose for example to delay programming exercises until a good computer model has been constructed by the students (no mention is made, which kind of model is needed).”*

[355] wie z. B. der Physik und der Technik

Ben-Ari übersieht m. E. auch, dass die Kenntnis tiefer liegender Details oft nur sehr eingeschränkt zum Verständnis der Phänomene auf Ebenen beiträgt, die etwas entfernt von der sind, auf der diese Details behandelt werden. Biochemische Vorgänge erklären vielleicht Vorgänge in der Zelle, aber nicht das soziale Verhalten von Populationen. Quantenmechanische Gesetze bestimmen die Vorgänge auf der atomaren Ebene, im Großen (z. B. im gravitativen Bereich) aber gar nicht – und umgekehrt. Informatiksysteme haben inzwischen eine Komplexität erreicht, die eine Beschreibung auf sehr unterschiedlichen Ebenen erfordert – je nach Gesichtspunkt. Da diese Ebenen ggf. durch viele Zwischenebenen getrennt sind, können ähnlich wie in den Naturwissenschaften Detailkenntnisse auf weit entfernten Ebenen kaum noch zum Verständnis an anderer Stelle beitragen. Modelle z. B. der Hardwareebene sind deshalb in der OOP sicherlich noch gültig, aber nicht sehr hilfreich.

2.5 Zur Klassifizierung von Unterrichtseinheiten

Die aus allgemeindidaktischen Fragestellungen gewonnenen Kriterien für Unterrichtsinhalte bilden also den Rahmen, in den sich fachliche Inhalte des Unterrichts einzufügen haben. Damit ist die *Fachsystematik*, aus der besonders im Gymnasium die Unterrichtsinhalte abgeleitet werden, ein sicherlich wichtiger, aber nicht allein entscheidender Maßstab. Für mindestens ebenso wichtig halte ich die *Unterrichtsmethoden*, denn für die Unterrichtenden ist entscheidender als das Unterrichtsthema z. B. die Frage, ob sie sich die Inhalte selbst erschließen können oder ob sie diese „präsentiert" bekommen, und den *Unterrichtskontext*, denn für die Nachhaltigkeit des Lernens ist die Einordnung fachlicher Probleme in übergeordnete fachliche und allgemeine Zusammenhänge entscheidend. Erst die Einbeziehung aller drei Gesichtspunkte macht es den Unterrichtenden möglich, den Stellenwert möglicher Unterrichtseinheiten für ihren konkreten Unterrichtsgang zu beurteilen[356]. Eine solche Beurteilung z. B. externer Materialien ist aber erforderlich, denn nur so ist für voll unterrichtende Kolleginnen und Kollegen die kontinuierliche Anpassung ihres Unterrichts an die geänderten Verhältnisse mit vertretbarem Aufwand möglich. *"No mechanisms exist to train teachers in the new technology or to keep them up to date with the field. The biggest obstacle to teaching computer science in secondary schools is the lack of teachers with the appropriate skills and training. A substantial effort in this area is needed to address the problem."*[357] Auch die Effizienz der entwickelten Unterrichtsmaterialien wird erhöht, wenn deren Bewertung erleichtert wird und damit die Nutzungsmöglichkeiten steigen[358]. Natürlich ist eine *exakte* Klassifizierung eines solch komplexen Geschehens wie Unterricht nicht möglich, besonders dann nicht, wenn die konkret Beteiligten, also die agierenden Unterrichteten und Lehrenden, mit ihren individuellen Vorstellungen gar nicht berücksichtigt werden. Ebenso natürlich ist aber eine *tenden-*

[356] [Sche97]: „*Deshalb reicht es nicht aus, die Begründung für eine Didaktik der Informatik in der Entwicklung der wissenschaftlichen Disziplin zu suchen. Vielmehr muss der Blick gerichtet sein auf die gesellschaftlichen, ökonomischen, kulturellen Entwicklungen, um von dort ausgehend zu fragen, welche allgemein bedeutsamen Antworten die jeweilige wissenschaftliche Disziplin darauf geben kann, diese Entwicklungen zu begreifen und mit zu gestalten.*"

[357] [Tuc96]

[358] [Tuc96]: *"Because of the rapid pace of change in the discipline, it is imperative that computer science institutions develop better mechanisms for sharing educational resources."*

zielle Beurteilung sehr wohl möglich, denn sonst wäre z. B. eine Evaluation von Unterricht und seinen Inhalten aussichtslos[359].

Beginnen wir mit dem ***Kontext***. Hier halte ich drei unterschiedliche Zielrichtungen für wesentlich: Die Unterrichtseinheit kann

- die *kulturelle* Bedeutung des Themas, also seinen Beitrag zur Entwicklung der spezifischen Wissenschaft sowie der Wissenschaften überhaupt herausstellen. Dieser Beitrag zur Bildung wird sich in der Schule auf relativ wenige, dann aber grundlegende Themenbereiche beschränken. Beiträge des Faches zu den materialen Anteilen einer modern verstandenen Allgemeinbildung sind hier anzusiedeln. Als Beispiel mag die Erkenntnis dienen, dass sich Fragen zur prinzipiellen Entscheidbarkeit von Problemen so formulieren lassen, dass sie einer abschließenden Behandlung zugeführt werden können. Informatikspezifisch ist auch die Methode der Simulation, die auf einem ganz anderen Weg als die Mathematik zu Prognosen über die Entwicklungsmöglichkeiten der modellierten Systeme kommt[360].
- die *gesellschaftliche* Bedeutung des Themas, also seinen Beitrag zur Erfassung und/oder Lösung über die Fachwissenschaft hinausreichender Fragestellungen sowie deren Konsequenzen betonen. Hier bietet sich für die Informatik eine Fülle von Themen an, die Klafkis Schlüsselproblemen entsprechen.
- die *fachwissenschaftliche* Bedeutung des Themas, also seinen Beitrag zum fortschreitenden Verständnis der Disziplin als solcher behandeln. Schwills fundamentale Ideen können hier als Leitlinien dienen, wobei die Forderung nach formaler Bildung an allgemein bildenden Schulen singuläre Themen fast ausschließt (es sei denn, sie seien so bedeutsam, dass sie als kultureller Beitrag des Faches einzustufen sind). Wichtiger als die Fachsystematik ist hier also die Übertragbarkeit der Lösungen und Arbeitsmethoden.

Obwohl sich diese Zielrichtungen etwas abgehoben anhören, sind sie doch für die konkrete Unterrichtsplanung bedeutsam. Ein eher kulturell ausgerichtetes Thema wird sich nicht in Einzel- oder Partnerarbeit erschließen, sondern erfordert das Unterrichtsgespräch mit einem erfahrenen Unterrichtenden. Fachliche Fragen werden nur soweit bedeutsam sein, wie sie den kulturellen Beitrag des Themas verdeutlichen. Für Entscheidbarkeitsfragen z. B. ist das Gödelisierungsverfahren nur in dem Umfang wichtig, wie es dazu beiträgt, auf diese Probleme den mathematischen Apparat anwenden zu können. Die Frage, ob das benutzte Verfahren auch effizient ist, wird hier gar keine Rolle spielen. Ein gesellschaftlicher Kontext soll u. a. Auskunft über die Rolle der Informatik und ihren Anwendungen in der Gesellschaft geben. Auch hier wird fachlichen Fragen eine eher sekundäre Rolle zugeteilt. Wesentlicher ist z. B. die Erörterung von Rahmenbedingungen, die technische Details einer politischen Bewertung zugänglich machen. Damit werden informatische Sachthemen nicht bedeutungslos, denn eine solche Diskussion darf im Fachunterricht natürlich nicht losgelöst von allen Kenntnissen geführt werden[361]. Die Zielrichtung ist aber eine andere als in eher fachlich orientierten Unterrichtsphasen.

---

[359] Das zeigt sich z. B. bei der Beurteilung von Unterrichtsstunden, wo bei den Beurteilenden meist eine erstaunliche Einigkeit im Ergebnis besteht, auch wenn dieses Ergebnis nach recht unterschiedlichen Kriterien ermittelt wurde.

[360] Andreas Schwill [Schw95] S. 25, spricht hier von *enaktiven* Modellen, im Gegensatz zu *ikonischen* und *symbolischen*.

[361] Sonst könnte sie ja im Politikunterricht stattfinden.

Bevor auf Details der fachlichen Planung eingegangen werden kann, muss m. E. unbedingt geklärt sein, welche ***Unterrichtsmethoden*** in welchem Umfang eingesetzt werden[362]. Die Frage ist so wesentlich, weil sich daraus der ***Zeitrahmen*** des Unterrichts ergibt. Nimmt man projektartiges Arbeiten, selbstständiges Problemlösen durch die Schülerinnen und Schüler, gemeinschaftliche Auswahl von Problemstellungen und Lösungsansätzen[363], ... ernst, dann *sind* das schon weitgehend die Unterrichtsinhalte. Die hierfür bereitgestellte Zeit steht für systematischen Fachunterricht nicht mehr zur Verfügung[364], und die Fachsystematik kann nur in soweit relevant sein, wie sie zur Lösung der gewählten Probleme erforderlich ist. Berücksichtigt man weiter, dass von den Unterrichteten selbst erschlossene Fachinhalte meist denn doch nicht so strukturiert worden sind, dass sich eine Zusammenfassung im Unterrichtsgespräch erübrigen würde, dann engt das den Zeitraum weiter ein, der für andere Fachfragen zur Verfügung steht.

Kommen wir zuletzt – und das scheint mir angemessen – zu den ***Fachfragen***. Deren Behandlung ergibt sich jetzt einerseits aus dem Kontext, andererseits aus dem Zeitrahmen. Damit erübrigt es sich aufzuzählen, was aus systematischen Überlegungen *wünschenswert* ist, sondern es bleibt festzulegen, was in diesem Rahmen (noch) *möglich* erscheint. Da der Rahmen jetzt aber ziemlich präzise bekannt ist, sollten die vorgeschlagenen Fachthemen in diesem Kontext dann auch realistisch und realisierbar sein – und damit gewännen die Vorschläge erheblich an Relevanz gegenüber Alternativen, die die Rahmenbedingungen nicht berücksichtigen. ***Wir hätten den Praxisbezug gewonnen, mit dem diese Arbeit beginnt.***

Fachthemen lassen sich relativ leicht den – von mir modifizierten – Ideenbäumen von Andreas Schwill zuordnen. Nummerieren wir diese durch, dann lässt sich leicht feststellen, welche Ideen in welchen Unterrichtseinheiten besonders verdeutlicht werden sollen. ***Damit haben wir ein Klassifizierungsschema unabhängig von den konkreten Inhalten***, und damit ist es auch leicht möglich, Unterrichtseinheiten gegen vergleichbare auszutauschen, wenn es den Unterrichtenden denn angemessen erscheint[365]. Eine Zusammenstellung solcher Unterrichtseinheiten spiegelt dann die fachliche Sicht des Unterrichtenden auf sein Fach, so wie sie den Unterrichteten erscheint[366]. Die Verteilung der Themen über die Bäume macht ggf. unzulässige Einseitigkeiten (z. B. eine zu starke Konzentration auf die Algorithmik) sichtbar. Ermöglicht, zumindest erleichtert wird die angesprochene ortsnahe Curriculumentwicklung einerseits, andererseits eine effizientere Unterstützung der Unterrichtenden durch Materialien der Universitäten zu neuen Themenbereichen[367].

Zu beachten ist, dass auch in so konzipierten, hier fachlich orientierten Unterrichtseinheiten die Inhalte nicht reiner Selbstzweck sind, sondern zur Verdeutlichung fundamentaler Ideen herangezogen werden. Diese Ideen müssen sich wie beschrieben bei den Lernenden entwickeln, denn diese sollen die fachliche Sicht der Unterrich-

---

[362] [Hey95] S 55: *Die allgemein bildende Qualität von Mathematik- (und Informatik-)unterricht ist nicht in erster Linie vom Stoff abhängig, der unterrichtet wird, sondern – im weitesten Sinne – von der Methode, von der Art, wie im Unterricht mit dem Stoff und miteinander umgegangen wird, oder kurz: von der Unterrichtskultur.*

[363] Wozu auch das Verwerfen untauglicher Ansätze gehört, die erstmal als untauglich erkannt werden müssen.

[364] Die Nichtberücksichtigung dieser einfachen Tatsache verhindert m. E. die Umsetzung vieler, sonst interessanter Vorschläge zur Unterrichtsgestaltung, z. B. in [Schu99]

[365] In der „virtuellen Lehrerweiterbildung Informatik in Niedersachsen VLIN“ soll genau dieses versucht werden. Da die ca. 100 Teilnehmerinnen und Teilnehmer insgesamt ca. 400 Unterrichtseinheiten zu konzipieren haben, sollte sich daraus ein Fundus von austauschbaren Einheiten entwickeln.

[366] Denn diese „erfahren“ ja nur den stattfindenden Unterricht.

[367] [Tuc96]: *“If each institution simply publishes its materials independently, there will be no easy way for others to find the right materials in the vastness of the Web or to have any assurance about their quality.”*

tenden rekonstruieren. Damit muss der Unterricht so angelegt werden, dass sich z. B. aus unterschiedlichen Beispielen das Verbindende, eben die zugrunde liegende Idee herauskristallisieren kann. Ich meine, dass in vielen Fällen die Idee selbst thematisiert werden muss, also explizit zu behandeln ist. Zumindest in der Oberstufe ist das auch ein angemessenes Thema, weil sich in diesen Ideen die spezielle Weltsicht des Faches manifestiert. Der Vergleich solcher Sichten (und Fächer), deren spezielle Beiträge und Einschränkungen, das Entwickeln einer persönlichen Stellungnahme zu diesen Fragen und damit das Entwickeln unterschiedlicher Perspektiven auch für die eigene Berufswahl gehören eindeutig in die Sekundarstufe II.

Versuchen wir jetzt einmal, diese Aspekte in Form eines „Fragebogens“ zusammenzufassen, der etwa einer neu konzipierten Unterrichtseinheit beigelegt werden kann. Ich beschränke mich auf die übergeordneten Ideen der angegeben Bäume und setze voraus, dass die eine Unterrichtseinheit Klassifizierenden wissen, was unter den angegebenen Stichworten zu verstehen ist.

Bitte klassifizieren Sie die Unterrichtseinheit nach den folgenden Gesichtspunkten. Geben Sie Ihre Einschätzung für die drei Bereiche in Prozentwerten an, die grob ihre Gewichtung der einzelnen Aspekte wiedergeben.

Thema: _____

Zeitbedarf: _____ WStd.

Voraussetzungen: _____

_____

Die Einheit dient der Verdeutlichung
der kulturellen Bedeutung des Themas zu _____ %.
gesellschaftlicher Auswirkungen des Themas zu _____ %.
rein fachlicher Aspekte zu _____ %. (insg. 100 %)

Die folgenden Unterrichtsmethoden erfordern an Unterrichtszeit ca.
Lehrervortrag: _____ %.
Unterrichtsgespräch: _____ %.
Partnerarbeit: _____ %.
Einzelarbeit: _____ %.
Projektarbeit: _____ %. (insg. 100 %)

Das Thema verdeutlicht die folgenden fundamentalen Ideen:

1. Algorithmisierung
   - 1.1 Entwurfsparadigmen (Branch and Bound, Backtracking, ...): _____ %
   - 1.2 Programmierkonzepte (Alternative, Iteration, Rekursion, ...): _____ %
   - 1.3 Ablauf (Prozess, Nebenläufigkeit, ...): _____ %
   - 1.4 Evaluation (Verifikation, Komplexität, ...): _____ %

2. strukturierte Zerlegung
   - 2.1 Modularisierung (Methoden, Hilfsmittel, ...): _____ %
   - 2.2 Hierarchisierung (Darstellung, Realisierung, ...): _____ %
   - 2.3 Orthogonalisierung (Emulation, ...): _____ %

3. Formalisierung
   - 3.1 formale Sprache (Syntax, Semantik, ...): _____ %
   - 3.2 Automat (Zustand, Übergang, Vernetzung, ...): _____ %
   - 3.3 Berechenbarkeit (Grenzen, Durchführbarkeit, ...): _____ %

(insg. 100 %)

3. Zur Theorie im Informatikunterricht

In einem sehr ausgewogenen Artikel schreibt Heidi Schelhowe 1997 in LOG IN: *„Dies aber soll doch der Nutzen einer Theorie sein, dass sie uns hilft, uns in der Welt besser zurechtzufinden, die Phänomene, mit denen wir es zu tun haben, besser zu erklären und unsere Umwelt zu gestalten.“*[368] Noch kürzer drückt es Herschel aus: *„Nichts ist praktischer als eine gute Theorie.“*[369] Ist es aber wirklich dieser Eindruck, den Schülerinnen und Schüler mitnehmen, nachdem sie sich mit theoretischen Fragestellungen beschäftigt haben[370]?

3.1 Zur Stellung der Theorie in der Schule

Bevor die Eleganz theoretischer Einsichten beschrieben wird, sollte man sich klarmachen, dass in vielen Ausbildungsgängen – und besonders in der allgemein bildenden Schule – die theoretischen Anteile zu den unbeliebtesten überhaupt gehören. Ein (derzeit noch) reines Wahlfach wie die Informatik läuft deshalb ein hohes Risiko, wenn es den Anteil theoretischer Inhalte zu erhöhen versucht. Es muss sich also sehr genau klarmachen, was es bewirken will – und kann –, bevor es sich darauf einlässt. Die Unbeliebtheit der „Theorie“ in der Schule resultiert m. E. weitgehend daraus, dass Tätigkeiten als „theoretisch“ eingestuft werden, die nur sehr eingeschränkt mit richtig verstandener theoretischer Durchdringung zu tun haben. Oft besteht die „Theorie“ aus Übungseinheiten, in denen unverstandene Kalküle auf leicht variierte „Übungsaufgaben“ angewandt werden, die mit der Erfahrungswelt der Unterrichteten kaum etwas zu tun haben.[371] Statt also durch die Ausbildung theoretischer Modelle Klarheit in ein unübersichtliches Konglomerat kaum zusammenhängender Fakten zu bringen, werden nur die aus einer Theorie folgenden Methoden praktisch geübt, wobei die Praxisrelevanz solcher Fertigkeiten immer mehr abnimmt[372]. In der Informatik, mit Maschinen als Thema, die gerade entsprechende Algorithmen ausführen können, wäre ein vergleichbares Vorgehen obskur. Um solche Entgleisungen zu vermeiden, wollen wir uns die Auswirkungen einer Theorie im technisch-naturwissenschaftlichen Bereich etwas genauer ansehen[373]:

1. **Erfahrungen werden geordnet.**
   Es werden Begriffe eingeführt, die für eine einheitliche Beschreibung der in dem bearbeiteten Gebiet auftretenden Größen geeignet sind. Meist wird dabei ein Modell entwickelt, das die unterschiedlichen Erfahrungen unter diesen Aspekten zusammenfasst.
2. **Zusammenhänge werden gezeigt.**
   Es werden Zusammenhänge zwischen den Begriffen erkannt und in Form von Gesetzen beschrieben. Oft können einfachere Gesetze komplexeres Verhalten erklären.

---

[368] [Sche97]
[369] [Her74] S. 8
[370] oft besser: beschäftigt wurden
[371] Die Beliebtheit der Schulmathematik widerspricht dieser Aussage nicht: Mathematik wird allgemein als wichtig (und damit unvermeidbar) akzeptiert. (Für die Naturwissenschaften gilt diese Akzeptanz nicht! Sie sind „vermeidbar“.) Unter dieser Voraussetzung gefällt sie vielen Schülerinnen und Schülern gerade deshalb, weil ihre Kalküle weitgehend ohne Verständnis angewandt werden können („Termumformungen“). Weil hier stures Üben viel hilft, ist das Fach „gerecht“. Mathematik ist m. E. gerade deshalb so beliebt, weil im Unterricht weitgehend gar keine Mathematik betrieben wird.
[372] Denn das können Computer nun wirklich besser.
[373] ab jetzt weitgehend nach [Mod92a] S. 351ff

3. **Zusammenhänge werden begründet.**
   Die gefundenen Gesetze werden – so weit möglich – „bewiesen". Dazu wird – ausgehend von einem Satz von als richtig erkannten Grundannahmen – entweder die Mathematik bemüht, oder es wird induktiv vorgegangen. Damit können die gefundenen Gesetze – so weit möglich – als gesichert angesehen werden.
4. **Methoden werden neu begründet und/oder neu entwickelt.**
   Mit Hilfe der bewiesenen Gesetze wird für bestehende oder neue Arbeitsmethoden gezeigt, ob und unter welchen Voraussetzungen sie das Gewünschte leisten.

Man kann sich darüber streiten, ob wirklich alle vier Punkte erfüllt sein müssen, bevor von einer theoretischen Durchdringung eines Fachgebietes gesprochen werden kann. Zumindest der dritte Punkt wird sicherlich auf vielen Gebieten kaum erfüllt sein. In diesem Sinne bilden die Anforderungen eher ein Arbeitsprogramm als eine Beschreibung des Ist-Zustandes. Man kann m. E. aber *nicht* darüber streiten, ob das Arbeiten alleine auf der Ebene des ersten Punktes schon das Attribut „theoretisch" rechtfertigt: Es tut das eindeutig nicht! Werden also in den Rahmenrichtlinien der Schule Anforderungen im Bereich der theoretischen Informatik beschrieben als *„elementare Einführung in die Anfangsgründe der ..."* noch mit dem Zusatz *„Die mit den Begriffen verbundenen Inhalte nur an einfachen Beispielen erläutern!"*[374], dann legen es diese Formulierungen in meinen Augen nicht gerade nahe, ernsthaftes Bemühen um die Vermittlung von Verständnis auf diesem Gebiet von den Lehrenden zu fordern[375]. Verstehen wir „Theoretisieren" in der genannten Form, dann müssen wir auch in der Schule erwarten, dass nach einem Theorieteil den Unterrichteten die Vorgänge im betrachteten Bereich einfacher und geordneter erscheinen als vorher. Sie müssen über klar definierte Begriffe verfügen, mit deren Hilfe sie die erlernten Arbeitsmethoden begründen können. Sie müssen sich in diesem Gebiet im Sinne von Frau Schelhowe „*besser zurechtfinden*".

Auf die ordnende Wirkung fundamentaler Ideen wurde schon ausführlich eingegangen. Wenn aber Erfahrungen geordnet werden sollen, dann müsse sie auch vorhanden sein.[376] Es muss also genügend „Stoff" bereit liegen, anhand dessen theoretisches Arbeiten verdeutlicht werden kann, und er muss genügend umfangreich sein, um die ordnende Wirkung der Ideen zu erfahren. Weil sich Ideen in Begriffen manifestieren, die aus konkreten Erfahrungen abstrahiert werden, ist die *fachspezifische Begriffsbildung* eine notwendige Vorstufe der eigentlichen theoretischen Arbeit. Und weil die Beweisanteile, in denen sich die Bedeutung der Begriffe eigentlich erst erschließt, in der Schule nur einen beschränkten Umfang haben können, kommt dieser Stufe eine gesteigerte Bedeutung zu[377]. Zur Ordnung gehört, vorhandene Erfahrungen in einem gültigen Modell interpretieren zu können. Eine wesentliche Aufgabe des Theorieteils der Informatik-Kursfolge liegt deshalb darin, die Schülerinnen und Schüler dabei zu unterstützen, tragfähige Modelle der Computerhardware einerseits und der Software andererseits zu entwickeln. Tragfähig in dem Sinne, dass zwar keine *aktuelle* Hard- und Software analysiert zu werden braucht, dass die Modelle aber über prinzipiell korrekte Eigenschaften verfügen, die das Verhalten der Informatiksysteme hinreichend genau erklären.

---

[374] ehem. Rahmenrichtlinien Informatik Niedersachsen.

[375] Natürlich sind solche Sätze vor dem Hintergrund der weitgehend fehlenden universitären Lehrerausbildung im Fach Informatik zu sehen. Sie schreiben aber einen Zustand fest, in dem die Unterrichtenden selbst kaum etwas von dem Gebiet zu verstehen brauchen, das sie lehren.

[376] Wir können nicht das Vakuum neu organisieren!

[377] Weil für die Arbeit mit diesen Begriffen beim „Theoretisieren" nur relativ wenig Zeit bleibt, muss die Bedeutung der Begriffe unabhängig davon deutlich werden.

Weiterhin sollten die Lernenden über genügend methodische Kenntnisse und Fertigkeiten verfügen, um selbstständig Zusammenhänge zwischen den Begriffen zu finden und neue Verfahren anzuwenden, nachdem sie deren Prinzipien verstanden haben. Sie erwerben diese Fertigkeiten leider meist erst im Informatikunterricht, weil andere Fächer nur sehr eingeschränkt selbstständiges Arbeiten zulassen. Der Theorieteil der Kursfolge kann aus beiden Gründen nur in der zweiten Hälfte liegen. Es zeichnet nun fast jede gute Theorie aus, dass die grundlegenden Ideen und Modelle sehr einfach gehalten und ohne große formale Kenntnisse verständlich sind. Im Bereich der theoretischen Informatik werden darüber hinaus Beschreibungsformen gewählt, die weitgehend in der praktischen Informatik Verwendung finden, also offensichtlich effiziente Werkzeuge darstellen. Wir haben damit die beiden Seiten der Informatik eng beieinander, die das Fach für die Schule so attraktiv machen: einerseits eine Grundlagenwissenschaft, andererseits eine Technikwissenschaft zu sein. Betonen wir beide Aspekte, dann können wir auch auf dem Gebiet der Theorie konstruktiv arbeiten[378], und umgekehrt sollte „*der Geist der Theorie von der ersten Stunde an präsent* (sein)“[379]. Genau das sollten wir uns wünschen: dass Schülerinnen und Schüler Erfahrungen darin machen, wie die theoretische Durchdringung von Teilen eines Fachgebietes die Effizienz ihrer Arbeit deutlich erhöht.

Die Realisierung der Modelle der Theorie bietet die Chance, theoretische Informatik als ein Anwendungsgebiet der in den vorangegangenen Kursen erlernten Methoden aufzufassen. Unterschiedliche Realisierungen führen zur Abwägung der Vor- und Nachteile unterschiedlicher Entwurfskonzepte und damit zur kritischen Reflexion eigener Arbeit. Das kann sowohl auf dem Gebiet der Datenstrukturen wie auch der Hardware geschehen: Automaten können also als Software-Objekte, als digitale Schaltwerke, als Funktionen, ... erscheinen; ihre Arbeitsweise kann auf unterschiedlichste Verfahren abgebildet werden. Die vielfältige Anwendung von Algorithmen und Datenstrukturen festigt diejenigen fundamentalen Ideen, die dort eine führende Rolle spielen, sich in den Ideebäumen der Algorithmisierung und strukturierten Zerlegung finden. Die Algorithmisierung wird zusätzlich konkretisiert, die strukturierte Zerlegung findet neue Anwendungen. Die verschiedenen Modelle der theoretischen Informatik (Automaten, Sprachen, Funktionen, ...) betonen deren Aspektcharakter. Die Gleichwertigkeit einiger Modelle legt den Wechsel zwischen diesen nahe, wenn er zweckmäßig erscheint. Die Anwendbarkeit des gleichen Modells auf völlig unterschiedliche Gebiete (Schaltungsentwurf, Übersetzerbau, ...) zeigt deren Abstraktionsgrad. Eine Verwechselung des Modells mit der Realität, wie sie z. B. beim Begriff des physikalischen „Elektrons“ nahe gelegt wird („klein, rund und blau“), ist ausgeschlossen.

Fassen wir eine Theorie nur als ein logisches System auf, in dem ausgehend von bestimmten Grundannahmen Folgerungen gezogen und *bewiesen* werden können, dann halte ich die theoretische Informatik für ziemlich ungeeignet, in der Schule unterrichtet zu werden. Wir können aber die Notationsformen der theoretischen Informatik, wie etwa die Transitionsgraphen der endlichen Automaten oder die formalen Grammatiken, als ein höchst effizientes Beschreibungsmittel für zustandsabhängige Systeme wie Schaltwerke und Übersetzer vorrangig *benutzen*. Die so beschriebenen Automaten lassen sich direkt (und sogar automatisch) in Programme oder Schaltungen übersetzen, so dass die exakte Beschreibung eines geeigneten Systems schon

---

[378] [Sche97]: *„Informatikunterricht, der allgemein bildend sein will, muss gerade diesen im Computer enthaltenen Widerspruch zwischen Abstraktem und Konkretem, zwischen streng logisch-rationalem und intuitiv-kreativem Zugang aufgreifen und erklären, die Doppelseitigkeit des Computers zwischen Automat und Medium, zwischen Turingmaschine und Interaktion einsichtig machen.“*

[379] [Nie02]

die Problemlösung beinhaltet. Die Erweiterung der Modelle zu Kellerautomaten und Turingmaschinen erlaubt Fragestellungen, die zu prinzipiellen Grenzen der Computer führen. Theoretische Schulinformatik ist damit ein Gebiet, in dem sich exemplarisch zeigen lässt, wie Wissenschaft arbeitet:

> *Für einen neuen Problemkreis werden angepasste Beschreibungsmittel („Begriffe") entwickelt, die dann geeignet sind, auch weit über die ursprüngliche Fragestellung herausreichende Probleme überhaupt erst einmal formulieren – und damit auch bearbeiten – zu können usw.*

Theoretische Informatik in der Schule ist nicht mit der Theorie in der Hochschulinformatik gleichzusetzen. Sie benutzt nur (eingeschränkt) die gleiche Sprache, zeigt dagegen sehr viel über die Art, wie Wissenschaft „fortschreitet". So aufgefasst ist ein Theoriekurs eine sehr praktische Sache. Mit Hilfe der genannten Notationsformen werden zustandsabhängige Systeme beschrieben und spezielle Lösungen entwickelt. Schaltwerke, Übersetzer, suchende und erkennende Systeme können entworfen werden. Nebenbei werden alle nur denkbaren Datenstrukturen und Algorithmen benutzt, so dass der Kurs auch den Titel „Datenstrukturen" haben könnte. Der Kurs arbeitet konstruktiv, die Arbeitsmittel werden der Theorie entliehen[380]. Dementsprechend sollten keine „nicht direkt hilfreichen" Beweise, etwa zur Äquivalenz von Automatenklassen, geliefert werden.[381] Natürlich können einzelne Fragestellungen auch in Richtung einer „echten" Theorie vertieft werden; doch es ist zu befürchten, dass die Schülerinnen und Schüler dann schnell überfordert werden. Ich meine, dass in jedem Fall der aktiven Schülerarbeit der Vorzug zu geben ist.

Der so verstandene „Theoriekurs" ist also vielfältig, interessant, deckt zahlreiche Themen ab. Er ist überhaupt nicht „trocken" – aber wen überrascht das? Ist doch nichts praktischer als eine gute Theorie (s. o.).

3.2 Fundamentale Ideen der theoretischen Informatik

Im Bereich der theoretischen Informatik besteht eine erstaunliche Einigkeit zwischen den verschiedenen Autorinnen und Autoren, die auf anderen Gebieten oft sehr unterschiedliche Meinungen vertreten. Nach Wilfried Hergets Ansicht liegen die Unterschiede zwischen Mathematik und Informatik *„... noch am ehesten (...) zwischen dem stärker strukturorientierten, statischen Aspekt der Mathematik und dem eher prozessorientierten, dynamischen Aspekt der Informatik (...)"*[382]. Bussmann und Heymann meinen: *„Unseres Erachtens lässt sich dieses Defizit* [dass die durch den Maschinencharakter des Computers gesetzten Grenzen nicht erkannt werden] *ausgleichen, wenn die Schüler den Computer als symbolverarbeitende Universalmaschine kennen lernen."*[383] Sigrid Schubert schreibt: *„Worin besteht die Theorie der Informatik? Einen Zugang bildet die Verbindung von Algorithmen, Sprachen und Maschinenmodellen."*[384] Den gemeinsamen Aspekt aller

---

[380] [Tuc96]: "*Topics in the theory of computing need to be integrated with practical topics in the curriculum at all levels, beginning with the first course."*

[381] Mit einer Ausnahme: An geeigneten Stellen muss den künftigen Studierenden gezeigt werden, welche herausragende Rolle die Mathematik in den Wissenschaften spielt, und dass sie deshalb in vielen Fächern an einer intensiven Beschäftigung mit dieser Disziplin nicht vorbeikommen werden.

[382] [Her94] S. 38

[383] [Bus87] S. 29

[384] [Schu99] S. 8

dieser Äußerungen bilden die ***informatikspezifischen Maschinenmodelle*** (Automaten), die im Gegensatz zu den statisch beschriebenen Zusammenhängen in mathematischen Formulierungen das dynamische Verhalten eines Systems als Prozesse modellieren. Der momentane ***Zustand*** des Systems wird gespeichert und durch ***Zustandsübergänge*** verändert, die durch Eingaben ausgelöst werden. Ggf. kann der Automat auch Ergebnisse in Form von Ausgaben produzieren.

Die zentrale fundamentale Idee dieses Gebiets scheint mir deshalb die des Zustands zu sein, wobei sie sich im Modell des Automaten manifestiert.

Seine Anschaulichkeit macht dieses Modell besonders für die Schule geeignet. Es findet in der Fachwissenschaft auf sehr unterschiedlichen Gebieten und sehr unterschiedlichen Niveaus Anwendung (Horizontal- und Vertikalkriterium), wird seit den Anfängen der Informatik benutzt und besitzt gerade durch seine Anschaulichkeit eine lebensweltliche Bedeutung (Zeit- und Sinnkriterium), weil sich Aspekte sehr unterschiedlicher im Alltag benutzter Systeme und eben der Computer selbst auf diese Idee reduzieren lassen.

Damit kommen wir zum Bereich der Eingaben, der „Bedienung“ der Maschinen. Traditionell werden diese durch Eingabebänder beschrieben, die mit den geplanten Eingaben vorab beschrieben werden, also bevor die Maschine zu arbeiten beginnt. Diese Zusammenfassung von Eingabe und Maschine zu einer Einheit ist notwendig für die Arbeitsweise von Turingmaschinen, da diese das Eingabeband selbst manipulieren. Die einfacheren Maschinenmodelle (endliche Automaten und Kellerautomaten) können m. E. durchaus getrennt vom Eingabeband als Entitäten aufgefasst werden, so dass diesen die Einschränkung der „Vorabeingabe“ nicht eigen ist. Sie müssen nur zu jedem Arbeitstakt über ein zulässiges Eingabezeichen verfügen. Nach welchen Regeln dieses produziert wird, ob es vielleicht sogar von Ausgaben anderer Automaten stammt, wie in ***Netzen*** üblich, darüber ist erst mal gar nichts gesagt. Erst die *gelesene Eingabefolge* muss gewissen syntaktischen Regeln genügen[385]! Damit entfällt für diese Automatenklassen, mit denen in der Schule überwiegend gearbeitet wird, weitgehend auch der Einwand von Peter Rechenberg[386] und Birgit Schelhowe[387], die in Turingmaschinen kein adäquates Modell für „interagierende“ und „kommunizierende“ Computer(systeme) sehen. Einfache ereignisgesteuerte Systeme lassen sich sehr wohl auf einfache Automaten abbilden. Dass diese für realistische Fälle völlig unübersichtlich werden, ist kein Einwand, denn auch die Transitionsgraphen traditioneller Parser für echte Programmiersprachen sind natürlich kaum noch aufschreibbar. Obwohl die Arbeitsweise der traditionellen Automatenklassen sequentiell ist, kann die Wirkung von Interaktionen leicht über die Kopplung solcher Maschinen erfahren werden. Ordnet man einfache Automaten in einem ***Netz*** als zellulären Automat an, dann können unter geeigneten Bedingungen ganz neue Verhaltensweisen erscheinen. Auch die Modellierung von OOP-Systemen, deren Objekte meist gut als Automaten zu beschreiben sind, die mithilfe von Events kommunizieren, liefert über die nicht sequenziell ablaufende Ereignissteuerung Erfahrungen in „Vernetzung“. Ich halte also die Automatenmodelle gerade auch dann für geeignet, wenn wie heute (hoffentlich) üblich mit objektorientierten Sprachen gearbeitet wird,

wobei als zweite fundamentale Idee die Vernetzung zum Tragen kommt.

[385] Wenn sie „korrekt“ sein soll.
[386] [Rec97] S. 32
[387] [Sche97] S. 29

Statt uns auf die Arbeitsweise der Maschine selbst zu konzentrieren, betrachten wir nun deren Steuerung. Die Syntax der von einem Automaten akzeptierten Zeichenfolgen klassifiziert diesen ebenso wie seine innere Konstruktion. Die Verlagerung des Blickpunkts bewirkt einen Wechsel des Modells, ohne das beschriebene System selbst zu ändern. Interessieren wir uns eher für die Konstruktion des eigentlichen „Apparats", z. B. eines Parsers oder einer Schaltung, dann wählen wir das Automatenmodell. Wollen wir eher dessen „Drumherum" beschreiben, dann arbeiten wir mit Grammatiken. Bei Bedarf wechseln wir zwischen diesen Beschreibungen, z. B. weil im alternativen Modell bessere Werkzeuge zur Verfügung stehen.

Damit haben wir als dritte fundamentale Idee die Sprache gefunden.

Die Idee, Kommunikationsmittel syntaktisch zu beschreiben, gehört im Bereich der Schule zum Standard. Die Analyse von Sprachkonstrukten mithilfe von Grammatiken ist zwar mit der Verdrängung der alten Sprachen (leider) etwas in den Hintergrund geraten. Das ändert aber nichts am Wert der Erfahrung, formale Regelsysteme systematisch einzusetzen und deren Ergebnisse kritisch nach semantischen Gesichtspunkten zu bewerten. Die Verankerung in der Lebenswelt der Lernenden ist also auch hier gegeben. Die restlichen fachimmanenten Kriterien sind offensichtlich erfüllt.

Es bleibt als Letztes die Idee der Berechenbarkeit.

Dass Computer rechnen können, ist sicherlich für niemanden eine Überraschung. Berechnungen gehörten schließlich zu den ersten Aufgaben des „Rechners". Ansatzpunkte aus der Erfahrungswelt der Lernenden gibt es dafür genug. Unter Berechenbarkeit verstehen wir allerdings mehr die Frage danach, was alles berechnet werden kann[388], also Betrachtungen über die Grenzen der Computer in dieser Hinsicht. Zu diesen Grenzen gehören sowohl die Frage, ob es Grenzen für algorithmische Verfahren gibt, als auch, wo die Grenzen der Gültigkeit der Ergebnisse von Berechnungen liegen. Beide Themen durchziehen die Informatik von Anfang an, spielen auf unterschiedlichen Ebenen und in verschiedenen Bereichen eine zentrale Rolle. Damit sind Schwills Kriterien erfüllt. Beide Themen liefern aber auch sowohl sehr tiefgehende Fragen (und einige Antworten) als auch eine Fülle außerordentlich motivierender Beispiele, etwa aus den Bereichen des deterministischen Chaos oder der Simulation vernetzter Systeme.

3.3 Zur Rekonstruktion der fundamentalen Ideen

Beschränken wir uns auf die genannten fundamentalen Ideen der Informatik, so ist immer noch nicht geklärt, wie sich diese bei den Unterrichteten denn bilden sollen. Im Sinne des Konstruktivismus können wir uns die Vorschläge von Ben-Ari zu Herzen nehmen[389]:

A Guide for Educators:

- *Regardless of your teaching technique (lectures, labs, assignments), you must articulate to yourself the cognitive change that you wish to bring about in the students and structure the activity to achieve this aim. Merely transferring knowledge is not a meaningful aim.*

[388] [Bus87] S. 15: *„Deshalb scheint es uns notwendig, eine grundlegende gedankliche Durchdringung der Maschine ‚Computer' als Produkt des Menschen zum Ausgangspunkt zu machen."*
[389] [Ben02] S. 15

- *You must dig underneath your own expert knowledge to expose the prior knowledge needed to construct a viable model of the material that you are teaching. You must ensure that the students have this prior knowledge.*
- *In any particular course you will be teaching a specific level of abstraction; you must explicitly present a viable model one level beneath the one you are teaching.*
- *When a student makes a mistake or otherwise displays a lack of understanding, you must assume that the student has a more-or-less consistent, but non-viable, mental model. Your task as a teacher is to elicit this model and guide the student in its modification.*
- *You must provide as much opportunity as possible for individual reflection (for example, analysis of errors) and social interaction (for example, group labs).*

Wir können die Forderungen auch etwas kürzer fassen:

- *Einerseits müssen die individuellen Vorstellungen der Lernenden berücksichtigt und zielgerichtet weiterentwickelt werden,*
- *andererseits erfordert die Ausbildung veränderter mentaler Strukturen die aktive Auseinandersetzung mit Fragestellungen des bearbeiteten Bereichs auf möglichst vielfältige Weise.*

Beginnen wir mit den mentalen Modellen:

Eine diffuse Vorstellung von ***Automaten*** als Beispielen von *„(1) Maschinen, die etwas tun*“, hat vermutlich jeder. Das Tun beinhaltet Aktivität, also Dynamik. Wenn das Tun nicht nur aus einer einzigen Aktion besteht, dann kann das Anfangsmodell weiterentwickelt werden zu *„(2) Maschinen, die schrittweise etwas tun“*. Im einfachsten Fall wird dieses Nacheinander der Aktionen durch eine Art Zeittakt ausgelöst werden. Damit benötigen wir *„(3) Maschinen, die wissen, was als nächstes zu tun ist“*. Diese Information muss in den Maschinen vorhanden sein, dort gespeichert werden. Nach jeder Aktion muss sich damit diese Information ändern. Wir können sagen, dass wir es nun mit *„(4) Maschinen, die sich in unterschiedlichen Zuständen befinden können“* zu tun haben. Damit legt der ***Zustand*** fest, was als nächstes zu tun ist. Wir haben eine sequenziell arbeitende Maschine, die wir durch eine Folge von Zuständen und ***Übergängen*** wie üblich beschreiben können. Erweitern wir das Modell um einzugebende Steuerzeichen, dann können unsere Maschinen aus einem Zustand in unterschiedliche Folgezustände übergehen. Wir kommen zu den bekannten Transitionsgraphen.

Diese kurze Folge von Modellen will nun konstruiert sein. Dazu benötigen wir möglichst vielfältiges und möglichst interessantes Aufgabenmaterial, das an die Erfahrungswelt der Schülerinnen und Schüler anknüpft. Da es sich bei den hier zugrunde gelegten endlichen Automaten meist um ziemlich einfache Maschinen handelt, brauchen wir dafür keine besondere Unterrichtseinheit. Im Gegenteil: Weil die Automaten sich so leicht aufzeichnen lassen, sollten sie als Hilfsmittel und Teillösungen innerhalb von umfangreicheren Problemstellungen auftauchen:

- Schon ganz am Anfang der Kursfolge, wenn Zeichenketten modifiziert, durchsucht, verändert werden, z. B. bei Verschlüsselungsproblemen.
- In Unterrichtsprojekten etwa aus der Bioinformatik, wenn z. B. DNA-Replikation durch Polymerasen simuliert wird.
- Zur Beschreibung von Lichtschranken, Bahnübergängen, Alarmanlagen, …
- Zur Entwicklung von Schaltwerken wie Speichern, Addierern, …

Die übergreifende Einsetzbarkeit von Automatenmodellen lässt die Lernenden deren Fundamentalität erfahren. Die aktive Nutzung macht sie mit deren Möglichkeiten, Tücken und Grenzen vertraut. Die unterschiedliche Realisierung über Funktionen, OOP-Objekte und Schaltungen verdeutlicht deren Modellcharakter und ihre Brauchbarkeit als Werkzeug. Die solide mentale Verankerung dieser Modelle bietet dann den Zugang zu den weiteren fundamentalen Ideen des Theorieteils.

Verlagern wir nun unser Interesse auf die Bedienung von Automaten, die auf unterschiedliche Eingabezeichen reagieren können, so kommen wir auf die Frage nach deren Steuerbarkeit. Es gibt Eingabefolgen, die zu unsinnigen Ergebnissen führen, und solche, die einen angestrebten Zweck erfüllen. Wahrscheinlich gibt es auch unterschiedliche Zeichenfolgen, die zum gleichen Ergebnis führen. Diese bilden ***Sprachen***, deren Worte etwas bewirken. Die *Idee* der Sprache ist den Lernenden natürlich vertraut. Neu für sie ist die Anwendung dieses Begriffs auf die Kommunikation mit Maschinen, vor allem die Anwendung auf so einfache Konstrukte, wie wir sie meist behandeln. Da wir es noch nicht mit Programmiersprachen zu tun haben, halten die Unterrichteten es anfangs eher für absonderlich, kurzen Befehlsfolgen das Attribut „Sprache“ zuzuschreiben. Sie brauchen deshalb etwas Zeit zur Gewöhnung, vor allem daran, Sprachen rein formal zu behandeln. Sie finden diese Zeit in der konstruktiven Auseinandersetzung mit entsprechenden Problemen.

Die ***Struktur*** dieser Sprachen kann wie üblich durch Grammatiken beschrieben werden, wobei die Zugehörigkeit eines Wortes zu einer Sprache diesem eine ***Bedeutung*** gibt, wenn man sie auf den Zielautomaten anwendet. Die Schülerinnen und Schüler können nun im selben Kontext wie oben, aber mit einer veränderten Sichtweise Automaten steuern. Sie können geeignete Befehlsworte erzeugen, solche mithilfe von Parsern (also „Prüfautomaten“) testen, die Ergebnisse simulieren und erst dann dem „echten“ Zielautomaten zuführen. Ein außerordentlich motivierender Kontext hierfür ist die Beschäftigung mit kleinen Robotern[390], aber auch Technikmodelle, deren Motoren durch Relais geschaltet werden, Turtlegrafik-Umgebungen, Sprachspiele („Zufallsgedichte“, „Elisa“, …) sind sehr beliebt. Bleiben die Automaten und ihre Sprachen nur Hilfsmittel, so sind auch diese Teile in vorgelagerte Kurse integrierbar. Beschäftigen wir uns systematisch mit ihnen, dann sind eigene Unterrichtseinheiten etwa zum Thema „Compilerbau“ erforderlich.

Die Idee der ***Vernetzung*** wird besonders deutlich, wenn wir die vernetzten Komponenten durch Automatenmodelle beschreiben. Die Kommunikation erfolgt durch die Verknüpfung der Ein- und Ausgabekanäle der Maschinen – ein sehr anschauliches Modell. Setzen wir die Automaten in ein festes Gitter, dann erhalten wir zelluläre Automaten mit all ihren vielfältigen und auch optisch interessanten Einsatzmöglichkeiten[391]. Verknüpfen wir sie durch frei zu sendende Botschaften, dann haben wir ein Modell für Teilbereiche der OOP. Bilden wir sie auf die Maschen eines Netzes ab, dann finden wir Zugang z. B. zu den Protokollen des Internets. Gerade in diesem Bereich bieten die Möglichkeiten aktueller Programmiersprachen (z. B. von Java) für die Schule neue und relativ einfach realisierbare motivierende Möglichkeiten.

Die Idee der ***Berechenbarkeit*** ist, wenn man sie so wie in der theoretischen Informatik üblich auffasst, für Lernende neu und fremd. Hier halte ich entsprechende Einsichten nur als Endergebnis des Lernprozesses für möglich. Der Weg von Alltagserfahrungen hin zu Entscheidbarkeitsproblemen ist wohl doch zu weit, um ihn weitgehend durch Eigenaktivitäten zu finden. Es ist aber durchaus möglich, in Eigenarbeit

[390] z. B. die LEGO-Mindstorms-Maschinen
[391] Beispiele dafür z. B. in [Ger95]

so viel Erfahrungen mit den Einzelbausteinen dieses Weges (Codierungen, Turingmaschinen, …) zu machen, dass eine geschlossene Darstellung mit ihren verblüffend weittragenden Aussagen gut machbar wird. Ein möglicher Durchgang wird unten beschrieben. Die numerischen und durch Iterationen und Kombinationen auftretenden Grenzen sind allerdings nahe liegend und durch Experimente direkt erfahrbar[392]. Sie bieten ebenso wie die zellulären Automaten hochinteressante Probleme schon für den Anfangsunterricht.

3.4 Technische Informatik im Theoriekurs?

Obwohl Themen der technischen Informatik[393] zu den ältesten Inhalten der Schulinformatik gehören[394], scheint mir auf diesem Gebiet am wenigsten geklärt, ob und wie das Thema an allgemein bildenden Schulen unterrichtet werden soll.[395] Einen Hinweis erhalten wir z. B. von Mats Daniels et. al.:

> *One challenge in computer science is that the constraints to thinking within the discipline are not physical, but human: our artefacts are constrained primarily by our ability* ***to invent****. Hence computer science teaching is about what computer scientists have managed to think about so far, and in what manner: algorithms, paradigms, languages, engines, tools, solutions are all thought products. But they are thought products that interact crucially with the physical world, and the relationship between the reasoning discipline of computer science and its technology is central to its particular character.*[396]

Bei der Hardware von Informatiksystemen handelt es sich genauso um „Denkprodukte“ wie bei der Software. Sie manifestieren sich nur in einem anderen Medium. Auch Hardware ist in hohem Maße abstrakt, und sie ist in der Schule ähnlich wie die Softwareentwicklung ein exzellentes Anwendungsfeld, um über abstrakte Entwurfs- und Modellierungsverfahren[397] zu konkreten Ergebnissen zu kommen. In dieser Hinsicht ist sie ein fast schon ideales Anwendungsgebiet für Projektarbeit, die über die Sensorik und Steuerungsvorgänge auch mit anderen Unterrichtsfächern eng verknüpft werden kann.

Wenn es sich bei der digitalen Elektronik um eine Basistechnologie handelt, auf deren Grundlagen weite Bereiche unserer Gesellschaft beruhen, dann gehört dieses Thema in den Unterricht allgemein bildender Schulen, weil Kenntnisse auf diesem Gebiet zum Verständnis unserer Umwelt notwendig sind. Wir haben einen Fall, der direkt den Anwendungen des Elektromagnetismus vergleichbar ist: Auch hier werden die erforderlichen Grundlagenkenntnisse – im Physikunterricht – vermittelt, und zwar allen Schülerinnen und Schülern! Folglich gehört m. E. die Behandlung einfacher Schaltnetze in die Sek. I und ist damit Thema der ITG. Auch vom Anspruchsniveau sind Probleme wie

> WENN *die Alarmanlage eingeschaltet worden ist*
> UND *es später als 22.00 h ist*
> UND *die Fensterscheibe zerstört wird,*
> DANN FOLGT, *dass die bösen Buben kommen*

---

[392] z. B. in [Hen98] S. 12
[393] z. B. [Mod92b], [Gas92]
[394] ab hier weitgehend nach [Mod92a] S. 326
[395] Natürlich kann es unterrichtet werden, denn gerade hier liegen aus der Phase der „Hardwareorientierung“ reichlich Unterrichtsmaterialien und -erfahrungen vor.
[396] [Dan00]
[397] z. B. mithilfe von endlichen Automaten

eher auf dieser Altersstufe angesiedelt und werden dort seit Jahren erfolgreich unterrichtet. Konsequenterweise können diese Themen dann alleine nicht mehr die „technischen“ Inhalte eines Informatik-Grundkurses bilden. Möglicherweise werden sie auftauchen, weil Defizite der Sek. I aufzuarbeiten sind, aber mehr auch nicht. Der Informatikunterricht der Sek. II würde bei Beschränkung auf solche Themen nur ITG-Aufgaben wahrnehmen, die nicht zu seinem eigentlichen Kern gehören.

Sehen wir uns die beim Schaltungsentwurf meist benutzten Methoden der Schaltalgebra etwas genauer an, dann stellen wir fest, dass die eigentliche Schaltungsentwicklung gar nicht Thema dieser Disziplin ist. Schaltfunktionen werden nach unterschiedlichen Kriterien umgeformt, bis sie bestimmten Anforderungen z. B. der Einfachheit oder der Darstellungsform genügen. Jeder dieser Schaltfunktionen entspricht aber direkt eine Schaltung, deren Aufschreiben eine eher triviale Aufgabe darstellt. Haben wir überhaupt eine Schaltfunktion, die ein Problem löst, dann haben wir auch die entsprechende Schaltung. Diese ist vielleicht zu umfangreich, nicht effizient, ..., aber sie löst das Problem; und da Effizienzkriterien m. E. in allgemein bildenden Schulen nur sehr eingeschränkt Sinn haben, sind die mathematischen Methoden der Schaltalgebra zumindest nicht notwendig, um Schaltungsentwurf in der Schulinformatik zu betreiben. Die Methoden der Schaltalgebra sind deshalb ggf. hilfreich, aber zur Begründung der Existenz der Technik innerhalb des Informatikunterrichts ebenfalls nicht geeignet.

Viel wichtiger als mathematisch begründete Umformungen ist die Frage nach der Existenz einer Schaltfunktion für eine gegebene Problemstellung. Im Informatikunterricht sollte deutlich werden, dass einerseits die aussagenlogische Formulierung einer Problemstellung (WENN *dasunddas gegeben ist* DANN FOLGT *diesunddas*) eine Lösung impliziert, andererseits aber nur ein eingeschränkter Satz von Problemklassen auf diese Art behandelbar ist: Gerade bei den „interessanten“ Fragen lässt sich *„dasunddas“* und *„diesunddas“* nicht aussagenlogisch scharf fassen. Standardbeispiel für solch einen „unscharfen“ Begriff ist die Größe eines Menschen: Sicherlich ist ein 1,95 m-Mensch „groß“, und ebenso sicher ist ein 1,55 m-Mensch „klein“. Wie aber sind 1,80 m einzuschätzen? Als „ziemlich groß“?[398] Die Existenz anderer als aussagenlogischer Methoden (Fuzzy-Logik, neuronale Netze) zeigt sehr deutlich, dass mithilfe der Aussagenlogik formulierbare Probleme noch nicht einmal die Bereiche vollständig abdecken, in denen schon heute technisch „hart“ gearbeitet (gesteuert, geregelt, ...) werden soll: Wir brauchen gar nicht die „unscharfen“ Begriffe der Geisteswissenschaften wie „Schönheit“, „Harmonie“, ... zu bemühen; auch die Steuerung eines Brennofens oder einer Videokamera geht oft über die Möglichkeiten der traditionellen Aussagenlogik hinaus[399]. Im Hardwareunterricht können also die Möglichkeiten und Grenzen einer Methode erfahren werden, die zwar für einige der so gern behandelten „klar formulierten“ Probleme wunderbar funktioniert, für andere aus dieser Klasse aber schon nicht mehr; und die Grenzen der Technik sind ein zentrales Thema der Informatik: sei es bei den Algorithmen, sei es an dieser Stelle. Im Gegensatz zu den Berechenbarkeits- und Entscheidbarkeitsproblemen handelt es sich bei der Schaltungsentwicklung aber um sehr handfeste Probleme, so dass die Grenzen der Computer nicht in (für die Schülerinnen und Schüler) esoterische Bereiche verschoben werden.

[398] Die Fragestellung ist direkt auf Sensorenabfragen, wie sie zur Steuerung von Schaltungen benutzt werden, übertragbar.
[399] Was z. B. ist eine „scharfe“ Bremsung?

Die zentrale Aufgabe des Hardwareunterrichts ist es m. E. zu zeigen, wie technische Komponenten Eigenschaften erwerben, die sonst eher dem geistigen Bereich zugeordnet werden – und dafür ein gültiges Modell zu entwickeln: wie also eine Schaltung (in vorgegebener Weise) sinnvoll auf unterschiedliche Situationen reagieren kann, Entscheidungen trifft, sich etwas merkt oder programmiert werden kann.[400] Begreifen wir digitale Systeme als ein System von „Schaltern", dann reduziert sich unser Problem auf die Frage: „Wer bedient die Schalter?" Nun kann man schnell zeigen, dass einerseits Schalter von außen bedient werden (durch „Sensoren", Menschen, ...), andererseits Schaltersysteme auch auf sich selbst zurückwirken können. Wir kommen dadurch zu einer Konkretisierung des Begriffs des ***Zustands*** des Systems, der z. B. als eine Art Gedächtnis (Speicher, ...) fungieren kann. Die Abfolge solcher Zustände beschreibt dann die Reaktionen des Systems; und wenn unterschiedliche Abfolgen möglich sind, dann kann die Schaltung auch auf unterschiedliche Weise reagieren. Beschreiben wir die Zustände und deren Änderungen auf eine geeignete, systematische Art und zeigen, wie die so gefundene Beschreibung wiederum systematisch in eine Schaltung umgesetzt werden kann, dann haben wir unsere Aufgabe gelöst: Wir können beliebige (auch programmierbare) Schaltungen entwerfen.

Es kommt dabei nicht so sehr auf die Komplexität der Ergebnisse als auf das systematische Vorgehen an: Für die Schule (und die Schülerinnen und Schüler) ist es „normal", mächtige Verfahren anhand einfacher Beispiele kennen zu lernen und einzuüben. Technische Informatik darf sich damit nicht auf „Kochrezepte" beschränken, die zeigen, dass eine vorgegebene Schaltung funktioniert (z. B. ein FlipFlop aus NANDs). Vielmehr müssen Methoden entwickelt werden, die zeigen, wie man zu einer Lösung kommt. Als systematisches Verfahren zum Schaltungsentwurf bieten sich die Notationsformen der Automatentheorie und die dazu passenden Methoden an. Hier liegt auch die enge Verbindung zur theoretischen Informatik, für die der Technikkurs eine Art „projektorientierte Vorstufe" bilden sollte. Entsprechend betont die technische Informatik *die gleichen fundamentalen Ideen* wie in der Theorie: (hier: konkrete) Automaten mit ihren Zuständen, Berechenbarkeit und Sprachen, die die Automaten steuern, manifestieren sich zwar anders, aber in vergleichbarer Weise[401]. Strukturierte Zerlegung und Algorithmisierung – mit ihren „Unterideen" – zeigen sich beim Entwurf von Schaltungen sowie im Bereich der Simulation.

Wichtig erscheint mir die Erfahrung, dass quantitative Änderungen der Zahl der Schalter in einer Schaltung qualitative Änderungen bewirkt: Mit einem kann man eine Glühbirne „schalten", mit zwei oder drei Schaltern eine Alarmanlage bauen, mit weniger als zehn ein Treibhaus steuern, mit einigen Dutzend einfache Rechnungen durchführen, mit Hunderten einen Taschenrechner bauen und mit Tausenden, Millionen, ... ? Zu jeder Größenordung von Zahlen gehört eine andere Problemklasse, die damit gelöst werden kann. Miniaturisierung ist nicht nur ein technisches Problem.

Ich habe mit Absicht die sehr „tiefen" Ebenen des Schaltungsentwurfs zuerst besprochen, obwohl mir natürlich bewusst ist, dass man auf diese Art keinen Computer entwickeln kann. Zur Beschreibung derart komplexer Systeme gehören die Benutzung von Blockschaltbildern, das Denken in Komponenten und die Beschreibungsformen der Rechnerarchitekturen[402]. Legt man eine bestimmte Struktur zugrunde (von-Neumann-Rechner, ...), dann kann in diesem Bild auf der entsprechenden Softwareebene gearbeitet werden: Man programmiert im Assembler. Wohlgemerkt:

[400] Ich meine also nicht eine technische Variante des Leib-Seele-Problems, sondern „harte" Computertechnik.
[401] Die Berechenbarkeit tritt hier meist unter dem Aspekt der Durchführbarkeit auf.
[402] Mir ist dieser Begriff für die Schule eigentlich zu hochgestochen, aber er hat sich nun einmal auch hier eingebürgert.

man kann – aber wozu? In Blockschaltbildern dieser Art erscheint die Struktur von Computern sicherlich differenzierter als im Anfangsunterricht, und es lassen sich natürlich auch unterschiedliche Architekturen unterscheiden. Gehen wir davon aus, dass nicht die Einzelheiten des gerade benutzten Prozessors und seiner Komponenten behandelt werden[403], sondern prinzipielle Strukturen, dann ist das Blockschaltbild eines Von-Neumann-Rechners nicht sehr kompliziert und in kurzer Zeit zu erlernen. Die oben genannten Fragen werden dabei sicherlich präzisiert, aber sie werden nicht beantwortet. Für die Schülerinnen und Schüler bleibt noch offen, wie weit sie von den technischen Grundlagen entfernt sind: Sie „schwimmen" auf einem Meer von Fragen, dessen Tiefe sie nicht einschätzen können. Die Frage, wie eine Schaltung „etwas entscheidet", bleibt bestehen. Ich halte deshalb den Umgang mit „echter" Hardware auf dem Niveau der Schaltalgebra (im Kontext eines Computer-Blockschaltbildes) für unverzichtbar. Sollten aus Zeitgründen nur entweder Rechnerarchitekturen (mit (Pseudo-) Assemblerprogrammierung) oder Schaltungsentwurf betrieben werden können, dann würde ich dem zweiten immer den Vorrang geben. Ohne diese Basis erscheint mir die Assemblerprogrammierung nur als (überflüssiger) Wechsel der Programmiersprache, der kaum neue Erkenntnisse bringt, aber viel Zeit erfordert, die mit anderen Themen m. E. besser genutzt werden könnte. Umgekehrt aber kann bei Kenntnis der Methoden, programmierbare Schaltwerke zu entwerfen, die Arbeit mit einem (Pseudo-)Assembler die Kluft zwischen den (Modell-)Rechnern und den höheren Programmiersprachen schließen. Die algorithmischen Grundstrukturen können auf Sequenzen maschinennaher Befehle zurückgeführt werden, die einzeln direkt Schaltungskomponenten und -funktionen zuzuordnen sind und deren Komplexität mit den Mitteln der Schule zu bewältigen ist. Wird durch die Methoden des Schaltungsentwurfs eine solide Basis gelegt, dann kann auf dieser technisches Wissen über Computer erworben werden, das keinen Raum mehr für „Geheimnisse" lässt.[404]

Fassen wir also die Anforderungen an die technische Informatik innerhalb der Kursfolge noch einmal zusammen:

1. *Kenntnisse über den Entwurf und die Nutzung von Schaltnetzen gehören zumindest teilweise in den Bereich der ITG und sollten vom Informatikunterricht vorausgesetzt werden können. Fehlen Sie, dann müssen sie nachgeholt werden. In keinem Fall darf sich die Informatik aber auf das Thema beschränken. Zu diesem Bereich gehören die systematische Aufstellung von logischen Schaltfunktionen und deren Umsetzung in Gatterschaltungen. Beispiele finden sich bei Fragen des Steuerns und Regelns und bei einfachen Rechenschaltungen. Die Umformung und Vereinfachung von Schaltfunktionen mit mathematischen Methoden ist weit weniger wichtig als Fragen nach den Möglichkeiten und Grenzen des Verfahrens.*
2. *Wichtig für das Verständnis sind die Eigenschaften von Schalternetzen. Welche Art von Schaltern verwendet wird, ist eigentlich egal. Ich glaube nicht, dass der Physikunterricht der Sek. I tragfähige Grundlagen für die Verwendung von Halbleiter-*

[403] was ich für selbstverständlich halte.

[404] Wenn auch neuronale Netze nicht gerade im Zentrum des technischen Informatikunterrichts stehen, so halte ich sie als Kontrast zu den systematischen Verfahren des Schaltungsentwurfs für außerordentlich hilfreich. Diese Netze „lernen" ja (wobei die dabei erfolgenden Vorgänge auf dem Niveau der Sek. II mathematisch gut beschrieben werden können), und sie kommen etwa in der Mustererkennung zu Antworten, die auf anderem Wege nicht oder nur mit wesentlich höherem Aufwand zu erhalten sind. Dabei erkennen sie Muster, die von den gelernten ziemlich stark abweichen können, oft gut wieder. Manchmal aber auch nicht! Und ob sie zu einer Lösung kommen, wie sie zu einer Lösung kommen, ob dieses die beste Lösung ist (oder nur eine lokale Näherung), wie sicher die Antwort ist: All dieses können die Netze nicht beantworten – und wir auch nicht.

schaltern (TTL, ...) liefert. Wir sollten uns deshalb auf ein einfacheres Modell beschränken, z. B. auf elektromechanische Systeme.[405]

3. *Wirken Schaltungen auf sich selbst zurück, dann ergeben sich Schaltwerke mit unterschiedlichen Zuständen. Als systematisches Beschreibungsmittel bieten sich endliche Automaten an, die Zugang zu den Verfahren der Automatentheorie öffnen. Auf diesem Wege können alle Komponenten eines Computers (meist in vereinfachter Form) systematisch entwickelt werden: Zähler, Register, Rechenschaltungen, ...*
4. *Komplexere Systeme können im Automatenmodell nicht mehr übersichtlich beschrieben werden; deshalb bietet sich ab dieser Stufe modulares Arbeiten an. In Blockschaltbildern werden einfache programmierbare Systeme entwickelt, bei denen der Aufbau der wesentlichen Einzelkomponenten bekannt ist. Pseudoassemblerbefehle, die aus den Mikroprogrammschritten dieser Schaltungen abgeleitet werden, bilden den Übergang zu höheren Programmiersprachen.*
5. *Die Architektur dieser Schaltungen kann leicht zu der bekannter Systeme erweitert werden. Die Stärken und Schwächen unterschiedlicher Strukturen werden aus deren Funktionen abgeleitet. Erst der Vergleich sehr unterschiedlicher Architekturen lässt solche Vergleiche zu.*

Die Zahl der im Unterricht bearbeitbaren „Stufen“ hängt natürlich von der zur Verfügung stehenden Zeit ab. Weiterhin soll die Reihenfolge nicht zeitlich verstanden werden: Ich selbst hangele mich meist (sehr schnell) in einer Art Top-Down-Verfahren von einfachen Anweisungen der benutzten Programmiersprache über Pseudoassembler-Befehlssequenzen und die Blockschaltbilder eines Rechners bis zu einzelnen Hardwarekomponenten herab, so dass dann anschließend (ausführlich) entsprechende Schaltungen entworfen und im Bottom-Up-Verfahren zu programmierbaren Systemen zusammengesetzt werden können. Damit lässt sich problemlos ein ganzes Halbjahr verbringen; aber auch wenn wesentlich weniger Zeit zur Verfügung steht, muss auf den Entwurf (einfacher) programmierbarer Schaltwerke nicht verzichtet werden.

Ein Kurs zur technischen Informatik hat damit mehrere sehr unterschiedliche Funktionen:

- Er gestattet schon auf niedrigem Niveau praktisches Arbeiten mit „echter“ Technik[406] – eine Erfahrung, die im Gymnasium sonst kaum geboten wird.
- Er liefert ein solide fundiertes hardwarenahes mentales Computermodell, das durch praktische Arbeiten auf unterschiedlichen Gebieten und sehr unterschiedlichem Niveau gefestigt wird. Damit stützt er das Verständnis z. B. für die Arbeit mit Variablen, Parametern, Rekursionen, Blockstrukturen, … im Softwarebereich hilfreich ab.
- Er bietet Erfahrungen im Entwurf und der Anwendung endlicher Automaten,
- und er macht allen viel Spaß!

[405] Schon um Probleme der Schülerinnen und Schüler mit dem Physikunterricht nicht auf die Informatik zu übertragen!
[406] Hoffentlich „ausgelagert“ in die Sek. I, denn dort gehört diese Erfahrung hin.

3.5 Zum Unterrichtsgang[407]

3.5.1 Zur Anwendung der Kriterien für Unterrichtsinhalte

Die Auswahl der Inhalte eines Theoriekurses muss anhand der gefundenen *Kriterien für Unterrichtsinhalte* erfolgen. Dementsprechend können sie nicht nur aus fachimmanenten Überlegungen abgeleitet werden, sondern müssen auch den allgemein bildenden Aspekten gerecht werden. Gehen wir sie also einzeln durch:

Zur formalen Bildung:

Schon bedingt durch die starke Betonung der fundamentalen Ideen, die sich im Ideenbaum der *Formalisierung* finden, kann dieser Aspekt leicht erfüllt werden. Unterrichtsinhalte sollten den *Prozess der Begriffs- und Modellbildung* betonen, in dem ausgehend von konkreten Beispielen zielgerichtet abstrahiert wird, um übergreifende Eigenschaften zu finden und hantierbar zu machen.[408] Die *Anwendung formaler Verfahren* und der *Wechsel zwischen Modellen*, etwa beim Entwurf, die enge *Verknüpfung von Theorie und Praxis* und die Rückkoppelungen zwischen diesen müssen deutlich werden.[409] An wenigen, aber geeigneten Stellen muss im Theoriekurs die *herausragende Stellung der Mathematik* hilfreich in Erscheinung treten.

Zu den fundamentalen Ideen:

Aspekte der Algorithmisierung müssen in den *Entwurfsverfahren* und bei den *Anwendungen* (Simulationen, Realisierung von Modellen, …) herausgearbeitet werden. Modularisierung und Hierarchisierung sind fast allen Aufgabenfeldern aus diesem Gebiet immanent.[410] Die Ideen des Formalisierungs-Baums müssen deutlich zu Tage treten – aber das ist in diesem Bereich auch kaum zu vermeiden.

Zu den Schlüsselproblemen:

Hier muss der Bezug einerseits über die *Auswahl der Beispiele* erfolgen (Anwendungen der Codierung, Anwendungen und Grenzen der formalen Sprachen, Komplexität von zellulären Automaten, …), andererseits halte ich den *Weg zu Grenzfragen* im Bereich der Berechen- und Entscheidbarkeit für so wichtig, dass er auch inhaltlich den Kurs rechtfertigt. Verdeutlicht er doch sowohl exemplarisch wissenschaftliches Arbeiten wie die *Mächtigkeit der mathematischen Methoden* auf einem (für Schülerinnen und Schüler) ganz anderem Gebiet als üblich.[411]

Zum Transfer:

Hier gilt das Gleiche wie immer: Es dürfen nicht nur einzelne Beispiele „besprochen“ werden, sondern die entwickelten Methoden müssen in einem Themenfeld *selbstständig auf modifizierte Beispiele angewandt* werden und so Werkzeugcharakter gewinnen.[412] Im Bereich Technik/Theorie bieten sich für sehr ähnliche Verfahren drastisch *unterschiedliche Anwendungsbereiche*, so dass die Übertragbarkeit besonders deutlich wird. Diese Möglichkeit muss genutzt werden.

---

[407] ab hier weitgehend nach [Mod92a] S. 354ff

[408] [Koe00] *„Die Integration der beiden Aspekte* [Algorithmik/gesellschaftskundlicher Bereich] *wird durch Modellbildung geleistet. Modellbildung wird dabei verstanden als Konstruktion eines zweckmäßigen formalen Systems, das der Darstellung und der Lösung einer realen Problemsituation dient.“*

[409] [Schu99]: *„ … Dazu gehören in stärkerem Maße theoretische Grundlagen für praktisches Handeln.“*

[410] Wenn man sich nicht nur auf einzelne „ausgewählte“ Beispiele beschränkt.

[411] [Sche97] *„Computer liefern anschauliche Darstellungen höchst abstrakter mathematischer Modelle. (...) unterstützen nicht-lineares Darstellen und Zugreifen, vernetztes Denken. (...) ermöglichen entdeckendes Lernen auf eigenen Pfaden.“*

[412] [Bus87] S. 10: *„Die angestrebte Orientierung lässt sich nicht erreichen, wenn die Schüler allein mit den fertigen Resultaten neuzeitlicher Wissenschaften konfrontiert werden.“*

Zur Modellbildung:

Die Erzeugung gültiger Modelle für Informatiksysteme ist eine der Hauptaufgaben des Technik-/Theorieteils der Kursfolge. Der Unterricht muss deshalb aus den vorhergehenden Kursen vorliegende Erfahrungen ordnen und daraus jeweils ein stark reduziertes, aber valides *Hardwaremodell* entwickeln, um sie „nach unten" abzusichern, und ein *Softwaremodell*, das einerseits den Algorithmusbegriff präzisiert und auf seine Grenzen abklopft, anderseits die Grundfunktionen erklärt.

Zum Projektunterricht:

Die Anwendungsmöglichkeiten der Modelle des Theorie-/Technikkurses sind so vielfältig, dass es schon fast fahrlässig wäre, auf projektartige Phasen zu verzichten.[413] Einerseits können die *gleichen Methoden auf unterschiedliche Aufgabenstellungen* in Gruppen angewandt, dokumentiert und vorgestellt werden (jeweils in den Bereichen Steuern und Regeln, „Taschenrechner", programmierbare Schaltungen, einfache Computersprachen, generative Grammatiken, Robotersteuerung, Kryptografie, spezielle Turingmaschinen, …), andererseits können auch *unterschiedliche* (und damit unterschiedlich anspruchsvolle[414]) *Methoden bei ähnlichen Aufgabenstellungen* Anwendung finden (Entwicklung und Parsen einer Sprache, Entwicklung einer Schaltung für bestimmte Aufgaben, …). Im zweiten Teil der Kursfolge haben die Lernenden sowohl die Kenntnisse wie die Erfahrungen, um solche Aufgaben selbstständig anzugehen. Im konstruktivistischen Sinne kann und muss von ihnen auch erwartet werde, aktiv an ihren Lernfortschritten zu arbeiten. Wo anders als in Projektphasen können sie diesen Anspruch vorrangig erfüllen?[415]

Zum offenen Kanon:

Die technische und theoretische Informatik sind m. E. die Musterbeispiele für Themenbereiche, an denen sich dieses Kriterium demonstrieren lässt. Inhaltlich ändert sich hier kaum etwas, da in der Schule weder die Theorie noch die Technik an die aktuelle Entwicklung gekoppelt sind. Damit lassen sich die fachlichen Ziele „zeitlos" festschreiben. Es ändern sich aber die Werkzeuge, die den Schulen zur Verfügung stehen. Wollen wir also z. B. die Modularisierung und Hierarchisierung am Beispiel von (Modell-)Rechnern behandeln, dann können wir das (fast zeitlos) mithilfe von TTL-Bausteinen erreichen[416]. Die Simulation der Bauteile kann aber mit aktuellen Werkzeugen erfolgen. Ob hier als OOP-Anwendung selbst programmiert, ein spezielles Simulationsprogramm benutzt oder sogar mit Spreadsheets gearbeitet wird, bleibt der aktuellen Situation überlassen. Entsprechendes lässt sich über die Realisierung der Automatenmodelle sagen. In jedem Fall liegen Inhalte und Ziele fest, die konkrete Umsetzung ist offen.

---

[413] [Sche97] *„Informatikunterricht, der allgemein bildend sein will, muss gerade diesen im Computer enthaltenen Widerspruch zwischen Abstraktem und Konkretem, zwischen streng logisch-rationalem und intuitiv-kreativem Zugang aufgreifen und erklären, die Doppelseitigkeit des Computers zwischen Automat und Medium, zwischen Turingmaschine und Interaktion einsichtig machen."*

[414] [Brun70] S. 78: *„Der Gesichtspunkt, das Streben nach hohen Leistungen solle nicht nur auf den begabten Schüler beschränkt bleiben, ist bereits vorgetragen worden. Aber die Idee, der Unterricht solle sich an den Durchschnittsschüler wenden, um jedem etwas zu bieten, ist eine ebenso inadäquate Formel. Worauf es ankommt, so erscheint es vielen von uns, ist, Lernmaterial bereitzustellen, das dem besseren Schüler Anreiz bietet, ohne das Vertrauen und den Lernwillen der weniger vom Glück Begünstigten zu zerstören. Wir geben uns keinen Illusionen über die Schwierigkeit eines solchen Kurses hin, er ist jedoch der einzige, der uns übrig bleibt, wenn wir hohe Leistungen anstreben und zur selben Zeit der Vielfalt der Talente gerecht werden sollen, die wir auszubilden haben."*

[415] Projekte kann man nicht „abhocken"!

[416] Solange es die noch gibt. Die ersten Bauteile sind schon nicht mehr lieferbar.

3.5.2 Eine mögliche Unterrichtssequenz[417]

Hätten wir nur das Ziel, Aussagen zur Berechenbarkeit zu machen, dann könnten wir uns natürlich auf Turingmaschinen beschränken, weil diese Apparate alles können, was überhaupt von Maschinen ausführbar ist. Allerdings sind sie für viele Zwecke sehr unpraktisch, z. B. weil sie nur über *ein* Arbeitsband verfügen.[418] Stattdessen werden wir gleich unterschiedliche, für unterschiedliche Zwecke optimierte Maschinen (oder Sprachklassen) benutzen. Mit dem Maschinen- oder Sprachtyp klassifizieren wir dann das bearbeitete Problem selbst – und umgekehrt: Die äußere Form z. B. der Regeln einer vereinbarten Sprache zeigt unabhängig von deren Bedeutung, ob diese Sprache analysierbar ist und welcher Typ von Übersetzer diese Analyse liefern kann. Mir scheint es in der Schule nur wenige Punkte zu geben, an denen der Begriff des formalen Systems und die Mächtigkeit dieser Idee derart deutlich wird.[419]

Der Transitionsgraph hat im Bereich der endlichen Automaten eine zentrale Bedeutung. Rechtfertigt seine Benutzung allein deshalb schon den Begriff „theoretische" Informatik? Ein Transitionsgraph ist zuerst einmal eine besondere Art, zustandsabhängige Systeme zu beschreiben: Man notiert sehr übersichtlich, was in den einzelnen Zuständen bei der Eingabe bestimmter Zeichen geschehen soll. Diese Aufgabe ist ziemlich direkt dem Problem vergleichbar, für ein mehrstufiges Zufallsexperiment den Wahrscheinlichkeitsbaum zu zeichnen: eine typische Anforderung des Mathematikunterrichts der Sek. I. Entsprechend sollte das Hilfsmittel der Transitionsgraphen schon früh im Unterricht benutzt werden: bei der Beschreibung von Schaltwerken oder bei der Mustererkennung in der Textverarbeitung. Theorie betreiben wir auf diese Weise aber nicht. Vergleichbares lässt sich über Elemente der formalen Sprachen sagen. Deren Grammatiken machen ausführlich von der Möglichkeit Gebrauch, in Zeichenketten Teile durch andere zu ersetzen. Die Realisierung dieser Fähigkeit und auch deren Anwendung bedeutet aber noch lange nicht, der Theorie der formalen Sprachen näher zu kommen. Wir haben damit ein Problem, das schon mehrfach genannt wurde: Wenn in einem bedeutenden Teilbereich der Hochschulinformatik ein bestimmtes Hilfsmittel (z. B. PROLOG) benutzt wird, dann bringt uns in der Schulinformatik die Benutzung dieses Werkzeuges allein diesem wichtigen Thema noch keinen Schritt näher. Wir wollen also festhalten: *Die Benutzung von Konstrukten der theoretischen Informatik (Transitionsgraphen, Textersetzungsmöglichkeiten, Stapelbegriff, ...) in vorausgehenden Kursen bereitet einen Theoriekurs effizient vor. Sie ersetzt ihn aber nicht.*

Die eigentlichen Aufgaben des Theoriekurses liegen an anderer Stelle: Mithilfe des (möglichst schon bekannten) endlichen Automatenmodells werden Hard- oder Softwaresysteme auf systematische Weise erzeugt. Die „geistige" Arbeit liegt damit alleine in der exakten Beschreibung des Systems. Seine Realisierung kann weitgehend standardisiert erfolgen[420]. Auf diesem Gebiet finden wir eine Fülle von Anwendungsmöglichkeiten für Datenstrukturen (Objekte, Listen, Bäume, Netze, ...), algorithmische Verfahren und Schaltungsentwurf in bunter Mischung.

---

[417] [Dan00] *"The syllabus coverage issue is one of the main discussion topics with regard to project-oriented courses."*

[418] Man kann einwenden, dass auch zwei, drei oder mehr Arbeitsbänder zugelassen sind – doch dann haben wir das Problem zu zeigen, dass alle diese Maschinen äquivalent sind.

[419] [Hop96] *„Das zentrale Argument für Informatik als Bestandteil der Allgemeinbildung: Informatik repräsentiert und ‚transportiert' mehr als jedes andere Fach (...) das historisch und kulturell bedeutsame Bemühen um die Automatisierung geistiger Tätigkeiten (...)"*

[420] Wir haben dann Aufgabenstellungen, die der „Kurvendiskussion" in der Mathematik vergleichbar sind: an manchen Stellen ein unverzichtbares Hilfsmittel des Unterrichts und der Leistungsmessung.

Mithilfe eines konstruktiven Beweises lässt sich sehr einfach die Äquivalenz endlicher Automaten und (z. B.) linksregulärer Grammatiken zeigen. Damit finden wir einen einfachen Zugang zum Übersetzerbau. Sprachkonstrukte, die sich regulär beschreiben lassen, sind deshalb analysierbar, und wir können sogar direkt die Maschine angeben, die „Sätze“ dieser Art analysiert. Auch hier gibt es eine Fülle von einfachen Anwendungen (Analyse einfacher Programmiersprachen-Konstrukte, Definition von Grafiksprachen („LOGO für Arme" – s. Beispiel 4.4), ...) die praktische Arbeit zulassen. Das Zusammenspiel von Scanner, Parser und Compiler/Interpreter ist direkt aufzeigbar.

Kellerautomaten und kontextfreie Grammatiken öffnen den Zugang zur Analyse „echter“ Computersprachen. Der Äquivalenzbeweis ist m. E. zu aufwendig, deshalb lasse ich ihn weg. Auf diesem Gebiet bietet sich die Arbeit mit den formalen Grammatiken an, weil sie einfach „praktischer“ sind. Formelinterpreter, Rekursionen, Parsingtabellen, ... sind geeignete Arbeitsthemen, die arbeitsteiligen Unterricht zulassen. Über unterschiedliche Grammatiken, die die gleiche Sprache beschreiben, lassen sich auf Schulniveau Aussagen über die Effizienz der entsprechenden Algorithmen machen. Unterschiede zwischen der prinzipiellen und der praktischen Berechenbarkeit lassen sich zeigen und erfahren. Die Bedeutung von LR(k)-Grammatiken ebnet einen Weg zu Fragestellungen des Aufwands bei Berechnungen, etwa beim P/NP-Problem. Insgesamt sollte ein gültiges Modell entstehen, das die Funktionsweise von Programmiersprachen und die Arbeit der erzeugten Programme innerhalb des Betriebssystems zureichend erklärt.

Bis hierhin kam die Theorie eigentlich kaum vor. Die Beschäftigung mit den Arbeitsmitteln der theoretischen Informatik ermöglicht es aber, in relativ kurzer Zeit zu zentralen Fragen der Theorie vorzustoßen. Mit Hilfe der Transitionsgraphen, Grammatiken, Funktionen, ... sind Fragestellungen z. B. nach den Grenzen der Algorithmisierbarkeit überhaupt erst formulierbar. Sie können so gefasst werden, dass sie sich in dieser Begriffswelt beantworten lassen. Alleine dieses genügt in meinen Augen schon als „Theorieanteil“ in der Schule. Trotzdem können einige geeignete Beispiele (Halteproblem, Radosche Funktion, ...) aus diesem Gebiet besprochen werden. Gerade die amerikanische Literatur liefert auch für Schulen geeignete hübsche Beispiele auf diesem Gebiet.

Auch in der Schule ist ein schneller Weg zu Fragen der Berechenbarkeit und Entscheidbarkeit zu finden, wenn wir die den Schülerinnen und Schülern mittlerweile vertrauten Begriffe der Automatentheorie benutzen. Wir müssen dabei allerdings gelegentlich das Modell wechseln, und wir müssen einige Kompromisse bzgl. der Exaktheit machen: Wir „beweisen“ also eine Aussage nur für einen Teilbereich (hier: der primitiv berechenbaren Funktionen), und zwar durch Konstruktion, indem wir entsprechende Maschinen „bauen“. Ich meine, dass durch die Einbeziehung eigener Aktivitäten der Schülerinnen und Schüler und vor allem durch die Benutzung der sehr konkreten Maschinenmodelle diesen die Bedeutung der Aussagen viel deutlicher wird als durch passives „Zuhören beim Beweis“.

1. Schritt:
Wir zeigen, dass der intuitiv beschriebene Algorithmusbegriff einer Abbildung innerhalb der natürlichen Zahlen äquivalent ist. (Wechsel von den Algorithmen zu den mathematischen Funktionen)

- Ein Algorithmus ordnet einer endlichen Menge von Eingabezeichen eine endliche Menge von Ausgabezeichen zu.

- Durch Gödelisierung lassen sich sowohl die Eingabedaten als auch die Ausgabedaten als jeweils eine natürliche Zahl auffassen. (Im Primitivfall werden die Zeichenfolgen im ASCII-Code aufgeschrieben. Die so entstehende Folge kann als (sehr große) natürliche Zahl interpretiert werden.)

  ABC → 65 66 67 → 656667

- Folglich kann jedes algorithmische Verfahren als Abbildung einer natürlichen Zahl auf eine andere aufgefasst werden.

2. Schritt:
Es wird gezeigt, dass es nur abzählbar viele Algorithmen, aber überabzählbar viele Abbildungen innerhalb der natürlichen Zahlen gibt.

- Ein Algorithmus muss sich mit endlich vielen Zeichen beschreiben lassen.
- Die Folge dieser Zeichen kann wieder als natürliche Zahl aufgefasst werden (s. o.).
- Damit entspricht jedem Algorithmus eine natürliche Zahl.
- Damit gibt es nur abzählbar viele Algorithmen.
- Mit einer Variante des Cantorschen Diagonalverfahrens zeigt man, dass es überabzählbar viele Abbildungen innerhalb der natürlichen Zahlen gibt.
- Damit ist plausibel gemacht worden, dass es für wenigstens einige dieser Abbildungen keine entsprechenden Algorithmen gibt, die diese Abbildung berechnen.

3. Schritt:
Nach der Churchschen These ist alles, was berechenbar ist, rekursiv berechenbar. Für die Teilmenge der primitiv-berechenbaren Funktionen wird gezeigt, dass Turingmaschinen genau die dazu erforderlichen Operationen ausführen können. „Glaubt“ man dann, dass sie auch die Werte aller rekursiv berechenbaren Funktionen bestimmen können, dann sind die Turingmaschinen die leistungsfähigsten Maschinen überhaupt. (Wechsel von den Funktionen zu den Maschinen)

- Die primitive Berechenbarkeit wird auf Nachfolger-, Projektions- und konstante Funktion sowie ein einfaches Rekursionsverfahren zurückgeführt. Mit Hilfe von Teilprogrammen werden entsprechende rekursive Funktionen realisiert.
- Es werden Turingmaschinen konstruiert, die diese Operationen ausführen können.
- Damit ist ein Algorithmus ein Verfahren, das eine Turingmaschine ausführen kann. Mehr als eine Turingmaschine kann keine Maschine leisten.

4. Schritt:
Anhand wenigstens eines Beispiels wird gezeigt, dass es tatsächlich nicht berechenbare bzw. nicht entscheidbare Probleme gibt. Das Halteproblem, der „fleißige Biber“, ... sind so behandelbar.

Es könnte der Eindruck entstehen, dass hier überwiegend „theoretisch“ gearbeitet wird. Dem ist aber nicht so. Die meiste Zeit gehört der Realisierung der rekursiven Funktionen und dem Bau von Turingmaschinen für unterschiedlichste Zwecke, also deren Anwendung auf Probleme, die nicht direkt zur Argumentationskette gehören. Die verbindenden Glieder sind auf wenige Stunden zu beschränken. Sie dienen aber dazu, den Schülerinnen und Schülern zu zeigen, in welchem Kontext sie arbeiten, so dass das reine (und sehr beliebte) „Herumspielen“ mit Turingmaschinen den Sinn dieser Arbeit nicht verdeckt.

Im Anschluss an diese Phase können (ich meine: müssen) populärwissenschaftliche Texte zu Themen gelesen und besprochen werden, die mit dem Gödelschen Unvollständigkeitssatz zu tun haben – und von denen gibt es reichlich auch gute[421]. Die Lernenden wissen jetzt, wovon dort geredet wird, weil sie auf diesem Gebiet selbst gearbeitet haben; und ich denke, dass dieses sehr viel wichtiger ist als z. B. die Fähigkeit, den Beweis zum Halteproblem wiedergeben zu können. Die fehlende Fähigkeit formaler Systeme, die Aussagen „über sich selbst“ machen können (wozu z. B. die Arithmetik durch Gödels „Trick“ mit der Abbildung von Algorithmen auf Zahlen gehört), die Wahrheit jeder dieser Aussagen zu beweisen, führt auf zentrale Fragen der Erkenntnistheorie und der künstlichen Intelligenz. Es sollte gezeigt werden, dass kein System rein formal arbeiten kann, sondern immer auf einem Satz als wahr erkannter Elementaraussagen fußt. Die enorme Mächtigkeit der dann daraus ableitbaren Aussagen muss gewürdigt werden, aber ebenso muss deutlich werden, dass nicht alle Aussagen ableitbar sind. Formale Systeme beruhen deshalb auf menschlicher Einsicht, und die Einsicht geht über sie hinaus, weil es Aussagen gibt, die als wahr erkannt, aber nicht formal als wahr bewiesen werden können[422]. Der Gödelsche Satz spielt m. E. im Bereich der Allgemeinbildung eine ebenso zentrale Rolle wie die Heisenbergsche Unschärferelation, denn er macht prinzipielle Aussagen über die Fähigkeit, die Welt zu erkennen.

Theoretische Informatik beschäftigt sich in diesem Sinne auch mit Computern, sie beschäftigt sich aber vor allem mit Menschen, ihrer Suche nach Wahrheit und dem gescheiterten Versuch, wenigstens im Bereich der Mathematik vollständige Wahrheit zu finden.

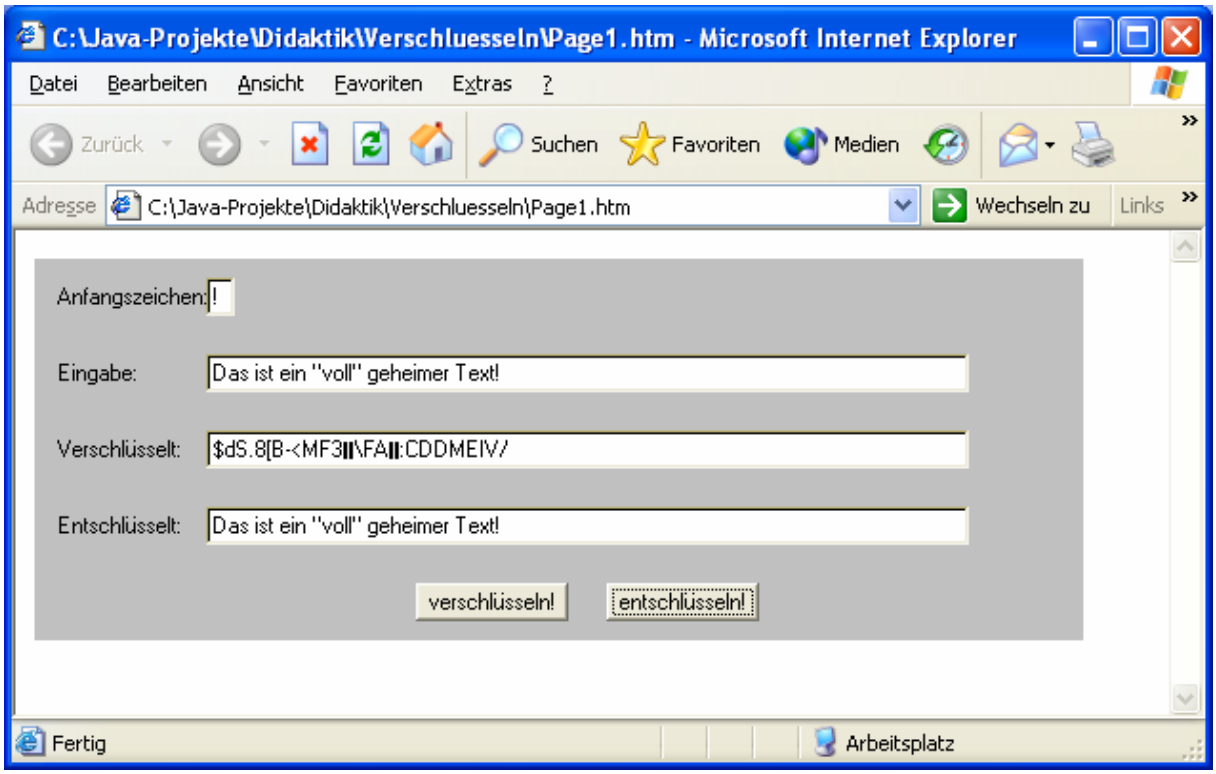
[421] Interessant sind immer noch [Hof79] und [Hof91]
[422] s. dazu [Pen91]

4. Beispiele

4.1 Beispiel 1: Zur Präfiguration der Zustands-Idee

Sollen fundamentale Ideen präfiguriert werden, dann muss das früh geschehen. Entsprechende Einschübe sollten immer dann erfolgen, wenn sie sich ohne Zwang in die üblichen Unterrichtsgänge integrieren lassen. Als Beispiel dafür sollen die Ideen des *Zustands* und des *Zustandsübergangs* schon im Anfangsunterricht vorbereitend eingeführt werden. Geeignet dafür ist z. B. eine Unterrichtseinheit über kryptografische Verfahren, die innerhalb des Themas *„Bezahlen im Internet“* oder *„Geschichte der Datenverarbeitung“*[423] auftauchen kann. Je jünger die Teilnehmerinnen und Teilnehmer sind, desto mehr interessieren sie sich für dieses Thema. Deshalb, und weil die Verfahren einfach sind, gehört es in den Informatikunterricht der Klassen 9/10, notfalls in den Anfangsunterricht der Sek. II.

Wenn Texte verschlüsselt werden, dann geschieht das oft zeichenweise, indem die Zeichenkette mithilfe von Zählschleifen durchlaufen wird. Das Thema macht also mit typischen Elementen des Anfangsunterrichts vertraut und enthält eine Fülle von Veränderungsmöglichkeiten, die auf sehr unterschiedlichem Niveau behandelt werden können. Es eignet sich damit sehr gut für Einzelarbeit bzw. projektartige Phasen. In diesem Beispiel soll experimentierend auf verschiedene elementare Datentypen eingegangen werden, auch die binäre Darstellung von Zahlen und Zeichen wird thematisiert. Als Sprache wählen wir *Java* und als Ziel ein *Applet*, weil Schülerinnen und Schüler der anvisierten Altersstufe gerne ihre Ergebnisse ins Netz stellen. Java-Programmiereinheiten lassen sich so gut in Kurse über Informationsgewinnung und -darstellung integrieren.



[423] Zahlreiche Beispiele finden sich z. B. in [Kip98], Unterrichtsgänge in [Mod96a]

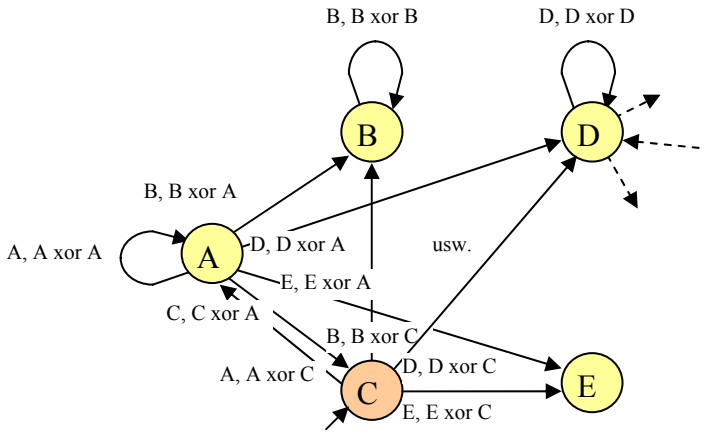
Ein mögliches Codierungsverfahren läuft so:

z ← ein frei gewähltes Startzeichen
ergebnis ← “”
mit allen Zeichen der zu verschlüsselnden Zeichenfolge tue
v ← das nächste Zeichen, verschlüsselt mithilfe von z
ergebnis ← ergebnis + v
z ← das zuletzt verschlüsselte Zeichen

Das Verschlüsselungsverfahren verschlüsselt also nicht alle Zeichen auf die gleiche Art, sondern es hängt von seinem ***Zustand*** ab, welches Ergebnis produziert wird. Ausgehend von einem ***Anfangszustand*** wird das System in Folgezustände ***überführt***, die sich aus der zu verschlüsselnden Zeichenfolge ergeben. (Ob das Verfahren besonders intelligent ist, wird sich in weiteren Tests ergeben.)

Solch ein kompliziertes Verfahren will erst einmal gut verstanden sein. Hilfreich ist deshalb ein ***Zustandsgraph***, der die Wechsel zwischen den Zuständen überschaubar darstellt. Vor allem aber kann dieser Graph im Gespräch an der Tafel ***konstruiert*** werden, wobei Unklarheiten und Fehlinterpretationen deutlich werden können und zu beseitigen sind. Da das Verfahren wiederholt angewandt werden muss, kann die Konstruktion des Graphen so weit getrieben werden, bis die Arbeitsweise von allen verstanden wurde. ***Ausgabe-*** und ***Überführungsfunktion*** sollten klar getrennt werden.

Beschreiben wir den Zustand einfach durch das „aktuelle“ Zeichen, dann können wir z. B. das nächste Zeichen des Klartextes mit diesem xor-verschlüsseln. (Auch das will geübt sein! Im Programm werden die Zeichencodes deshalb so verschoben, dass das ‚A’ der Zahl ‚0’ zugeordnet wird usw. Dadurch wird das Nachvollziehen der XOR-Verschlüsselung „per Hand“ erleichtert.) Der Folgezustand ergibt sich im einfachsten Fall aus dem gerade verschlüsselten Zeichen. Ein Ausschnitt[424] des entstehenden Graphen könnte dann wie folgt aussehen (der beliebig gewählte Anfangszustand ist farblich hervorgehoben):



[424] Beschränkt man sich zuerst auf nur wenige Eingabezeichen, dann kann der Graph auch vollständig aufgezeichnet werden. Vor allem aber werden die Zeichencodes kurz, so dass die XOR-Verschlüsselung einfach ist.

Das Verfahren sollte, je nach Kenntnisstand der Lernenden, entweder teilweise oder vollständig, gemeinsam oder in Partnerarbeit in der gewählten Sprache programmiert werden. Danach gibt es sehr viel für die Lernenden selbstständig zu tun:

- Arbeitet die Codierung in allen Fällen korrekt? Gibt es Sonderfälle, z. B. dass bestimmte Zeichenfolgen gar nicht (kaum, zu einfach, ...) verschlüsselt werden?
- Ist die Codierung umkehrbar? Das Entschlüsselungsverfahren sollte dann auch realisiert werden. Wettbewerbe im Ver-/Entschlüsseln zwischen verschiedenen Gruppen sind möglich. (Das Verfahren ist einfach genug, um schnell zu Entschlüsseln.)
- Die Überführungsfunktion kann verändert werde. Der Folgezustand könnte vom alten Zustand und dem nächsten Zeichen abhängen, oder nur vom alten Zustand, indem eine Zeichenfolge zirkulär durchlaufen wird[425], oder sonstwie.
- Die Ausgabefunktion kann verändert werden, indem andere Verschlüsselungsmethoden angewandt werden, die auf zwei Zeichen beruhen, auf Verschiebungen, ...
- Es können mehrstufige Verfahren realisiert werden.
- ...

Die Einführung des Zustandsbegriffs und einiger Formalien akzentuiert die Arbeit etwas – aber nur leicht – anders als üblich. Sie schafft dabei durch die klare Unterscheidung der einzelnen Aktionen Eingriffsmöglichkeiten und so Ansatzpunkte für eigene Veränderungen, deren Schwierigkeitsgrad dem Anfangsunterricht angemessen ist. Sie schafft Freiräume in einem einheitlichen Begriffsfeld und erleichtert so die Kommunikation zwischen den Gruppen, die klar benennen können, an welchen „Schrauben sie gedreht“ haben.

Haben wir als Lerngruppe z. B. eine 11. Klasse, üblicherweise mit sehr unterschiedlichen Vorkenntnissen und Interessen bei den Unterrichteten, die sich zur Einführung einige Wochen mit Problemen der Computergrafik beschäftigt haben[426], wobei elementare Datentypen (*ganze Zahlen*, *boolesche Größen*, ggf. *Zeichen*) und die algorithmischen Grundstrukturen *Sequenz*, *Verzweigung* und ggf. *Iteration*[427] benötigt wurden, dann kann man sich danach z. B. mit Sicherheitsfragen im Internet beschäftigen („elektronisches Geld“, „Softwarepiraterie“, …), wobei das „Verschlüsseln“ als Teilthema auftaucht. Da Komponenten (*Buttons*, *Editierfelder*, …) mit ihren Eventhandlern von Anfang an benutzt werden sollten, kann man sich jetzt völlig auf die Zeichenkettenverarbeitung konzentrieren, wobei anfangs die Strings den benutzten Komponenten entnommen und ihnen zum Schluss wieder übergeben werden. Weil die Ereignisbehandlungsmethoden als Prozeduren realisiert sind, kann auch die Zeichenkettenverarbeitung in Unterprogramme verlagert werden[428].

Ziel des Unterrichts ist es neben der Einführung einiger fachlicher Grundlagen, die Schülerinnen und Schüler, ausgehend von einem gelösten Problem, langsam an selbstständiges Arbeiten zu gewöhnen.

---

[425] Das wäre dann das übliche Verfahren mit „Schlüsselwort“.
[426] Das Thema lässt sehr gut Binnendifferenzierung zu.
[427] Statt Schleifen hat man aber wahrscheinlich Timer-Komponenten eingesetzt.
[428] Damit unterscheiden sich die Lösungen der jetzt hinzukommenden Teilaufgaben der Zeichenkettenmanipulation kaum noch von denen, die z. B. vor 20 Jahren benutzt wurden.

Unter diesen Voraussetzungen benötigen wir eine Unterrichtsstunde, um anhand eines sehr einfachen Verfahrens gemeinsam das Grundgerüst eines Verschlüsselungsprogramms zu entwerfen. Die nächsten vier Stunden stehen für Gruppenarbeiten bereit, in denen die Schülerinnen und Schüler das Gerüst verändern und erweitern bzw. durch bessere Verfahren ersetzen. Das ist auf sehr unterschiedlichem Niveau möglich (s. o.), so dass auf die unterschiedlichen Vorkenntnisse eingegangen werden kann. Die Unterrichtenden sollten in diesen Stunden anhand von Transitionsgraphen unterschiedliche Verschlüsselungsverfahren mit den Kleingruppen entwerfen und diese diskutieren lassen.

Mit diesen – relativ knappen – Informationen lässt sich die Unterrichtseinheit entsprechend den Vorschlägen in Kapitel 2.5 klassifizieren:

Thema: Kodierungsverfahren

Zeitbedarf: ab 5 WStd.

Voraussetzungen: elementare Datentypen, Zeichenketten, Ein- und Ausgabe über Komponenten in Applets, Methoden, ggf. XOR-Verschlüsselung

Die Einheit dient der Verdeutlichung

der kulturellen Bedeutung des Themas zu 5 %.
gesellschaftlicher Auswirkungen des Themas zu 15 %.
rein fachlicher Aspekte zu 80 %.

Die folgenden Unterrichtsmethoden erfordern an Unterrichtszeit ca.

Lehrervortrag:
Unterrichtsgespräch: 20 %.
Partnerarbeit: 40 %.
Einzelarbeit: 40 %.
Projektarbeit:

Das Thema verdeutlicht die folgenden fundamentalen Ideen:

1. Algorithmisierung
   - 1.1 Entwurfsparadigmen (Branch and Bound, Backtracking, ...)
   - 1.2 Programmierkonzepte (Alternative, Iteration, Rekursion, ...) zu 60 %
   - 1.3 Ablauf (Prozess, Nebenläufigkeit, ...):
   - 1.4 Evaluation (Verifikation, Komplexität, ...):

2. strukturierte Zerlegung
   - 2.1 Modularisierung (Methoden, Hilfsmittel, ...) zu 20 %
   - 2.2 Hierarchisierung (Darstellung, Realisierung, ...)
   - 2.3 Orthogonalisierung (Emulation, ...)

3. Formalisierung
   - 3.1 formale Sprache (Syntax, Semantik, ...)
   - 3.2 Automat (Zustand, Übergang, Vernetzung, ...) zu 20%
   - 3.3 Berechenbarkeit (Grenzen, Durchführbarkeit, ...)

Der Java-Quelltext findet sich bei den Materialien am Ende der Arbeit.

4.2 Beispiel 2: Rechnermodelle

Fasst man „Programmiertwerden“ als eine Form von „Maschinenlernen“ auf, dann müssen Maschinen über die Fähigkeit verfügen, auf unterschiedliche Situationen unterschiedlich zu reagieren, also Entscheidungen zwischen Alternativen zu fällen. Da es Aufgabe von Unterrichtseinheiten über technische Informatik in ihrer Gesamtheit ist, ein gültiges Maschinenmodell bei den Lernenden entstehen zu lassen, gehört m. E. der Übergang von den elementaren Grundschaltungen zu eben dieser Fähigkeit zu den wichtigen Elementen eines Technikkurses. Darauf aufbauend kann eines der üblichen Rechenwerke, das über mindestens zwei unterschiedliche ***Zustände*** verfügt (z. B: Addieren und Subtrahieren) gesteuert werden. Die Belegung der Steuerleitungen bildet dann einen Elementarbefehl. Fügt man mehrere Elementarbefehle zu einer ***Sequenz*** zusammen und speichert sie geeignet, dann hat man schon die rudimentäre Form eines Programms gefunden. Kann der Wert des erforderlichen Programmschrittzählers mithilfe der „Entscheidungsschaltung“ beeinflusst werden, dann können wir ***Alternativen*** und ***Sprünge*** und somit ***Iterationen*** programmieren. Das genügt als Maschinenmodell für ein Grundverständnis programmierbarer Automaten. Wenn man die Baugruppen der Schaltung zu funktionalen Einheiten wie Rechenwerk, Steuerwerk, Speicher, … zusammenfasst, dann hat man ein schönes Beispiel für ***Modularisierung*** und ***Hierarchisierung*** gefunden. Aber auch Erfahrungen mit der zunehmenden Integration der TTL-ICs dienen demselben Zweck. Die angegebenen fundamentalen Ideen treten hier in einem ganz anderen Gewand auf als in den anderen Kursen. Sie verdeutlichen damit deren übergreifenden Charakter.

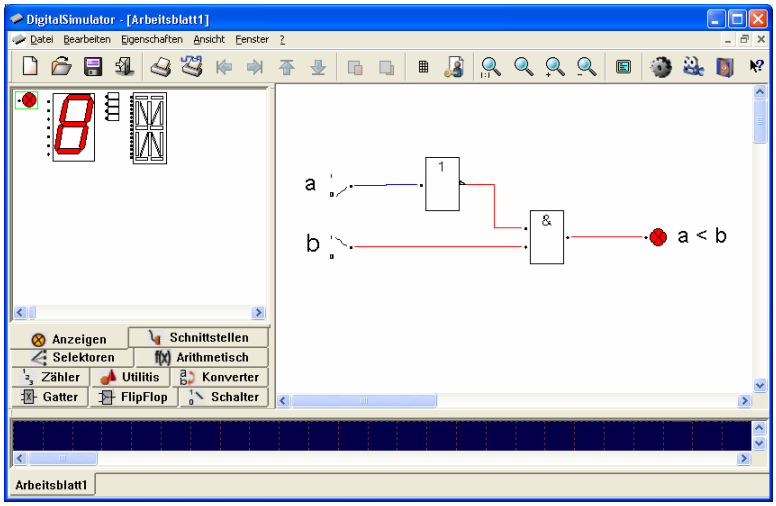
Schülerinnen und Schüler der Sekundarstufe I mögen den Umgang mit echter Hardware (TTL-ICs, …) – meist nach anfänglichem Sträuben[429] – sehr. Sie stellen der Schulgemeinschaft mit großem Stolz die entwickelten „Rechenwerke“ z. B. in Schaukästen vor. Da die Entwicklung von Schaltungen und die Komposition von einfachen Schaltwerken aus vorhandenen ICs auch vom Anspruch her in diese Altergruppe gehört, sollten zumindest die Anfänge der technischen Informatik in diesen Bereich verlegt werden. Nicht zu unterschätzen sind die Erfahrungen mit echter Technik, die sonst in der Schule kaum vorkommen.

Beginnen wir mit den Alternativen. Da sich TTL-ICs in einem Buch schlecht unterbringen lassen, benutze ich stattdessen ein frei verfügbares Simulationsprogramm[430]. Reduzieren wir die Fragestellung so weit wie nur möglich, dann lautet die Aufgabe:

„Gesucht ist eine Schaltung, die entscheiden kann, ob eine 1-Bit-Zahl kleiner als eine andere ist.“

Das Problem kann leicht über eine Schaltwerttabelle gelöst werden. Wir erhalten z. B. die folgende Schaltung und ihre Gatterdarstellung:

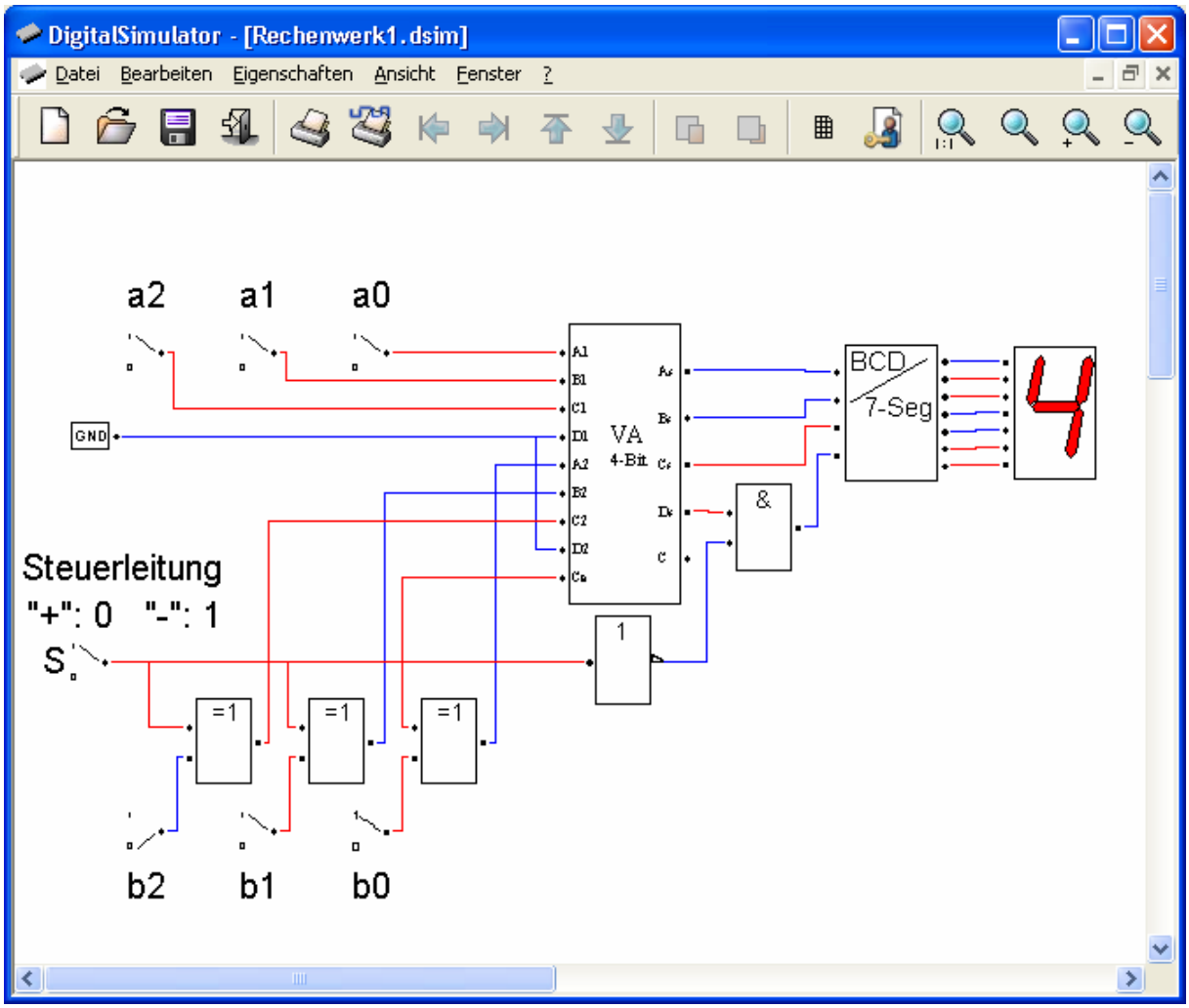
$$a < b = \overline{a} \wedge b$$



[429] Das Thema könnte ja etwas mit Physik zu tun haben, und die ist leider sehr unbeliebt.
[430] Den Digitalsimulator 4.0 von A. Herz

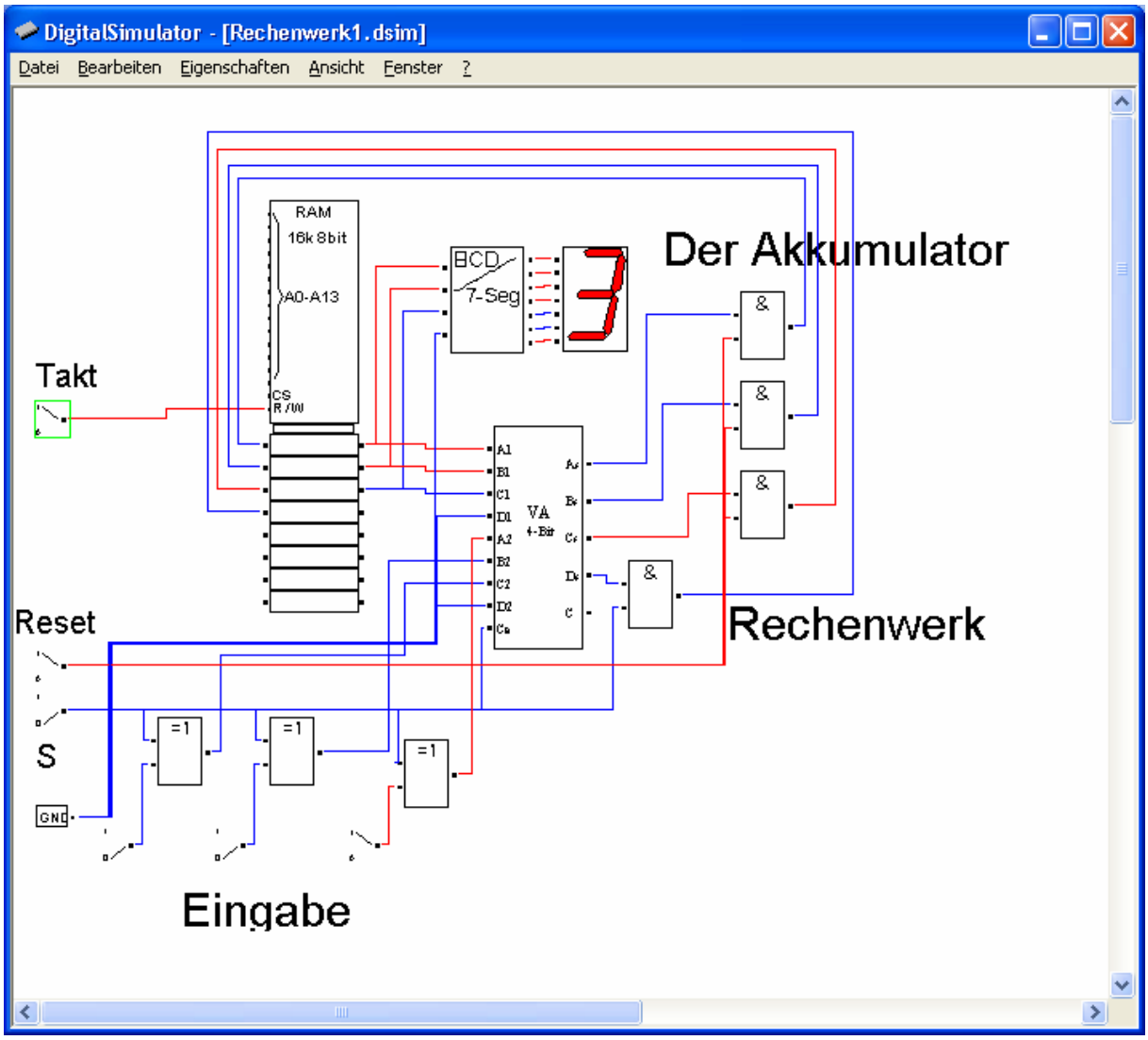
Betrachten wir jetzt eine Sequenz von Befehlen, die durchnummeriert sind, dann kann nach einem Vergleich dessen Ergebnis zum nächsten Programmschritt addiert werden. Schlägt der Vergleich fehl, dann wird der nächste Befehl ausgeführt, sonst der übernächste.[431] Folgen auf einen Vergleichsbefehl zwei Sprungbefehle, dann stellen die angesprungenen Befehlssequenzen die gesuchten Alternativen dar.[432]

Wir benötigen als Ausgangspunkt ein Rechenwerk, das wenigstens zwei Funktionen ausführen kann. Ich wähle einen umschaltbaren 3-Bit-Addierer/Subtrahierer.



In diesem ersetzen wir eine Schalterreihe durch ein Register[433], das Zahlen speichern und Zwischenergebnisse aufnehmen kann - dann haben wir schon mal einen Akkumulator. Da das hier benutzte Simulationsprogramm leider in der derzeitigen Version kein Register enthält, missbrauche ich einfach das 16k x 8Bit-RAM dafür. Das ist zwar nicht schön, vereinfacht aber die Schaltung drastisch. Weiterhin führen wir einen Takt(-Schalter) ein, da der Speicher einen solchen benötigt, und einen Reset-Schalter, um den Akkumulator bei Bedarf zu löschen. Damit haben wir zwei Steuerleitungen (und den Takt), um die Funktionalität der Schaltung zu beeinflussen.

[431] So arbeiteten z. B. die ersten programmierbaren Taschenrechner von HP.
[432] Mit diesem einfachen Verfahren können wir schon **alle** algorithmischen Verfahren realisieren – wenn auch nicht besonders elegant.
[433] z. B. IC 74194



Wollen wir mit der Schaltung eine Rechenaufgabe lösen (deren Ergebnisse allerdings im 3-Bit-Rechenbereich bleiben müssen), dann können wir diese durch eine Folge von Steuerleitungsbelegungen und die erforderlichen „Daten“ (Zahlen) „programmieren“. Wählen wir z. B.

3 + 4 – 5 + 1 =

dann brauchen wir das folgende Programm:

Reset	S	Eingabe	Kommentar
0	bel.	bel.	Akku löschen
1	0	011	3 laden
1	0	100	4 addieren
1	1	101	5 subtrahieren
1	0	001	1 addieren

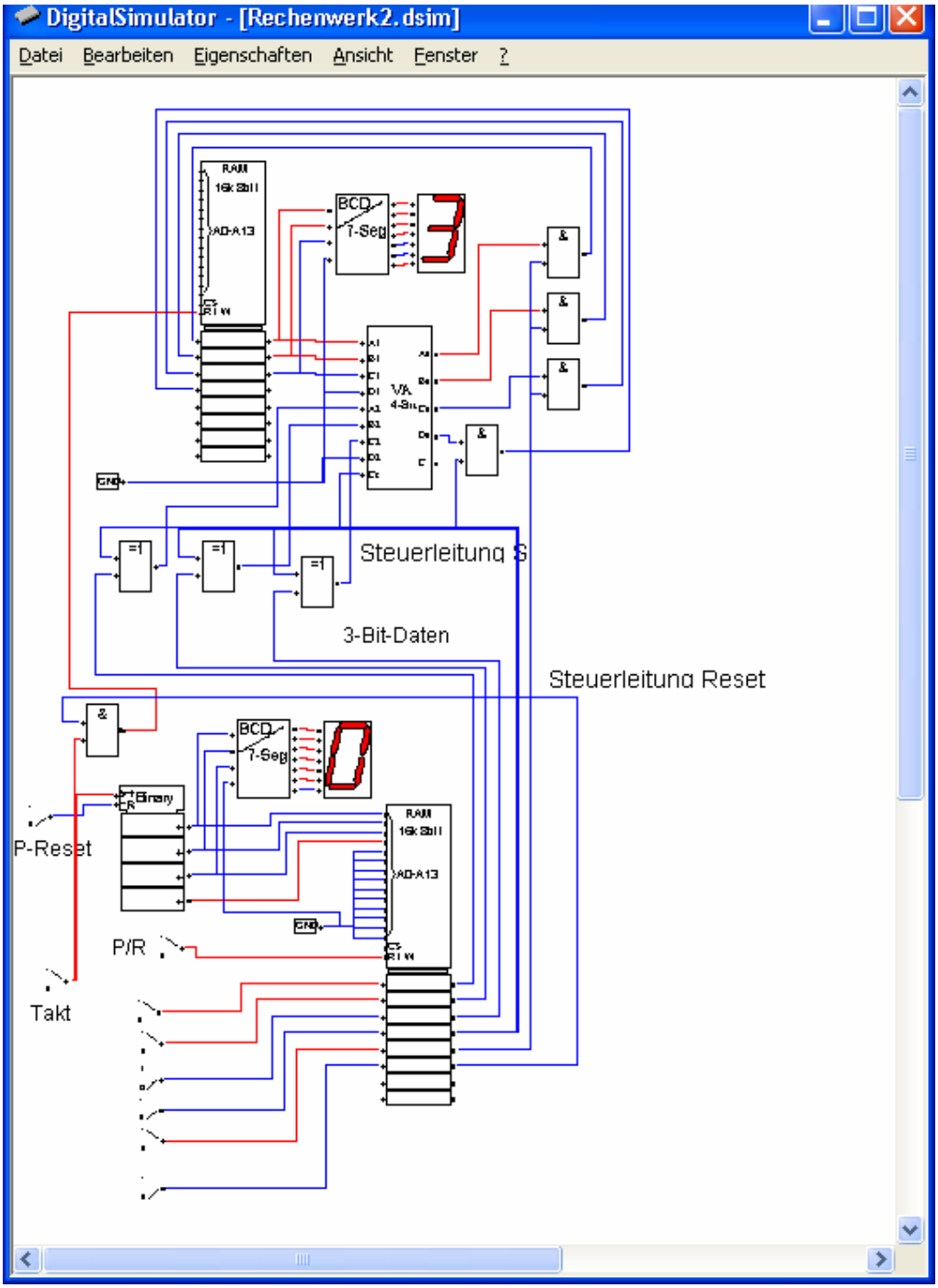
Wir können die Bitfolge als „Maschinenprogramm“ auffassen. Zwischen den Befehlen folgen „Takte“.

00000
10011
10100
11101
10001

Bis zu dieser Ebene ist auch in der Sek. I mühelos vorzudringen. Die Schülerinnen und Schüler können mit der Schaltung experimentieren, lernen vergleichend mit Simulationsprogrammen und echter Hardware umzugehen, können die Schaltung abwandeln

- durch andere Bauteile (hier: „richtige“ Register benutzen, Anzeigen, …),
- durch andere Steuerleitungen (z. B. um den Akkumulators zu laden),
- durch weitere Register (z. B. als Speicher für Zwischenergebnisse)
- oder durch Erweiterung des Zahlenbereichs.

Kurz: Sie machen sowohl auf einer relativ abstrakten wie auf einer konkreten Ebene Erfahrungen im Umgang mit echter Technik, lernen abzuschätzen, ob ihnen diese Art des Vorgehens liegt.



Betrachten wir das oben abgebildete Rechenwerk als ein Modul, in das drei Datenleitungen, zwei Steuerleitungen und eine Taktleitung hereinführen, dann können wir dieses von außen steuern, indem wir die erforderlichen Befehle in einem Speicher ablegen. Dieser selbst muss natürlich auch geordnet arbeiten, also „gesteuert" werden. Dazu benötigen wir einen Binärzähler als *Program-Counter*, der den aktuellen Befehl angibt, und eine Möglichkeit, die Programminhalte einzugeben. Ich realisiere das hier über geeignete Schalter im unteren Teil des Bildes.

Leider kann unser Binärzähler (im Gegensatz zu vielen „echten"[434]) keine Werte laden. Deshalb ist in dieser Schaltung nur ein „Sprung zur Adresse 0" möglich – über Reset. (Ich habe hier stattdessen einfach die Taktleitung zum Rechenwerk unterbrochen, um einen Stopp zu realisieren.) Trotzdem tauchen schon die ersten Elemente eines echten Prozessors auf. Das System ist – bei einem geeigneten Zähler – leicht zu erweitern. Es können von den Schülerinnen und Schülern Maschinen erfunden werden, die natürlich nur immer einige Ausschnitte eines richtigen Befehlssatzes „verstehen", trotzdem aber z. B. in der Lage sind, rekursive Programme auszuführen.

Die programmierbaren Schaltwerke erfordern einiges an analytischem Denken und sorgfältige Abstimmung der Schaltungsmodule. Sie sind von der Komplexität her in der Sek. II anzusiedeln, können da aber durchaus auch im Grundkurs entwickelt werden. Sie liefern ein stark reduziertes, aber im Prinzip richtiges Hardwaremodell des Von-Neumann-Computers, das für das Verständnis der Abläufe bei Unterprogrammaufrufen und deren Parameterübergabe, Rekursionen und Referenzen m. E. unerlässlich ist. Und sie liefern auch in dieser Altersstufe Erfahrungen in technik-orientierter Arbeit.

Haben wir als Lerngruppe z. B. einen Grundkurs der Stufe 13, dann sollte dieser einerseits solide Kenntnisse über Entwurfstechniken und Programmierung besitzen, andererseits an Gruppen- und selbstständige Einzelarbeit gewohnt sein. Weil die Simulation von TTL-Bauteilen in der Schule eines der besten Beispiele für den Einsatz von OOP-Techniken ist, kann kursbegleitend ein eigener Hardwaresimulator als Hausarbeit entwickelt werden, wobei der im Unterricht neben der echten Hardware eingesetzte Simulator als Vorlage dient. Die dabei auftauchenden Probleme lassen sich sowohl auf einem elementaren Niveau als auch außerordentlich anspruchsvoll lösen: Es bietet sich ein weites Feld für Binnendifferenzierung.

Ziel des Unterrichts ist es neben der Anwendung fachlicher Grundlagen, bei den Schülerinnen und Schülern ein valides Rechnermodell entstehen zu lassen.

Unter diesen Voraussetzungen wird der Entwurf eines Rechnermodells im zweiten Kursteil liegen, weil anfangs die *logischen Grundschaltungen*, *Rechenschaltungen*, *Speicher* und *Rechenwerke* kennen gelernt werden müssen[435]. Zusammen mit Teilproblemen der Hardwaresimulation (Anwendung von Listen, OOP-Techniken, …) braucht das seine Zeit. Wählen wir jetzt den oben skizzierten Weg, über stufenweise Erweiterungen von einem umschaltbaren Rechenwerk zu einem programmierbaren System zu kommen, dann erfordern die einzelnen Schritte von Stufe zu Stufe nur relativ wenig Zeit für gemeinsame Unterrichtsgespräche, die bei Bedarf eingestreut werden können. Der überwiegende Teil der Unterrichtszeit wird für Einzel- und Gruppenarbeit sowie intensive Einzelgespräche benötigt, in denen auftauchende Probleme geklärt werden - insbesondere, wenn mit echter Hardware gearbeitet wird.

---

[434] z. B. IC 74191

[435] Auf entsprechende Vorkenntnisse aus der Sek. I wird man (noch) kaum zurückgreifen können.

Mit diesen Informationen lässt sich die Unterrichtseinheit entsprechend den Vorschlägen in Kapitel 2.5 klassifizieren:

Thema: Rechnermodelle

Zeitbedarf: ab 10 WStd. (je nach Ziel)

Voraussetzungen: logische Grundschaltungen, einfache Rechenschaltungen, einfache Schaltwerke (Zähler, Speicher, ...)

Die Einheit dient der Verdeutlichung

der kulturellen Bedeutung des Themas	zu 10 %
gesellschaftlicher Auswirkungen des Themas	zu 20 %
rein fachlicher Aspekte	zu 70 %

Die folgenden Unterrichtsmethoden erfordern an Unterrichtszeit ca.

Lehrervortrag:	
Unterrichtsgespräch:	20 %
Partnerarbeit:	40 %
Einzelarbeit:	
Projektarbeit:	40 %

Das Thema verdeutlicht die folgenden fundamentalen Ideen:

1. Algorithmisierung
   - 1.1 Entwurfsparadigmen (Branch and Bound, Backtracking, ...)
   - 1.2 Programmierkonzepte (Alternative, Iteration, Rekursion, ...) zu 20 %
   - 1.3 Ablauf (Prozess, Nebenläufigkeit, ...):
   - 1.4 Evaluation (Verifikation, Komplexität, ...):

2. strukturierte Zerlegung
   - 2.1 Modularisierung (Methoden, Hilfsmittel, ...) zu 20 %
   - 2.2 Hierarchisierung (Darstellung, Realisierung, ...) zu 20 %
   - 2.3 Orthogonalisierung (Emulation, ...)

3. Formalisierung
   - 3.1 formale Sprache (Syntax, Semantik, ...)
   - 3.2 Automat (Zustand, Übergang, Vernetzung, ...) zu 20%
   - 3.3 Berechenbarkeit (Grenzen, Durchführbarkeit, ...) zu 20 %

4.3 Beispiel 3: Gekoppelte Automaten

Das überraschende Verhalten gekoppelter Systeme kann mit *zellulären Automaten* sehr gut demonstriert werden[436]. Einerseits nutzt man hier ziemlich einfache endliche Automaten, so dass die Benutzung der entsprechenden Beschreibungsmittel geübt werden kann, andererseits sind die Resultate so vielfältig, dass die Arbeit trotz dieser Einfachheit alles andere als langweilig ist. Man kann leicht Systeme modellieren, deren Verhalten sich kaum aus dem Simulationsprogramm selbst erschließen lässt. Die Anordnung der Elementarautomaten in Gittern liefert ein exzellentes Beispiel für den Umgang mit zweidimensionalen Feldern. Verteilt man die Funktionalität auf geeignete Objektklassen, dann kann die Darstellung der Ergebnisse standardisiert werden, so dass Experimente an den Automaten ohne großen Programmieraufwand zu realisieren sind. Die Unterrichtseinheit ist je nach der Fortschrittlichkeit der eingesetzten Programmiertechniken im Unterricht der Sek. II anzusetzen. Der erforderliche Zeitbedarf wird wesentlich dadurch bestimmt, ob – und wenn, welche – Teile des Simulationsprogramms vorgegeben werden. In diesem Beispiel benutze ich *Delphi*, weil ich diese Sprache im Unterricht für bestens geeignet halte.

Wir wollen einen zellulären Automaten bauen, der auf dem *Gefangenendilemma* aufbaut, aber etwas abgewandelt auf den *Handel im Internet*. Das Verhalten der Handelspartner wird durch endliche Automaten simuliert, die auf einem in beiden Dimensionen abgeschlossenen Gitter sitzen und innerhalb einer Moore-Nachbarschaft Handel mit den Partnern treiben. Sie tauschen – wie im Internet üblich – Waren gegen Geld[437]. Dabei gibt es unterschiedliche Arten von Geschäftpartnern:

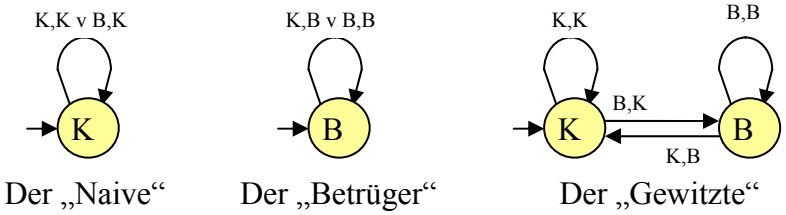
- ***Naive*** kooperieren immer, liefern also den korrekten Gegenwert.
- ***Betrüger*** kooperieren nie.
- ***Gewitzte*** kooperieren anfangs und reagieren danach so, wie der Partner beim letzten Mal.

Auch hier ist die Idee des ***Zustands*** und seiner Wechsel zentral. Des Weiteren spielt aber auch die Idee der ***Berechenbarkeit*** in ihrem prognostischen Aspekt eine Rolle, da sich die Frage stellt, ob Voraussagen über das Verhalten des Systems möglich sind, ohne es vollständig zu realisieren, also wirklich „laufen zu lassen“. Die Implementierung über Objekte liefert Beispiele für ***strukturierte Zerlegungen*** und ***einfache Programmierkonzepte***.

Wir können dieses Verhalten der Handelspartner leicht durch Zustandsdiagramme beschreiben:

K: „kooperieren“

B: „betrügen“



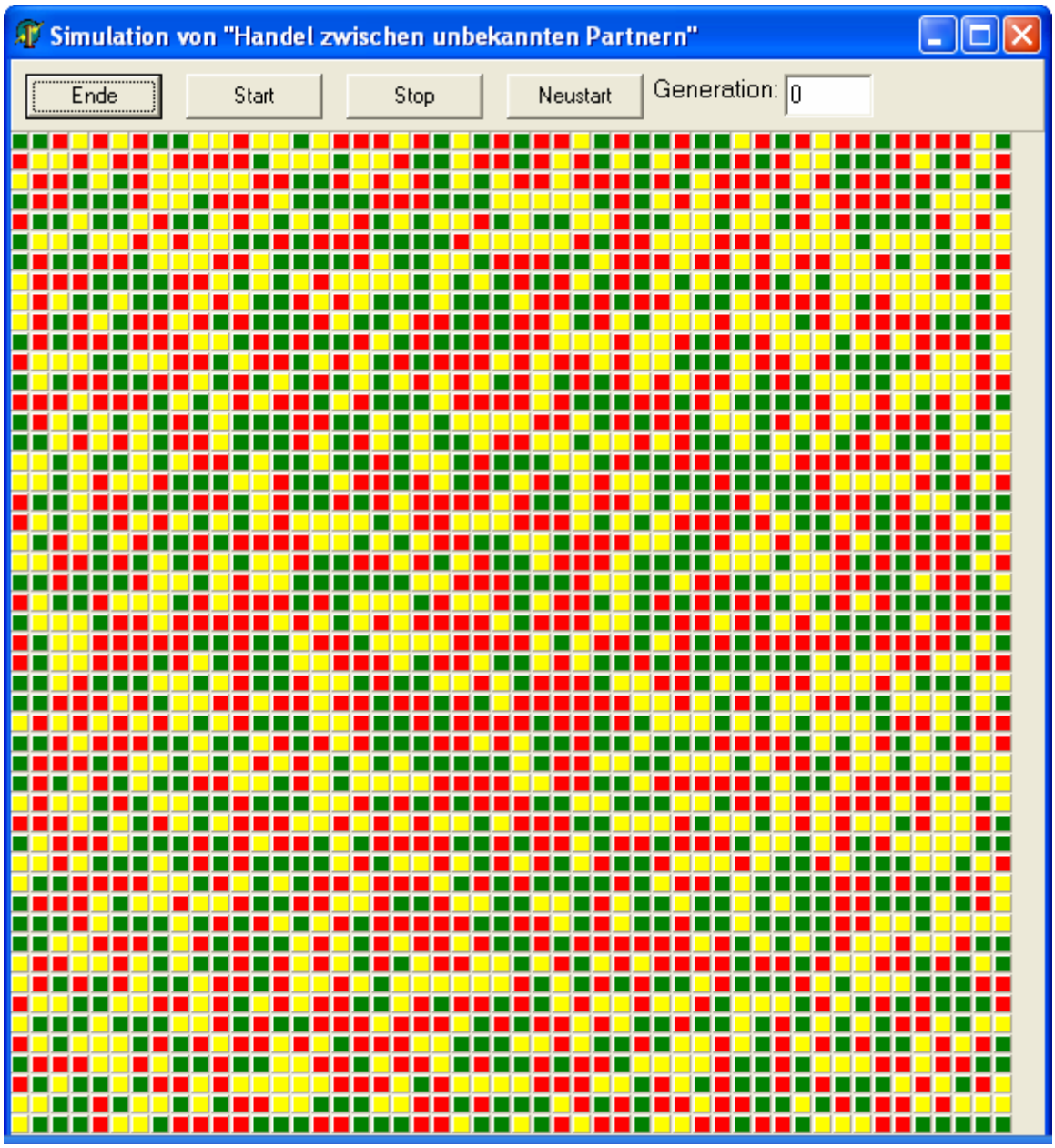
Der „Naive“ Der „Betrüger“ Der „Gewitzte“

Ordnen wir solche Automaten in einem Gitter an, verteilen sie zufällig und färben sie entsprechend ihrem Zustand (grün als „Naiver“, rot als „Betrüger“ oder gelb als „Gewitzter“) ein, dann erhalten wir ein Bild ähnlich dem folgenden:

---

[436] Zahlreiche gut umsetzbare Beispiele finden sich z. B. in [Ger95]

[437] Man kann sich das wie in einer Auktionsbörse organisiert vorstellen, z. B. in ebay.



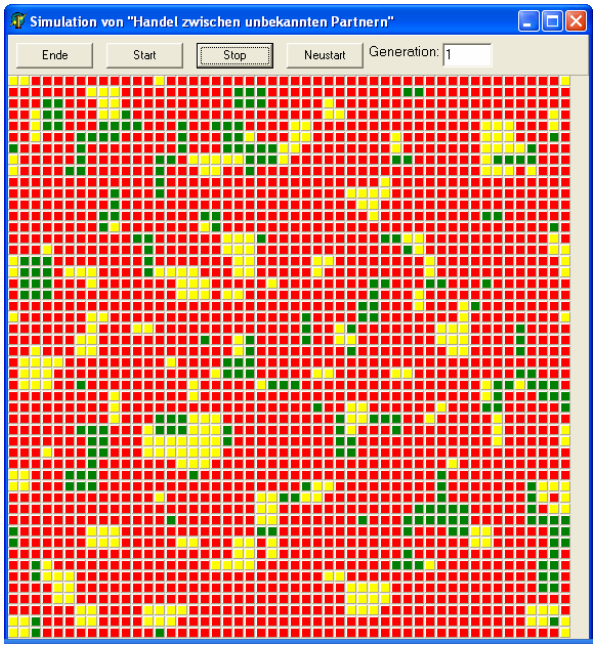
Der weitere Ablauf ist einfach:

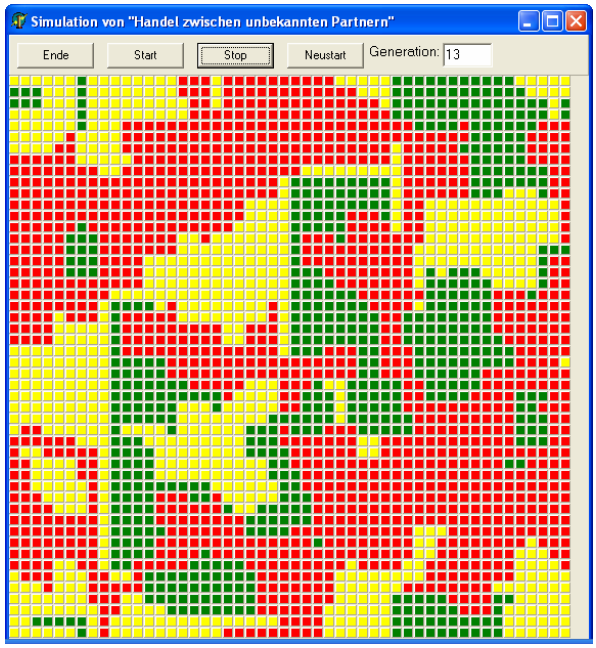
- Zuerst handeln alle Partner einmal mit ihren Nachbarn aus der Moore-Nachbarschaft. Dabei ist einiges an Orientierung im Gitter (als Array) vonnöten: es wechselt die „Blickrichtung“ (Stellung in der Nachbarschaft), und an den Rändern muss man auch überlegen.
- Danach bewerten alle Partner den Erfolg ihrer Nachbarn.
- Als Opportunisten übernehmen sie daraufhin den Zustand des erfolgreichsten Nachbarn oder behalten ihren Zustand bei, wenn sie selbst besser waren.

Im Beispiel (s. Materialien) wird eine Automatenklasse in einer eigenen Unit vereinbart und darauf aufbauend eine „Welt“ (auch in einer Unit), die mit einem Gitter aus Automaten hantiert. Beides wird von der Programmoberfläche des zellulären Automaten gesteuert. Die Abläufe zwischen diesen Klassen sind zwar nicht ganz trivial (besonders bei der grafischen Darstellung in der Windows-Welt), dafür aber nur einmal zu lösen. Danach können die Automaten in ihrer Unit bzw. die „Welt“ in der an-

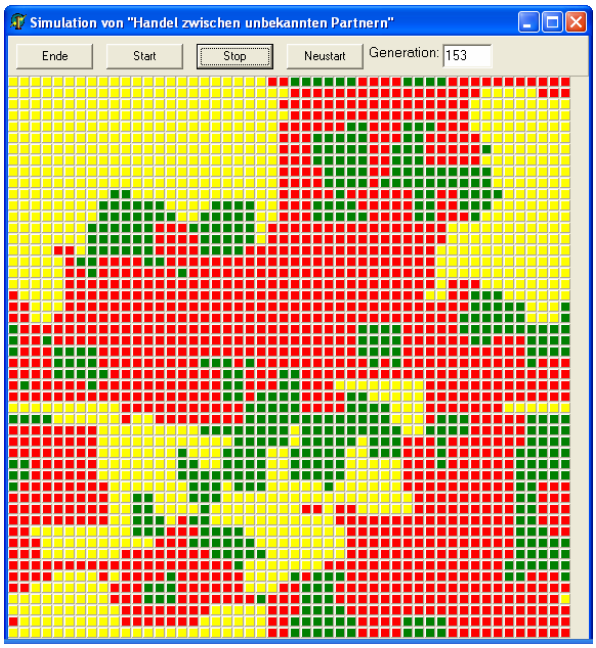
deren getrennt und ohne direkte Beeinflussung manipuliert werden. Man kann also „schön einfach“ experimentieren.

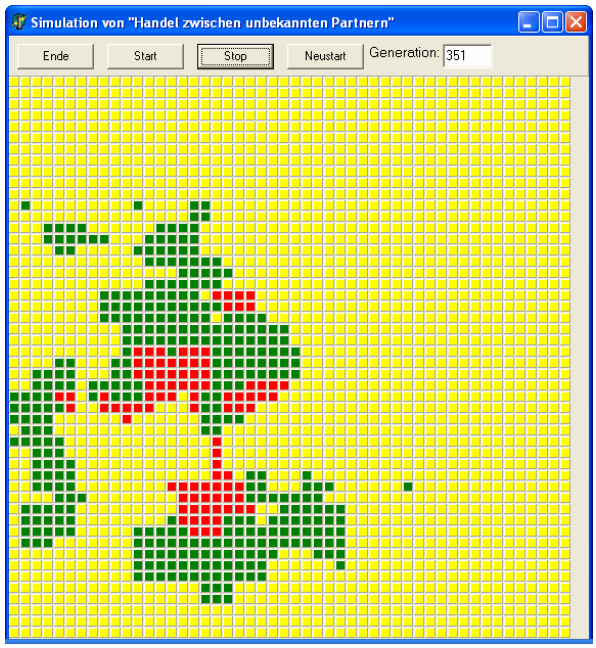
In den ersten Generationen setzen sich meist „die Bösen“ durch. Doch danach bilden sich Cluster aus „Guten“ bzw. „Gewitzten“, und dann beginnt eine wilde „Schlacht“.





Zwar werden die „Guten“ hart von den „Betrügern“ bedrängt. Sie halten sich aber ganz gut in Gruppen. Die „Gewitzten“ setzen sich gegenüber den „Betrügern“ – je nach Konfiguration – meist durch und kooperieren mit den „Guten“.





Am Ende siegen meist die „Gewitzten“ – aber eben nicht immer.

Das Beispiel ist als Vorbereitung auf die Automatentheorie eher schlicht. Es bietet aber eine extreme Bandbreite von sowohl programmiertechnischen wie inhaltlichen Variationen und legt sozialwissenschaftliche bzw. naturwissenschaftliche Interpretationen nahe. In dieser Beziehung wird der Modellcharakter besonders deutlich.

Bleiben wir zuerst bei den Variationen:

- Das Verhalten der „Partner“ kann leicht durch veränderte Strategien ergänzt werden. „Wettbewerbe“ zwischen verschiedenen Strategien werden möglich, wobei Statistiken geführt werden müssen, da der Einzelfall nicht viel über das Gesamtverhalten aussagt. Schon hier ist systematisches Arbeiten gefragt.
- Die „Welt“ kann verändert werden durch veränderte Nachbarschaften (von-Neumann-Nachbarschaft, andere Reichweiten, …), andere Abläufe, ... Statt des zweidimensionalen Gitters kann die zeitliche Entwicklung linearer Automaten in der üblichen Weise dargestellt werden, wobei chaotische Vorgänge auftreten[438]. Bei beidem können sowohl die Art der Automaten wie das Steuerprogramm völlig ignoriert werden.
- Die Gewichtungsfaktoren, die den Gewinn bei unterschiedlichen Vorgängen bestimmen, sind veränderbar.[439] Auch hier sind die anderen Größen irrelevant.
- Das Steuerprogramm kann verbessert werden, z. B. um während des Programmlaufs die unterschiedlichen Faktoren zu verändern.[440] Dafür sind „Oberflächenprogrammierer“ gefragt.
- Die Eigenschaften der Gitterautomaten sollten auch per Mausklick gesetzt werden können (einige „Gewitzte“ in einer Welt aus „Bösen“, …), um das Verhalten bestimmter Konfigurationen gezielt untersuchen zu können.

Es kann aber auch versucht werden, die beobachteten Vorgänge systematisch zu *bewerten*:

- Zur Bewertung der Systeme sind globale Größen geeignet („Bruttosozialprodukt“ als Summe aller „Handelspunkte“, …). Der Einfluss der Parameter auf das Erreichen und die Art des ggf. erreichten Endzustands kann abgeschätzt werden. Lineare Automaten lassen sich entsprechend klassifizieren.[441]
- Es können Beschränkungen eingeführt werden („Anzahl der handelbaren Güter“, „Geldmenge“, …), und die Verteilung der Größen auf die Gruppen sowie deren zeitliche Entwicklung ist darstellbar.

Die Beobachtung der manchmal überraschenden Abläufe liefert Ansatzpunkte zur Diskussion ethischer Fragen. Auch wenn das Beispiel natürlich nicht direkt auf gesellschaftliche Systeme übertragbar ist, so haben wir doch ein für die meisten neuartiges Argument für kooperatives, soziales Verhalten gefunden, das nicht aus transzendenten oder philosophischen Überlegungen gewonnen wird, sondern aus Effizienzbetrachtungen. Es steht darin in klarem Gegensatz zur Egozentrik des Primitivdarwinismus, der oft die öffentliche Diskussion in dieser Hinsicht beherrscht. Unterrichtseinheiten dieser Art steuern damit einem die „Ellenbogengesellschaft“ fördernden und fordernden Biologismus entgegen.

Unsere zellulären Automaten sollten den Handel im Internet simulieren. Was erhaltene Modell ist aber auch ganz anders interpretierbar: Betrachten wir den „Handel“ als Energieaustausch benachbarter Teilchen, dann kommen wir recht schnell zum *Ising-Modell* für Spingitter[442]. Die globalen Größen „Temperatur“ und „Magnetisierung“ liefern Bewertungsmaßstäbe zur Beurteilung des Systems. Relativ kleine Än-

[438] [Schr91] ab S. 388, [Man91] S. 436
[439] Im vorgestellten Programm wurde dafür etwas „Zufall“ eingebaut.
[440] Das ist auch bitter nötig, da die Instantiierung so vieler Objekte einige Zeit dauert.
[441] z. B. [Ger95] S. 59 nach Stephen Wolfram
[442] [Arg94] S. 405 ff.

derungen führen auf Strukturbildungsprozesse und/oder Modelle für biologische und chemische Systeme[443].

Die Interpretation desselben Modells in unterschiedlichem Kontext, die (teilweise) Unvorhersagbarkeit der Ergebnisse, das dynamische Verhalten und sein Bezug zu nichtlinearen Systemen liefert Erfahrungen mit Modellen, Simulationen, deren Mächtigkeit und deren Grenzen – und das fast ohne Mathematik. Die Visualisierungsmöglichkeiten der Computer machen so einerseits der Schule völlig neue Gebiete zugänglich und verdeutlichen andererseits den von der Mathematik unterschiedenen Charakter der Informatik als „Prognosesystem“. Nebenbei ist das Thema eine unerschöpfliche Quelle von „Facharbeiten“ der Schülerinnen und Schüler.

Haben wir als Lerngruppe z. B. den ersten Leistungskurs der Stufe 12 etwa zur Mitte des Semesters, dann sollte dieser eigentlich einen soliden Anfangsunterricht in der Klasse 11 durchlaufen haben. Leider ist derzeit aber zumindest bei schulübergreifenden Kursen davon auszugehen, dass die Vorkenntnisse kaum vergleichbar sind. Die Lernenden werden deshalb in diesem ersten Kurs die komprimierte „Nachlieferung“ fehlender Inhalte[444] noch kaum „verdaut“ haben. Sie sollten aber mit der *elementaren Algorithmik*, *primitiven Datentypen* und *einfacher Computergrafik* inzwischen halbwegs vertraut sein. Beginnen wir danach eine Unterrichtseinheit über *„Visualisierung großer Datenmengen“*, dann steht uns einerseits das weite Feld der Falschfarbendarstellungen von astronomischen[445], geografischen[446] oder medizinischen[447] Daten mit ihren Interpretationsmöglichkeiten offen, andererseits bilden Gitterautomaten ein interessantes Arbeitsgebiet, das hier betrachtet wird.

Repräsentieren wir die Teilautomaten durch Objekte, dann können die entsprechenden Klassen aus grafischen Komponenten der GUI abgeleitet werden[448]. Wir haben damit neben einem elementaren Zugang zu endlichen Automaten auch einen einfachen Einstieg in OOP-Methoden gefunden. In der angegebenen Form erfordert die Nutzung der Transitionsgraphen kaum Zeit: Bei einer Diskussion unterschiedlicher Strategien, z. B. der des „Langmütigen“, wird diese Notationsform nebenbei eingeführt und gefestigt. Erheblich mehr Zeit benötigt eine eingehende Analyse der Abläufe, die daraus folgende Verteilung der Informationen und ihre Repräsentation. Wird hier nicht sorgfältig gearbeitet, dann kann es später erhebliche Probleme geben. Sind die Alternativen und deren Konsequenzen betrachtet und die wesentlichen Entscheidungen getroffen, dann werden die Teilprobleme einzeln angegangen:

- Eine Automatenklasse mit den erforderlichen Methoden wird vereinbart, implementiert und an einzelnen Objekten getestet.
- Ein Gitter solcher Automaten wird als Array definiert.
- Die Aktionen im Gitter werden realisiert.

Ziel des Unterrichts ist es, bei der Implementierung eines relativ einfachen Modells elementare informatische Techniken kennen zu lernen, sowie sich mit Simulationen zu beschäftigen, deren Ergebnisse kaum prognostizierbar sind und die zu Diskussionen anregen.

---

[443] [Cra88]

[444] die man ganz gut mit einer Einheit über Programmverifikation „anreichern“ kann, damit nicht in Vergessenheit gerät, dass es sich um einen Leistungskurs handelt.

[445] z. B. aus dem Projekt „Hands-On Universe“

[446] z. B. Satellitenbilder

[447] z. B. mit NMR-Daten aus der Tomografie

[448] Das ist zwar nicht effizient, aber sehr lehrreich.

Unter diesen Voraussetzungen wird eine anfängliche eingehende Diskussion der Problematik im Unterrichtsgespräch unbedingt erforderlich sein. Aus dieser sollten sich dann die zu lösenden Teilprobleme ergeben, und aus diesen folgen die erforderlichen Programmiertechniken. Die Vereinbarung einer Tochterklasse z. B. von GUI-Panels, die mit Automateneigenschaften ausgestattet wird, sollte ebenfalls gemeinsam durchgeführt werden, wobei der Neuigkeitswert nicht bei den in Methoden auftretenden Algorithmen, sondern in den Zugriffstechniken liegt. So etwas lässt sich schnell abhandeln, die notwendigen Erfahrungen werden bei der Anwendung gewonnen. Sind zweidimensionale Felder schon bekannt, dann können die Lernenden das Zusammenspiel der Automaten selbst realisieren. Die dabei möglichen Fehler sollten erkannt und korrigiert werden. Als Hilfe wird ggf. das zufällige Erzeugen einer Anfangsbelegung des Feldes vorgegeben. Danach wird das Modell wie beschrieben variiert und erprobt.

Mit diesen Informationen lässt sich die Unterrichtseinheit klassifizieren:

Thema: zelluläre Automaten

Zeitbedarf: ab 10 WStd. (je nach Ziel)

Voraussetzungen: Arrays, elementare OOP, einfache grafische Darstellungen

Die Einheit dient der Verdeutlichung

der kulturellen Bedeutung des Themas	zu 10 %
gesellschaftlicher Auswirkungen des Themas	zu 50 %
rein fachlicher Aspekte	zu 40 %

Die folgenden Unterrichtsmethoden erfordern an Unterrichtszeit ca.

Lehrervortrag:	10 %
Unterrichtsgespräch:	30 %
Partnerarbeit:	
Einzelarbeit:	
Projektarbeit:	60 %

Das Thema verdeutlicht die folgenden fundamentalen Ideen:

1. Algorithmisierung
   - 1.1 Entwurfsparadigmen (Branch and Bound, Backtracking, ...)
   - 1.2 Programmierkonzepte (Alternative, Iteration, Rekursion, ...) zu 10 %
   - 1.3 Ablauf (Prozess, Nebenläufigkeit, ...):
   - 1.4 Evaluation (Verifikation, Komplexität, ...):

2. strukturierte Zerlegung
   - 2.1 Modularisierung (Methoden, Hilfsmittel, ...) zu 20 %
   - 2.2 Hierarchisierung (Darstellung, Realisierung, ...) zu 20 %
   - 2.3 Orthogonalisierung (Emulation, ...)

3. Formalisierung
   - 3.1 formale Sprache (Syntax, Semantik, ...)
   - 3.2 Automat (Zustand, Übergang, Vernetzung, ...) zu 40%
   - 3.3 Berechenbarkeit (Grenzen, Durchführbarkeit, ...) zu 10 %

Der Delphi-Quelltext findet sich bei den Materialien am Ende der Arbeit.

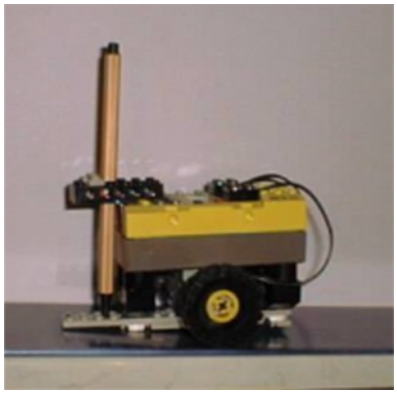
4.4 Beispiel 4: Eine Sprache für LEGO-Roboter

LEGO-Mindstorms-Roboter sind für Unterrichtszwecke in jeder Altersstufe hervorragend geeignet. Sie betonen als „Spielzeuge“ die spielerischen Elemente des Unterrichts, können entweder über eine ActiveX-Komponente in Windows-basierte Programmiersprachen oder mithilfe besonderer Sprachen[449], die im Internet frei verfügbar sind, eingebunden und direkt gesteuert werden oder als Geräte mit eingebauter eigener Programmiersprache als Ziel von Kompilierungsprozessen dienen.

Steuert man die Roboter direkt[450], dann kann man ihnen über eine geeignete Programmoberfläche einzelne Befehle oder Befehlsfolgen schicken und die Ergebnisse direkt beobachten. Die Sensoren des Geräts können abgefragt, ausgewertet und wiederum zur Steuerung benutzt werden. Wohlgemerkt: nicht nur in den grafischen LEGO-Programmsystemen, sondern genauso unter Java, C++, C# oder Delphi. In diesem Kontext veranschaulichen die Geräte die ***Idee des Automaten*** besonders gut. Verfügt der Roboter z. B. über einen Stift, der angehoben und gesenkt werden kann, dann ist auch sein ***Zustand*** direkt beobachtbar. Valentin Braitenberg hat in seinem interessanten und sehr witzigen Buch[451] eine Fülle von „kybernetischen Wesen“ beschrieben, deren erste Versionen sich direkt als Roboter nachbauen lassen.

In diesem Beispiel[452] sollen die Roboter als Ziel von Übersetzungsprozessen einer formalen Sprache dienen, die definiert, geparst und in die „Robotersprache“ übersetzt wird. Das Beispiel ist deshalb als Projektphase in einem Kurs über theoretische Informatik anzusiedeln und erfordert einige Vorkenntnisse und – je nach Komplexität der definierten Sprache – auch einiges an Zeit. Ziel ist es also, eine erweiterbare ***Turtlegrafik für LEGO-Mindstorms-Roboter***[453] zu schreiben, die

- die spielerischen Elemente der Roboter für den Unterricht nutzt,
- die Grundelemente einer Computersprache am konkreten Beispiel einführt: Editor, Parser und (rekursiver) Interpreter werden „gebaut“,
- die Möglichkeiten der OOP ausnutzt und
- Testmöglichkeiten am Bildschirm vorsieht.



Unter einer *Turtle* versteht man ein computergesteuertes Modell, das ähnlich wie eine Schildkröte aussieht, einen Stift trägt und sich bewegen kann. Je nach Ausbau „versteht“ das Modell unterschiedliche Befehlssätze, die seine Bewegungen steuern. Bei seinen Bewegungen hinterlässt die Turtle bedingt durch den Stift eine „Spur“, die eine Zeichnung erzeugt.

---

[449] z. B. mit NQC

[450] Was etwas langsam und fehlerträchtig ist, aber den Schülerinnen und Schüler besonders dann viel Vergnügen macht, wenn sich mehrere Geräte im Raum befinden.

[451] [Bra93]

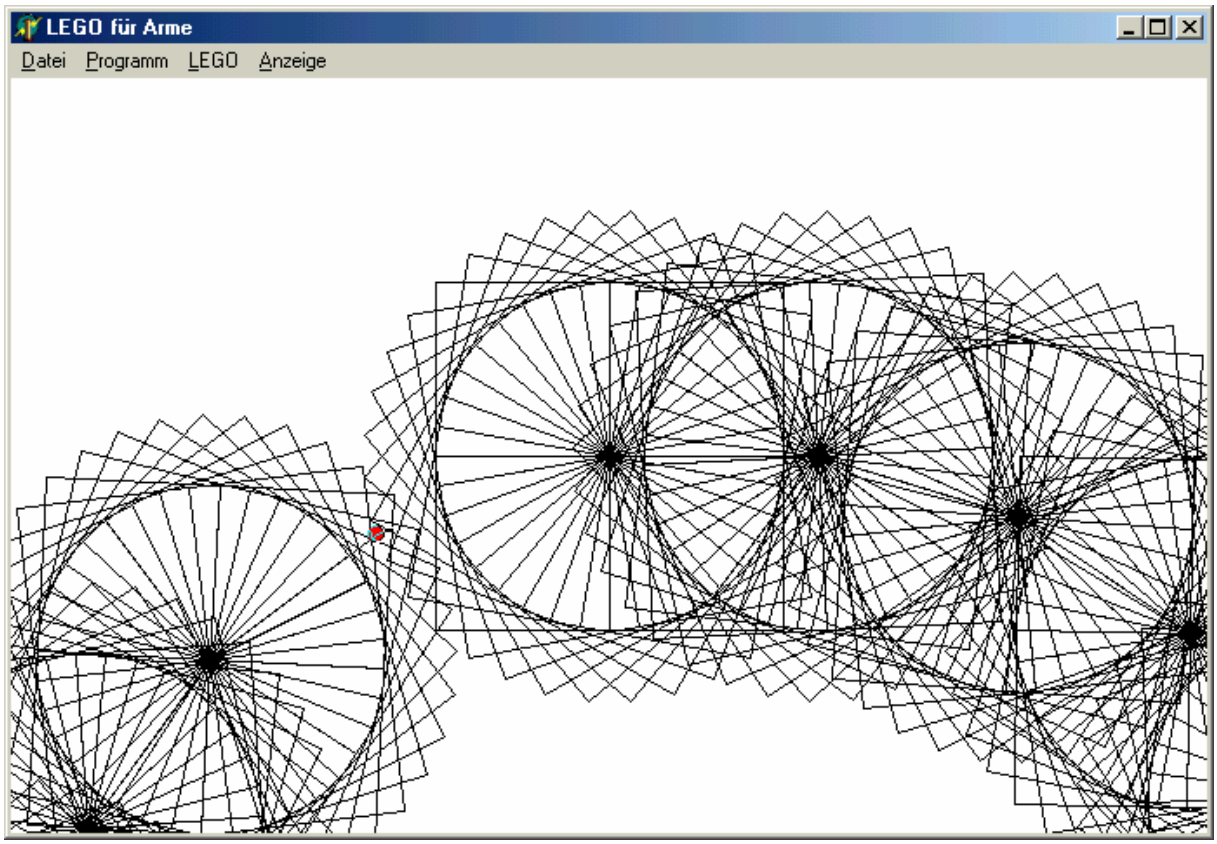
[452] Das Beispiel stammt aus einem N21-Projekt über Roboter und robotische Teleskope, das ich für diese niedersächsische Initiative entwickelt habe.

[453] Grundlage dieses Beispiels sind [Mod92b], [Mod98] und [Mod00]

Eine funktionsfähige Turtle zu bauen ist mechanisch nicht gerade einfach.

- Entscheidend ist die Position des Stifts möglichst im Drehpunkt des Modells (was beim abgebildeten Primitivmodell nicht der Fall ist),
- eine funktionsfähige Einrichtung zum Heben und Senken des Stifts (fehlt beim Modell, wird aber durch „Kippen“ bei Vorwärtsbewegungen teilweise realisiert),
- eine genaue Winkelteilung, die stark von der Mechanik und dem Untergrund abhängt. Entsprechend muss das Modell oft „geeicht“ werden.

Um einen Eindruck von den Möglichkeiten des Systems zu erhalten folgt hier ein Screenshot[454] von der *ScreenTurtle*, die zum Testen der Programme dient:



Das entsprechende Programm lautet:

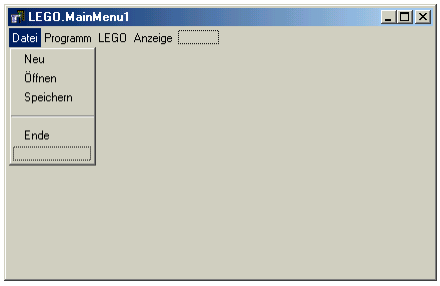
```
W 20 (
      W 36 (
            W 4 (
                 V 100 R 90
                )
            R 10
           )
      V 120 R17
     )
$
```

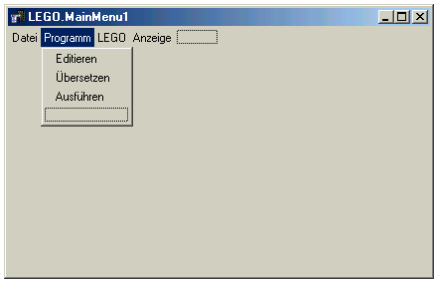
Geschachtelte Schleifen. Innen wird ein Rechteck gezeichnet.

[454] Grundlage dieses Beispiels sind [Mod92b], [Mod98] und [Mod00]

Zur Realisierung ist es erforderlich

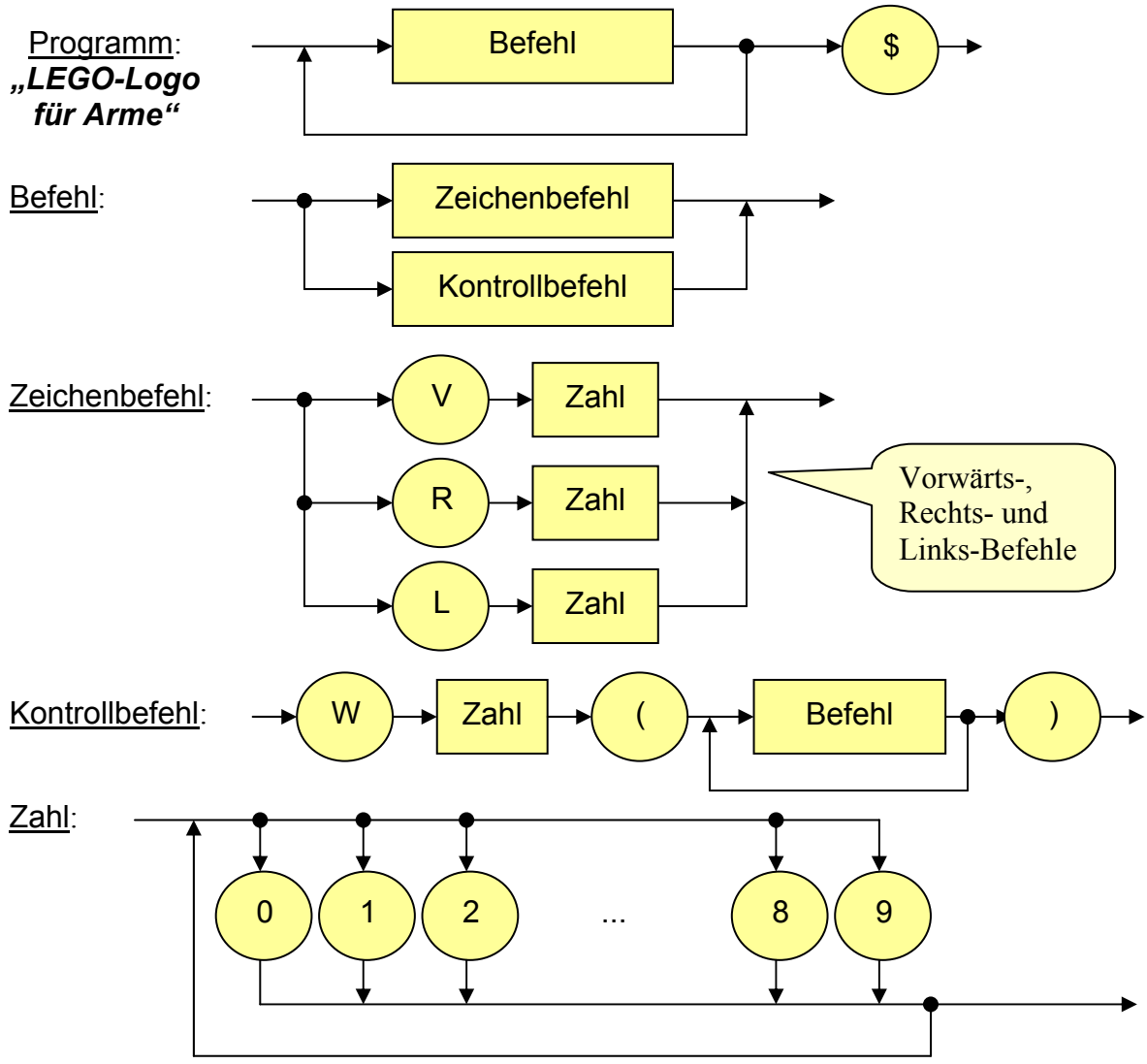
- ein Menüsystem mit Delphi zusammenzustellen, das eine einfache Dateiverwaltung für Texte realisiert (editieren, laden, speichern),
- die geschriebenen Programmtexte auf syntaktische Korrektheit zu überprüfen und ggf. Fehler anzuzeigen und
- die „übersetzten“ Programme richtig zu interpretieren.





Der erste Punkt lässt sich leicht mithilfe vorhandener Delphi-Komponenten innerhalb eines Menü-Systems realisieren. Interessanter ist schon die Programmiersprache. Wir müssen Programmtexte schreiben, übersetzen und ausführen können. Dazu benutzen wir das zweite Untermenü. Das dritte dient zur Umschaltung zwischen der „echten“ LEGO-Turtle und dem Bildschirmmodell u. Ä.

Um die Programmtexte übersetzen zu können, benötigen wir eine Sprachdefinition, die wir über Syntaxdiagramme geben:



Da die Programme innerhalb der Schleifen geschachtelt werden können, ist ein Kellerautomat zur Analyse der Klammerstruktur erforderlich. Die restlichen syntaktischen Konstrukte können von einem einfachen endlichen Automaten analysiert werden, der ggf. Fehlermeldungen produziert. Der endliche Automat wird durch seinen Transitionsgraphen beschreiben. (Die etwas eigenwillig notierten Kellerbefehle sind fett hervorgehoben.)

Parser für „LEGO-Logo für Arme“:

Eingabealphabet: IE = {V,R,L,W,0,1,2,3,4,5,6,7,8,9,(,),$}

Zustandsmenge: IS = {s0, s1, s2, s3, s4, s5, s6, s7, se, sf}

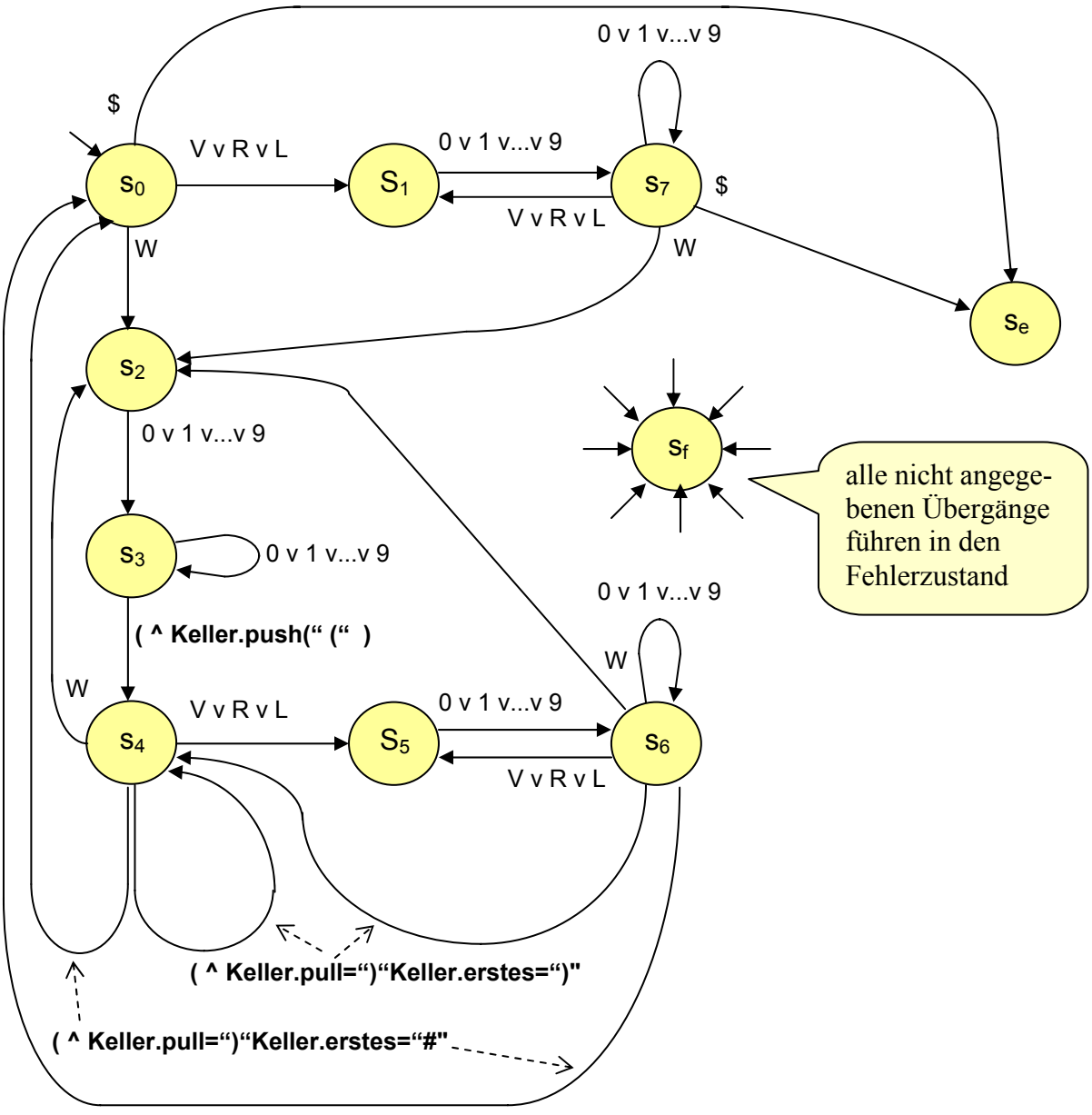
Überführungsfunktion des Kellerautomaten:

[(,s0,#] →[s0,(#]

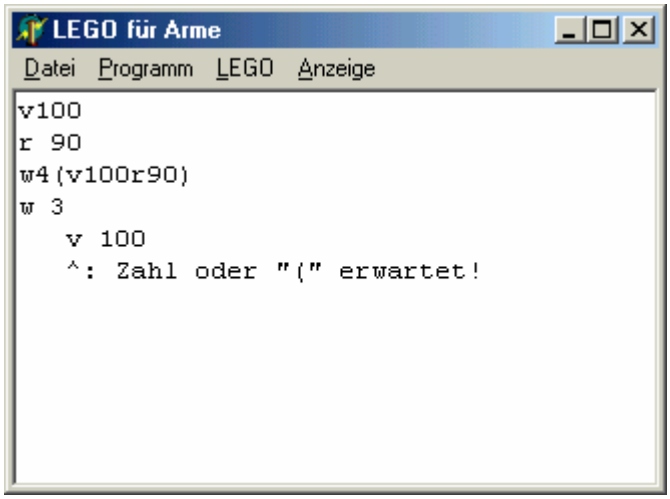
[(,s0,(] →[s0,((]

[),s0,(] →[s0,]

[,s0,#] →[se,]



Fehlermeldungen des entsprechenden Parsers sehen dann z. B. so aus:



Für die Interpretation des Programms wählen wir eine „reduzierte“ Version des Quelltextes, in der überflüssige Zeichen (Leerzeichen, Zeilenwechsel, ...) weggelassen worden sind. Als Ziele der Befehle stehen eine Bildschirmturtle und eine LEGO-Turtle zur Verfügung, die über die gleichen Befehlssätze („Methoden“) verfügen. Mit diesen Methoden kann der Interpreter implementiert werden. Damit geschachtelte Schleifen abgearbeitet werden können, interpretiert der Interpreter (Befehls-)Zeichenketten. Trifft er auf eine Schleife, dann sucht er sich den Schleifenkörper zusammen (der weitere Schleifen enthalten kann) und ruft sich selbst mit diesen Befehlen wieder auf. Der Interpreter arbeitet also rekursiv.

Für den korrekten Ablauf des Programms benötigt man weitere Methoden, z. B. um den Inhalt von Memofeldern zu kopieren, Anfangswerte zu setzen und vor allem, um die LEGO-Turtle zu „eichen“. Die von der Turtle zurückgelegt Strecke wird über eine Wartezeit definiert. Sie hängt aber natürlich auch von der Bauart der Turtle und dem Untergrund ab. Bei Vorwärtsbewegungen ändert sich nur die Größe der Figuren durch diese Werte. Bei Drehungen hängt der Drehwinkel und damit die Form der Zeichnungen stark von ihnen ab. Ich lasse meistens ein Rechteck zeichnen und prüfe, ob die Turtle danach in die gleiche Richtung zeigt. Damit kann man ganz gut die Zeit für eine volle Umdrehung korrigieren. Den entsprechenden Faktor für Vorwärtsbewegungen sollte man so einstellen, dass die LEGO-Turtle-Figuren zur Schrittweite der Bildschirmturtle passen.

Der Umgang mit verschiedenen Programmiersprachen gehört zumindest zur Informatik-Prüfungsfachkursfolge. Zu fragen ist dann natürlich, *wie* diese verschiedenen Sprachen „erfahren“ werden sollen. Eine der Möglichkeiten besteht m. E. darin, selbst eine solche – natürlich stark vereinfachte – Sprache zu schreiben[455]. Der Charme dieses Vorgehens liegt darin, dass in der bekannten Umgebung weiter gearbeitet werden kann und sich all die bekannten Teilprobleme des Theoriekurses der Schulinformatik zwanglos ergeben – aus der Anwendung. Es werden also zwei Fliegen mit einer Klappe geschlagen. Damit gewinnt man Zeit – und die brauchen wir dringend für die Projektphasen.

Sprachdefinitionen lassen sich auf sehr unterschiedlichem Niveau umsetzen. Sie dienen deshalb auch zur Binnendifferenzierung in den Kursen, ermöglichen gemeinsames Arbeiten am gleichen Thema bei sehr unterschiedlichem Zeiteinsatz. Das

[455] Das funktioniert auch ganz gut mit reduzierten deklarativen Sprachen wie „Miniprolog“ oder „MiniSQL“.

Editieren und Ändern von Programmen, das Speichern und Laden u. Ä. lässt sich mit den integrierten grafischen Programmentwicklungssystemen durch wenige Mausklicks und einige Programmzeilen umsetzen. Diese intellektuell nicht gerade herausfordernde Funktionalität hat also inzwischen den gebührenden – geringen – Stellenwert erhalten. In ähnlichen Zeiträumen können deshalb wesentlich anspruchsvollere Projekte in Angriff genommen werden als früher, ohne auf eine Implementierung der Ideen zu verzichten. Der Informatikunterricht hat deshalb m. E. mit den neuen Möglichkeiten an Wert gewonnen.

Haben wir als Lerngruppe z. B. den Grundkurs der Stufe 13 im zweiten Halbjahr, dann stehen wir kurz vor dem Abitur, sollten also bei Gelegenheit bekannte Inhalte wiederholen, aber auf neue nicht verzichten. Folglich bietet sich eine längere Projektphase an, weil bei etwas umfangreicheren Vorhaben unterschiedliche Techniken benutzt werden, wobei sich eine Wiederholung älterer Themen automatisch ergibt. Haben wir bei der technischen Informatik im vorhergehenden Kurs endliche Automaten benutzt, dann müssen in diesem Kurs noch die Themen *erkennende Automaten*, zusammen mit Sprachdefinitionen, Parsern, …sowie ggf. *Kellerautomaten* behandelt werden, wobei z. B. die schon bekannten Stapel und Listen wieder auftauchen. Danach kann unser LEGO-Projekt starten.

Arbeiten wir zuerst mit einer Screen-Turtle (also auf dem Bildschirm), dann können wir uns ganz auf die Definition und das Parsen der entsprechenden Sprachkonstrukte konzentrieren. Die Implementierung der Turtle-Befehle sollte den Schülerinnen und Schülern dieser Stufe keine Schwierigkeiten mehr bereiten. Nach Abschluss dieser Phase liegen die Sprache und die erforderlichen Turtle-Methoden fest. In einem zweiten Schritt kann jetzt die LEGO-Turtle gebaut und über einfach zusammengeklickte Oberflächen gesteuert und getestet werden (s. o.). Es bleibt allein das „handwerkliche“ Problem offen, die Screen-Turtle-Methoden auch für die LEGO-Turtle zu implementieren.

Verzichten wir auf geschachtelte Schleifen, dann kann der Parser drastisch vereinfacht werden. Berücksichtigen wir die Zustände der LEGO-Sensoren (Druckschalter, Lichtsensoren, …), dann können wir zusätzlich sinnvoll Alternativen oder weitere Schleifenarten in die Sprache einführen, z. B.:

```
IF (Sensor1=0) (R180 V30 R210)
```

Es bestehen also genügend Möglichkeiten, das Projekt sowohl einfacher (und damit kürzer) als auch umfangreicher zu halten oder es über unterschiedliche Teilgruppenaufgaben anders zu strukturieren. Vor allem aber gibt es mehrere Zeitpunkte, an denen das Projekt ggf. auch sinnvoll, also ohne Frustrationen zu hinterlassen, abgebrochen werden kann.

Ziel des Unterrichts ist es, die beim Entwurf und der Implementierung von Programmiersprachen auftauchenden Probleme zu diskutieren und durch Anwendung von Methoden der theoretischen Informatik in einem einfachen Fall zu lösen. Die dabei benötigten Datenstrukturen erfordern nebenbei eine effiziente Wiederholung vorheriger Kursinhalte.

Unter diesen Voraussetzungen wird zuerst eine eingehende Diskussion der Problematik im Unterrichtsgespräch erforderlich sein, wobei die Konsequenzen unterschiedlicher Syntaxen gegeneinander abgewogen werden. Die Implementierung des Parsers sollte zu diesem Zeitpunkt kein ernstes Problem mehr sein. Schwieriger ist die Interpretation der Sprachkonstrukte, also die Übersetzung der selbst definierten Sprache in Sprachkonstrukte des Roboters. Wählen wir hierfür einen rekursiven In-

terpreter, dann reduziert sich das Problem weitgehend auf das Erkennen der richtigen Klammerstruktur, und so etwas ist ggf. mit Hilfen gut machbar.

Das Schreiben einer Turtlegrafik für den Bildschirm stellt keine besonderen Anforderungen und kann parallel z. B. von einer Teilgruppe erledigt werden – wenn so ein System nicht schon vorhanden ist. Der Bau des Roboters ist ein gesondertes Problem. Man kann auf vorhandene Baupläne zurückgreifen, die Aufgabe ebenfalls einer Teilgruppe überlassen oder ein Minimalmodell wählen, wie es sich auf der Abbildung in diesem Kapitel findet. Die Übersetzung von Turtlebefehlen in die Sprache des Roboters erfordert dann wiederum Hilfen der oder des Unterrichtenden.

Die Unterrichtseinheit wird wie folgt klassifiziert:

Thema: Die LEGO-Turtle

Zeitbedarf: ab 15 WStd.

Voraussetzungen: zwei Informatikkurse incl. Datenstrukturen, erkennende endliche Automaten, ggf. Kellerautomaten

Die Einheit dient der Verdeutlichung

der kulturellen Bedeutung des Themas
gesellschaftlicher Auswirkungen des Themas zu 10 %
rein fachlicher Aspekte zu 90 %

Die folgenden Unterrichtsmethoden erfordern an Unterrichtszeit ca.

Lehrervortrag: 10 %
Unterrichtsgespräch: 20 %
Partnerarbeit:
Einzelarbeit:
Projektarbeit: 70 %

Das Thema verdeutlicht die folgenden fundamentalen Ideen:

1. Algorithmisierung
   - 1.1 Entwurfsparadigmen (Branch and Bound, Backtracking, ...)
   - 1.2 Programmierkonzepte (Alternative, Iteration, Rekursion, ...) zu 20 %
   - 1.3 Ablauf (Prozess, Nebenläufigkeit, ...):
   - 1.4 Evaluation (Verifikation, Komplexität, ...):

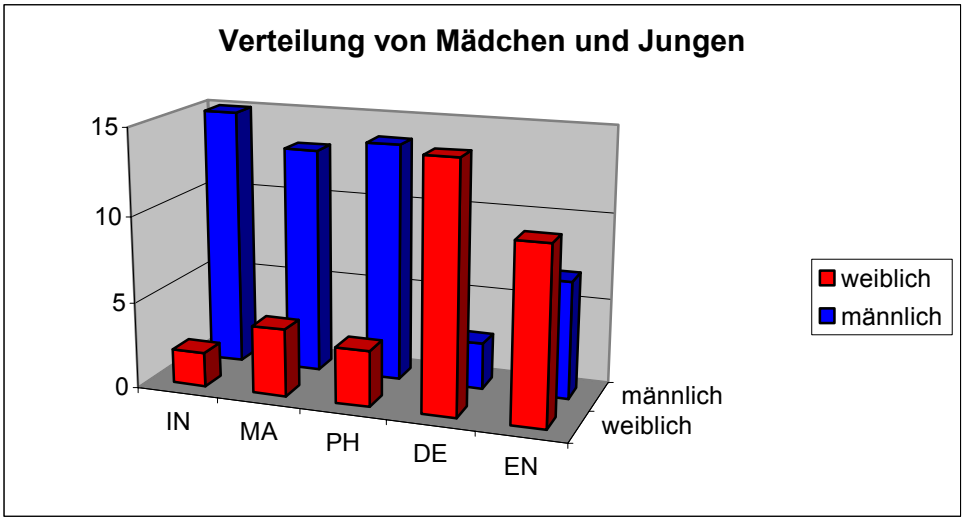
2. strukturierte Zerlegung
   - 2.1 Modularisierung (Methoden, Hilfsmittel, ...) zu 10 %
   - 2.2 Hierarchisierung (Darstellung, Realisierung, ...) zu 10 %
   - 2.3 Orthogonalisierung (Emulation, ...)

3. Formalisierung
   - 3.1 formale Sprache (Syntax, Semantik, ...)
   - 3.2 Automat (Zustand, Übergang, Vernetzung, ...) zu 40%
   - 3.3 Berechenbarkeit (Grenzen, Durchführbarkeit, ...) zu 20 %

Die Delphi-Quelltexte finden sich bei den Materialien am Ende der Arbeit.

Anhang: Einige Ergebnisse zweier Umfragen

Zu Beginn des Schuljahres 2001/02 wurden an meiner Schule insgesamt 95 Fragebögen von Schülerinnen und Schülern der Stufe 12 ausgefüllt, die gerade einen Leistungskurs begonnen hatten. Unter diesen waren 17 aus dem derzeit einzigen Leistungskurs Informatik in Niedersachsen. Die Umfrage ist nicht repräsentativ, und die Daten sind auch nicht direkt vergleichbar, weil es sich beim Informatikkurs um einen schulübergreifenden Kurs aus fünf Oberstufen handelt, bei den anderen Kursen aber nicht. Befragt wurden Lernende der Fächer Informatik, Mathematik und Physik, weil diese Kurse eine ähnliche Struktur bzgl. der Geschlechterverteilung haben, sowie Deutsch und Englisch zum Vergleich. Die Geschlechterverteilung ist wie üblich – leider.



Ziel der Befragung war es u. a. festzustellen, ob der geplante hohe Anteil selbstständiger Arbeitsphasen im Informatikkurs den Interessen der Lernenden entgegenkommt – oder nicht. Da die Kursteilnehmerinnen und -teilnehmer den Unterrichtenden zum Zeitpunkt der Befragung noch nicht kannten, sind deren Aussagen in diesem Punkt unbeeinflusst.

A1: Zur Selbsteinschätzung der Lernenden

Die erfragten Zensurenmittel entsprechen insgesamt der üblichen Verteilung, wobei die beiden „Sprachkurse“ wohl deshalb besser liegen, weil Mädchen insgesamt bessere Zensuren haben. Die Vorzensuren (aus der Klasse 11) aller Schülerinnen und Schüler sind in ihren Leistungsfächern besser als im Mittel – deshalb haben sie sie ja u. a. gewählt. Auffällig ist allerdings die starke Abweichung in Informatik.

	IN	**MA**	**PH**	**DE**	**EN**
Zensurenmittel gesamt	2,56	2,60	2,56	2,37	2,39
Zensurenmittel Fach	1,42	2,10	2,13	2,11	2,00
Differenz: Gesamt – Fach	**1,14**	**0,50**	**0,44**	**0,26**	**0,39**

Zu Beginn wurden einige Fragen zur Selbsteinschätzung persönlicher Eigenschaften gestellt, jeweils allgemein und bezogen auf das Leistungsfach. Als Erstes zu *Sorgfalt*, *Fleiß* und *Ehrgeiz*. Informatikschülerinnen und -schüler schätzen sich selbst allge-

mein nur zu 41 % als sorgfältig ein. Nur die „Physiker“ liegen ähnlich tief. Ganz anders sieht es dann bei der Sorgfalt im Fach aus:

	IN	MA	PH	DE	EN
Sorgfalt allgemein	41 %	62 %	48 %	95 %	76 %
Sorgfalt im Fach	76 %	85 %	57 %	95 %	71 %
Differenz: Fach – allgemein	**35 %**	**23 %**	**10 %**	**0 %**	**–5 %**

Informatikschülerinnen und -schüler zeigen also – nach eigener Einschätzung – in diesem Unterricht den größten Unterschied zu ihrem sonstigen Verhalten von allen Fächern überhaupt. Nur „Mathematiker“ machen das ähnlich, allerdings von einem höheren Niveau aus.

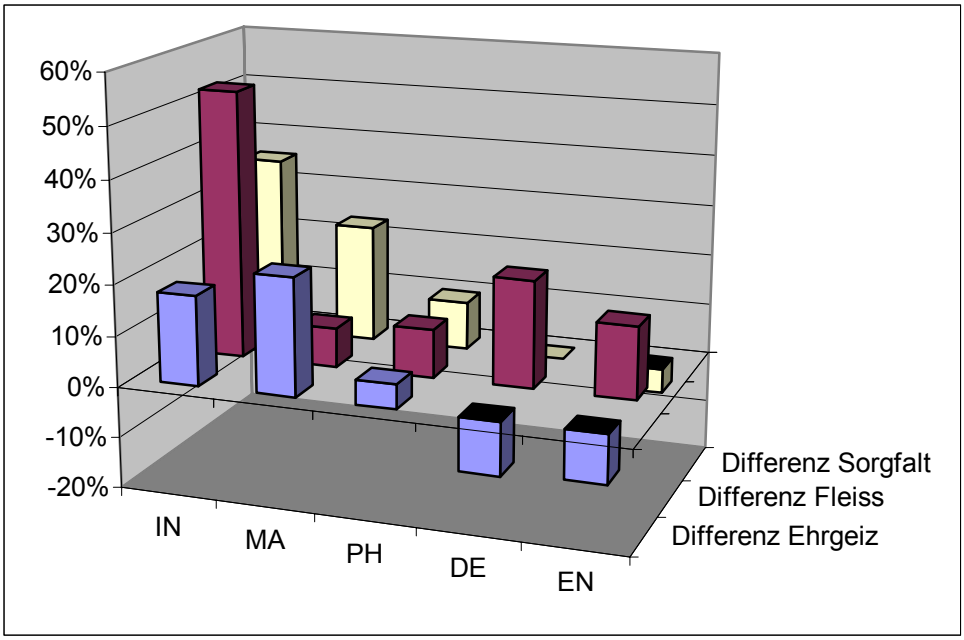
Ganz ähnlich sind die Ergebnisse beim *Fleiß*. Sie fallen noch drastischer aus.

	IN	MA	PH	DE	EN
Fleiß allgemein	6 %	38 %	33 %	58 %	57 %
Fleiß im Fach	59 %	46 %	43 %	79 %	71 %
Differenz: Fach – allgemein	**53 %**	**8 %**	**10 %**	**21 %**	**14 %**

Beim *Ehrgeiz* herrscht ziemlich Flaute bei den Informatikern, sehr im Unterschied zu den Kursen, in denen die Mädchen überwiegen (das gilt auch für den Fleiß). Fachlich gesehen machen hier die Mathematiker den größten Sprung.

	IN	MA	PH	DE	EN
Ehrgeiz allgemein	47 %	69 %	62 %	100 %	90 %
Ehrgeiz im Fach	65 %	92 %	67 %	89 %	81 %
Differenz: Fach – allgemein	**18 %**	**23 %**	**5 %**	**–11 %**	**–10 %**

Besonders deutlich werden die fachbezogenen Unterschiede im Diagramm.



Gefragt wurde entsprechend nach *Kreativität*, der *Vielseitigkeit der Interessen* und der *Vorausplanung der Arbeit*. Im Unterschied zu den Sprachkursen schätzen sich die „Naturwissenschaftler“ allgemein als deutlich weniger kreativ ein[456]. Im Fach ändert sich da wenig – außer bei den Informatikern. Die erreichen hier Werte höher als

[456] Vermutlich finden sich hier auch geschlechtsspezifische Unterschiede.

der Deutschkurs. Bemerkenswert ist die Selbsteinschätzung der Mathematiker in diesem Bereich.

	IN	MA	PH	DE	EN
Kreativität allgemein	65 %	69 %	57 %	74 %	71 %
Kreativität im Fach	82 %	46 %	62 %	79 %	57 %
Differenz: Fach – allgemein	**18 %**	**–23 %**	**5 %**	**5 %**	**–14 %**

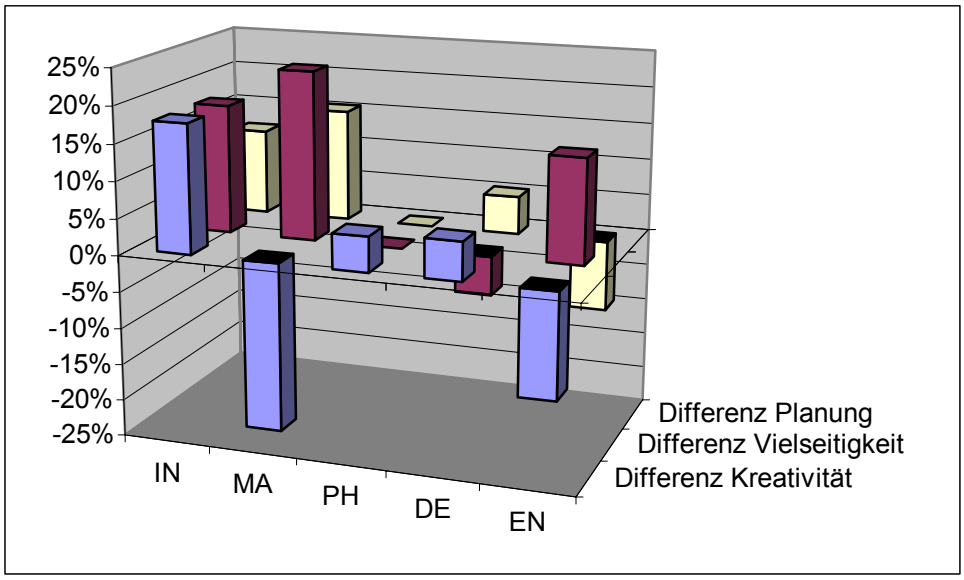
Bei der Vielseitigkeit der Interessen ähneln sich die Kurse im Allgemeinen erstaunlich. Fachlich gesehen legen dann die Mathematiker und Informatiker ähnlich stark zu. In diesen Fächern scheinen erhebliche *fachliche* Interessen vorzuliegen.

	IN	MA	PH	DE	EN
vielseitig interessiert allg.	76 %	69 %	81 %	84 %	81 %
vielseitig interessiert im Fach	94 %	92 %	81 %	79 %	95 %
Differenz: Fach – allgemein	**18 %**	**23 %**	**0 %**	**–5 %**	**14 %**

Auch bei der Vorausplanung von anfallenden Arbeiten ähneln sich Mathematiker und Informatiker auffallend. Beide legen hier im fachlichen Bereich zu, während sich in den anderen Fächern wenig tut.

	IN	MA	PH	DE	EN
Planung der Arbeit allgemein	29 %	31 %	43 %	32 %	48 %
Planung der Arbeit im Fach	41 %	46 %	43 %	37 %	38 %
Differenz: Fach – allgemein	**12 %**	**15 %**	**0 %**	**5 %**	**–10 %**

Auch hier die Differenzen noch einmal als Diagramm:



Etwas verblüffend ist die hohe Neigung der Informatiker, *bei auftretenden Problemen schnell aufzugeben*. In allen Fächern zeigen sich hier wenige Unterschiede vom Allgemeinen zum Fach. Nur die „Anglisten“ halten sich fachlich für hartnäckiger, und Mathematiker und Physiker geben „überhaupt nicht“ auf.

	IN	MA	PH	DE	EN
bei Problemen aufgeben allg.	24 %	8 %	5 %	11 %	24 %
bei Problemen im Fach aufg.	24 %	8 %	10 %	16 %	10 %
Differenz: Fach – allgemein	**0 %**	**0 %**	**5 %**	**5 %**	**–14 %**

Von der Gruppenarbeit halten die Informatiker ähnlich wie die „kreativen“ Deutschkurse viel, die Mathematiker deutlich weniger. Ähnlich ungern arbeiten nur die Anglisten in ihrem Fach zusammen.

	IN	MA	PH	DE	EN
Gruppenarbeit allgemein	59 %	38 %	52 %	53 %	48 %
Gruppenarbeit im Fach	59 %	38 %	57 %	63 %	38 %
Differenz: Fach – allgemein	**0 %**	**0 %**	**5 %**	**11 %**	**–10 %**

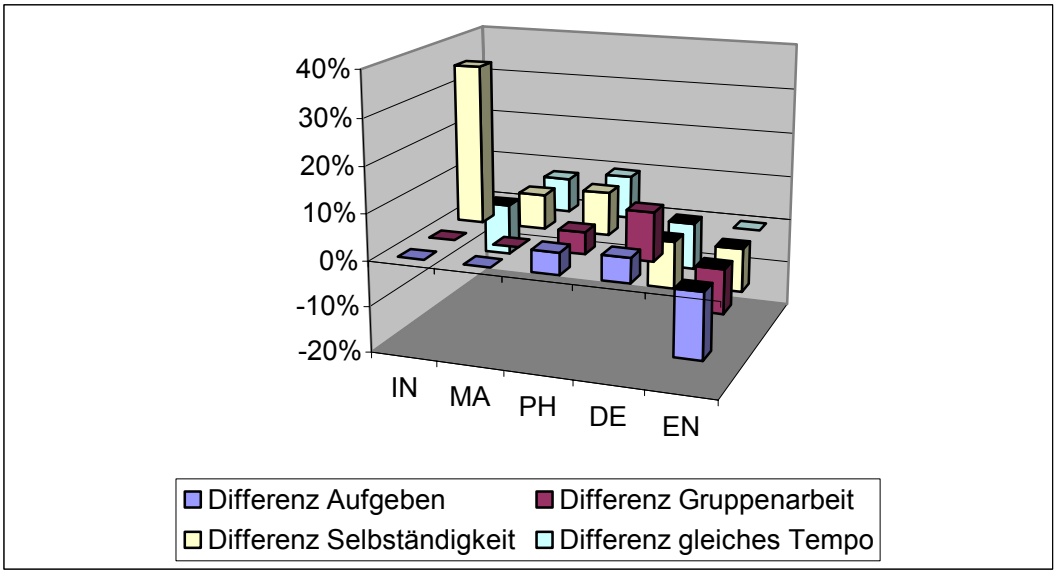
Fast schon dramatische Unterschiede zeigen sich in der Neigung zur *selbstständigen Arbeit*. Obwohl sich die Informatiker hier im Allgemeinen als eher unselbstständig einschätzen, erreichen sie im Fach Spitzenwerte. Erstaunlich sind die Ergebnisse in den Sprachkursen, die sich allgemein ja als sehr selbstständig einschätzen.

	IN	MA	PH	DE	EN
selbstständig arbeiten allgem.	29 %	38 %	57 %	68 %	71 %
selbstständig arbeiten i. Fach	65 %	46 %	67 %	58 %	62 %
Differenz: Fach – allgemein	**35 %**	**8 %**	**10 %**	**–11 %**	**–10 %**

Folgerichtig halten die Informatiker auch nichts vom „Lernen im gleichen Tempo“.

	IN	MA	PH	DE	EN
im gleichen Tempo allgemein	41 %	46 %	33 %	58 %	48 %
im gleichen Tempo im Fach	29 %	54 %	43 %	47 %	48 %
Differenz: Fach – allgemein	**–12 %**	**8 %**	**10 %**	**–11 %**	**0 %**

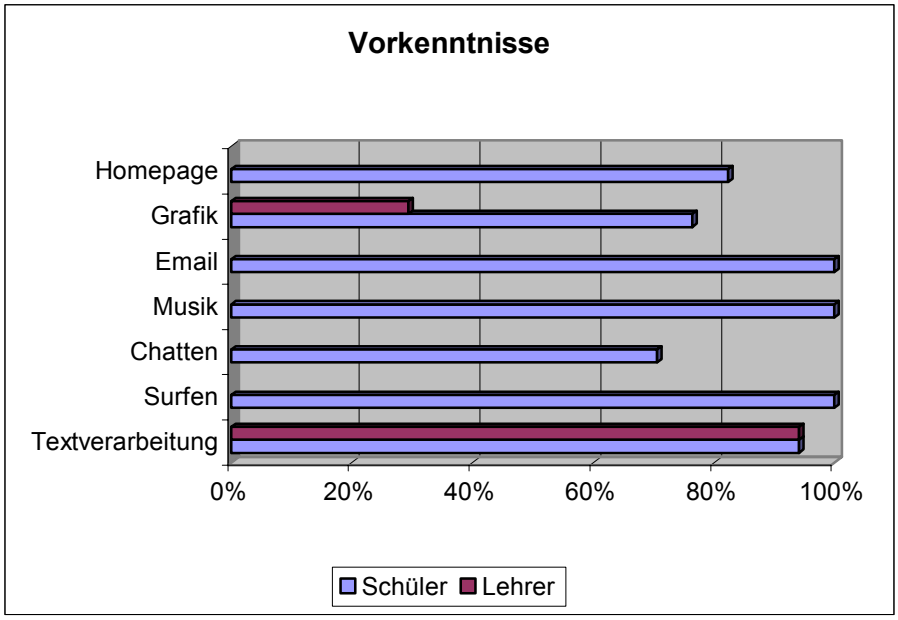
Auch hier die Unterschiede im Diagramm:



A2: Vergleich mit einem Lehrerweiterbildungskurs

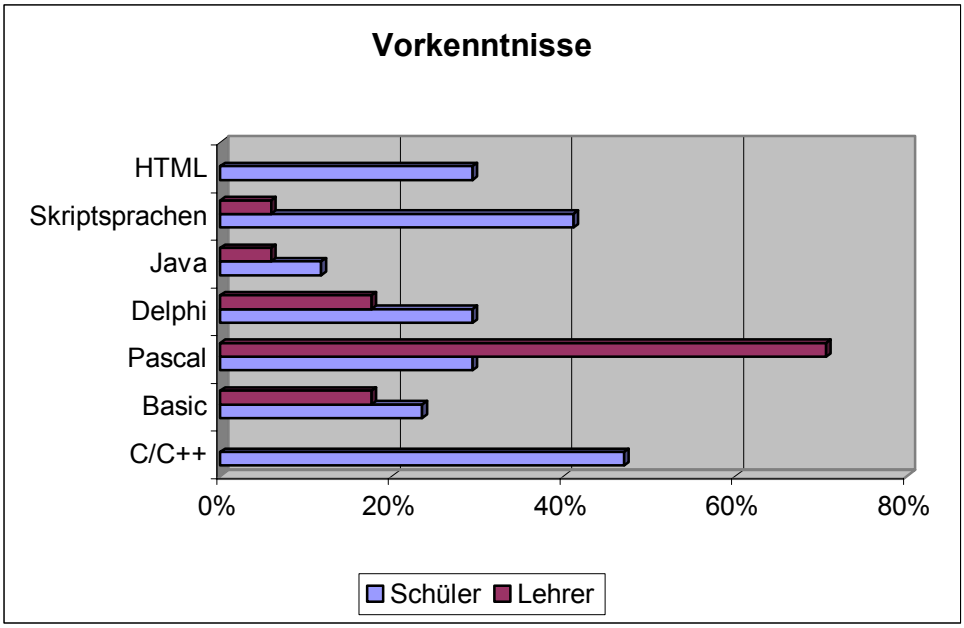
Von Interesse ist vielleicht der Vergleich von Schülern und Lehrern, die sich intensiv mit einem neuen Fach beschäftigen wollen: die Schüler im Leistungskurs, die Lehrer in der Lehrerweiterbildungskursfolge. Einige Monate nach der Schülerbefragung wurden zwanzig Kolleginnen und Kollegen u. a. nach ihren Vorkenntnissen befragt. (Allerdings wurden dabei eher „private“ Aktivitäten wie „Chatten“ ausgespart.) Das Medium Computer wird von den Schülerinnen und Schülern in allen Sparten praktisch flächendeckend benutzt. Auch die Kolleginnen und Kollegen nutzen praktisch alle den Computer zum Schreiben (die meisten auch als Spreadsheet). Andere „dienstliche“ Anwendungen wie Präsentationsprogramme, Datenbanken usw. sind nur noch spärlich vertreten.

Vorkenntnisse:	**Schüler**	**Lehrer**
Textverarbeitung	94 %	94 %
Surfen	100 %	k. A.
Chatten	71 %	k. A.
Musik	100 %	k. A.
Email	100 %	k. A.
Grafik	76 %	29 %
Homepage	82 %	k. A.



Noch drastischer sind die Unterschiede bei bekannten Programmiersprachen. Während ca. die Hälfte der Schülerinnen und Schüler sich mit C/C++ beschäftigt, kannte keiner der Kolleginnen und Kollegen diese Sprache. Fast umgekehrt liegt es bei PASCAL. Auch heute neu anfangende Lehrer/innen kennen überwiegend diese Sprache – vermutlich noch aus der Schule.

Vorkenntnisse:	Schüler	Lehrer
C/C++	47 %	0 %
Basic	24 %	18 %
Pascal	29 %	71 %
Delphi	29 %	18 %
Java	12 %	6 %
Skriptsprachen	41 %	6 %
HTML	29 %	k. A.



Für den Fachunterricht sind die unterschiedlichen Programmier-Vorkenntnisse der Schülerinnen und Schüler gravierend. Die breite Streuung, besonders die Nutzung von Scriptsprachen wie PHP, resultiert aus der weitgehend *privaten* Beschäftigung mit Computern. Da sowohl aus beginnendem Informatiklehrermangel (der weitgehend eine Folge des Mathematiklehrermangels ist[457]) als auch aus schwindendem Interesse bei den Schülern Informatikunterricht entweder ausfällt oder in „Nischen" wie „Homepagegestaltung", „Administration des Schulservers", … stattfindet, kann nicht mehr auf vergleichbare Voraussetzungen aufgebaut werden. Hier machen sich die fehlenden Standards des Faches drastisch bemerkbar. Der Kurs musste entsprechend fast ohne Voraussetzungen mit elementarer Algorithmik beginnen. Umso erfreulicher ist der schnelle Lernfortschritt der Lernenden. Offensichtlich kommt es wirklich nicht so sehr auf gelernte Inhalte an, sondern auf Methodenkenntnisse (die hier weitgehend in Eigeninitiative erarbeitet wurden) und die Bereitschaft, sich auf neue Themen einzulassen. In diesem Bereich schätzen sich die Informatik-Lernenden offensichtlich richtig ein.

Das Programmierenlernen in Eigeninitiative ist möglicherweise eine Ursache für den relativ hohen Prozentsatz von Informatikschülerinnen und -schüler, die von sich glauben, bei auftretenden Problemen „schnell aufzugeben". Im Unterricht hat sich diese Einschätzung nicht bestätigt. Es könnte aber sein, dass auf einem so schwierigen Gebiet, das anfangs auch kaum Zwischenschritte zwischen „richtig" und „falsch" zulässt, sich die fehlende Unterstützung in diesem Sinne bemerkbar macht.

[457] Schulleiter stehen bei Lehrermangel in diesem Bereich vor der Frage, ob sie das *Pflichtfach* Mathematik zugunsten des *Wahlfaches* Informatik kürzen. Da die Informatikkurse in der Regel auch noch klein sind, ist die Antwort klar.

A3: Zu den Erwartungen an den Unterricht

Da projektartige Unterrichtsphasen die Bereitschaft der Lernenden voraussetzen, sich auf diese Arbeitsform einzulassen und so phasenweise die Verantwortung für ihre Lern- und Arbeitsergebnisse selbst zu übernehmen, folgte noch eine Fragengruppe zu diesem Bereich. Anzumerken ist wiederum, dass die Befragten die Zielrichtung der Befragung nicht kannten.

Die erste Fragengruppe beschäftigte sich mit der Unterrichtsführung und der Art der Problemfindung:

„Der Unterricht soll vom Lehrer klar in Einzelschritten geführt werden.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	31 %	62 %	57 %	58 %	67 %
bezogen auf das Fach	31 %	85 %	81 %	47 %	62 %
Differenz: Fach – allgemein	**0 %**	**23 %**	**24 %**	**–11 %**	**–5 %**

Die Informatikschülerinnen und -schüler halten also generell und sehr deutlich am wenigsten von kleinschrittigem Unterrichten. In Mathematik und Physik wird das anders gesehen. In den Sprachkursen ist solch ein Unterricht kaum möglich.

„Der Unterricht soll größere Abschnitte enthalten, in denen ich selbstständig Aufgaben bearbeite.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	69 %	46 %	33 %	37 %	48 %
bezogen auf das Fach	94 %	31 %	43 %	53 %	48 %
Differenz: Fach – allgemein	**25 %**	**–15 %**	**10 %**	**16 %**	**0 %**

Entsprechend hoch wird das selbstständige Arbeiten eingeschätzt.

„Ich möchte mir selbst Aufgaben wählen, die in den Unterricht eingehen.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	63 %	46 %	55 %	68 %	62 %
bezogen auf das Fach	88 %	38 %	57 %	74 %	62 %
Differenz: Fach – allgemein	**25 %**	**–8 %**	**2 %**	**5 %**	**0 %**

Auch Einfluss auf die Aufgabenstellung wird gewünscht, und zwar signifikant stärker im Fach Informatik. Nur der Deutschkurs erreicht einen ähnlich hohen Wert, wahrscheinlich, weil er sich ebenfalls als sehr kreativ einstuft (s. dort). In den anderen Kursen wird kaum zwischen den Fächern differenziert.

„Die Unterrichtsinhalte sollen nur vom Lehrer vorgegeben werden.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	0 %	23 %	14 %	0 %	10 %
bezogen auf das Fach	0 %	31 %	43 %	0 %	5 %
Differenz: Fach – allgemein	**0 %**	**8 %**	**29 %**	**0 %**	**–5 %**

Die Ergebnisse sind folgerichtig und bedürfen kaum eines Kommentars. Auch hier ist die Parallelität zum Deutschkurs frappierend. Mathematiker und Physiker haben es ganz gerne, wenn Unterrichtende die Linie vorgeben.

„Unterrichtsinhalte sollen sich auch aus den bearbeiteten Problemen ergeben.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	69 %	77 %	90 %	79 %	90 %
bezogen auf das Fach	81 %	100 %	90 %	79 %	90 %
Differenz: Fach – allgemein	**13 %**	**23 %**	**0 %**	**0 %**	**0 %**

In allen Kursen wird problemorientiertes Arbeiten gewünscht.

Die nächste Fragengruppe bezog sich auf die Bereitschaft, Probleme „offen“ anzugehen, sie ggf. auch nicht abschließend im Unterricht zu behandeln.

„Alle Inhalte sollen im Unterricht eingeführt und dort vollständig behandelt werden.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	38 %	46 %	57 %	42 %	57 %
bezogen auf das Fach	31 %	54 %	57 %	42 %	48 %
Differenz: Fach – allgemein	–6 %	8 %	0 %	0 %	–10 %

Das erwarten Informatiklernende generell kaum.

„Ich möchte mir Inhalte auch selbst erschließen, z.B. aus Büchern oder aus dem Internet.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	31 %	31 %	29 %	47 %	71 %
bezogen auf das Fach	75 %	23 %	48 %	53 %	62 %
Differenz: Fach – allgemein	**44 %**	**–8 %**	**19 %**	**5 %**	**–10 %**

Hier zeigt sich extremer Unterschied von 44 % beim Informatikkurs. Im Gegensatz zu den Sprachkursen mögen die „Naturwissenschaftler“ diese Arbeitsweise allgemein nicht. Bezogen auf die Informatik ändert sich die Einstellung schlagartig und springt auf den höchsten Wert überhaupt. Wahrscheinlich macht sich hier die Erfahrung der Kursteilnehmerinnen und -teilnehmer bemerkbar, sich selbst in ein Thema einarbeiten zu können (s. unter „Vorkenntnisse“).

„Aufgaben und Lösungsanforderungen sollen im Detail eindeutig definiert sein, so dass ich genau weiß, was von mir erwartet wird.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	50 %	85 %	71 %	79 %	76 %
bezogen auf das Fach	50 %	85 %	86 %	74 %	76 %
Differenz: Fach – allgemein	**0 %**	**0 %**	**14 %**	**–5 %**	**0 %**

Hier ist die Meinung bei den Informatikern gespalten, während in den anderen Kursen überwiegend Klarheit gewünscht wird. Fehlende Eindeutigkeit kann nicht nur als Offenheit, sondern auch als Möglichkeit zur Willkür interpretiert werden, und diese Erfahrung möchten sich die meisten wohl ersparen. Diese Interpretation wird durch das folgende Ergebnis nahe gelegt, das eine analoge Aussage enthält, diesmal aber positiv formuliert. Auch hier ähneln die Ergebnisse eher den sprachlichen Kursen als den verwandten naturwissenschaftlichen.

„Aufgabenstellungen sollen so offen sein, dass ich eigene Ideen entwickeln und umsetzen kann.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	63 %	38 %	57 %	68 %	76 %
bezogen auf das Fach	81 %	54 %	76 %	74 %	86 %
Differenz: Fach – allgemein	**19 %**	**15 %**	**19 %**	**5 %**	**10 %**

Zuletzt folgen einige Fragen zum exemplarischen Lernen.

„Ich möchte einen möglichst breiten Überblick über fachliche Inhalte und Methoden.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	38 %	54 %	48 %	32 %	38 %
bezogen auf das Fach	25 %	62 %	52 %	16 %	52 %
Differenz: Fach – allgemein	**–13 %**	**8 %**	**5 %**	**–16 %**	**14 %**

„Ich möchte weniger Stoff behandeln, mich dafür aber exemplarisch mit einzelnen Problemen sehr gründlich auseinandersetzen.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	13 %	31 %	43 %	32 %	57 %
bezogen auf das Fach	31 %	31 %	57 %	42 %	43 %
Differenz: Fach – allgemein	**19 %**	**0 %**	**14 %**	**11 %**	**–14 %**

Auch diese Ergebnisse sind eindeutig. Informatiker möchten Methodenkenntnisse erwerben.

„Ich möchte im Unterricht Anregungen gewinnen, die ich über den Unterricht hinaus verfolgen will.“					
	IN	**MA**	**PH**	**DE**	**EN**
allgemein	38 %	54 %	67 %	79 %	71 %
bezogen auf das Fach	81 %	69 %	76 %	74 %	71 %
Differenz: Fach – allgemein	**44 %**	**15 %**	**10 %**	**–5 %**	**0 %**

Der Informatikkurs ist bemerkenswert desinteressiert an Anregungen in anderen Fächern, erwartet diese aber im Fach selbst in hohem Maße.

Fazit:

Auch wenn die Befragung nicht repräsentativ war (und schon wegen der fehlenden Vergleichsgruppen nicht sein konnte), zeigt sie doch Tendenzen, die ziemlich eindeutig sind:

- Informatikschülerinnen und -schüler stufen sich selbst als wenig sorgfältig, fleißig, ehrgeizig und vorausplanend ein. Sie ähneln darin den Mathematik- und Physikschülerinnen und -schülern, liegen aber auch, verglichen mit denen, noch unter dem Schnitt. Im Fach selbst aber billigen sie sich ein erheblich besseres Arbeitsverhalten zu.
- Im Bereich der Kreativität und dem sozialen Lernen ähneln die Selbsteinschätzungen eher den Sprachkursen.
- Extrem ausgeprägt sind die selbst eingeschätzte Fähigkeit und die Erwartung an den Unterricht bzgl. des selbstständigen, problemorientierten Lernens. Die Antworten sind auch in sich stimmig.

Es scheint zumindest in diesem Kurs so zu sein, dass die Möglichkeiten des Informatikunterrichts in Hinsicht auf konstruktivistisches, selbstbestimmtes und selbstverantwortliches Lernen und Arbeiten den Interessen der Kursteilnehmerinnen und –teilnehmern entsprechen. Auch die Kursergebnisse selbst zielen in diese Richtung. Während „Pflichtaufgaben“ eher unwillig erledigt werden, liefert der Kurs in freien Arbeitsphasen überwiegend sehr sehenswerte Ergebnisse völlig unterschiedlicher Art, die mit Freude und Stolz vorgestellt werden.

Nachtrag:

Im neuen Schuljahr 2002/03 haben 27 Schüler aus fünf Kursstufen der nächsten Informatik-Leistungskursfolge erneut den Fragebogen ausgefüllt. Da im Vergleich zur ersten Befragung jetzt offensichtlich andere Voraussetzungen gegeben waren – z. B. in Bezug auf die Vorbereitung der Schüler auf den Kurs – wurden die neuen Zahlen nicht in die alte Auswertung aufgenommen. Die zweite Befragung bestätigte aber im Mittel die Ergebnisse der ersten – mit den bei so geringen Zahlen üblichen Schwankungen. Es ergab sich, dass

- der Unterschied zwischen Informatik-Vorzensur und mittlerer Schulnote noch etwas größer war (1,28 statt 1,14),
- die Selbsteinschätzung bzgl. Sorgfalt, Fleiß, … in der Tendenz wie im Vorjahr, jedoch ähnlicher wie in Mathematik/Physik war,

- eine noch verstärkte Tendenz zum „schnellen Aufgeben“ bei Problemen im Fach auftritt, wobei die Unterschiede zwischen der fachlichen und allgemeinen Einschätzung groß war (19 % statt 0 %),
- die Schüler sich diesmal generell als ziemlich selbstständig einschätzten.

Interessant ist, dass sich der Ruf eines neuen Leistungsfaches sehr schnell bildet. Im neuen Schuljahr hatten 52 % (statt 29 %) der Schüler Vorkenntnisse in Delphi, und 85 % (statt 75 %) wollen sich neue Inhalte auch selbst erschließen. Es hatte sich also ein großer Teil der Schüler selbstständig auf die im Kurs benutzte Programmiersprache vorbereitet. Die Unterrichteten wollen sich nicht nur neue Inhalte selbst erschließen – sie haben es längst getan.

Fragebogen zu Ihren Erwartungen an den Informatikunterricht

Schuljahr 2001/2 (Alle Angabe sind freiwillig. Die Umfrage erfolgt anonym.)

zur meiner Person:

Alter: ____ Klassenstufe: _____ Geschlecht: m ○ w ○

Prüfungsfächer: 1. LF:_____ 2. LF:_____ 3. PF:_____ 4. PF:_____

Mein Zensurendurchschnitt insgesamt: etwa _____ in Informatik: etwa _____

Meine Hobbies: _____

Mein Berufswunsch: _____

Treffen die Aussagen auf Sie zu? (bitte ankreuzen)	trifft eher zu		trifft eher nicht zu	
	allgemein	für Informatik	allgemein	für Informatik
Ich arbeite sorgfältig.				
Ich bin fleißig				
Ich bin ehrgeizig.				
Ich bin kreativ.				
Ich bin vielseitig interessiert.				
Ich plane meine Arbeit im Voraus.				
Bei Problemen gebe ich schnell auf.				
Ich arbeite lieber in einer Gruppe als alleine.				
Ich arbeite lieber selbstständig als angeleitet.				
Ich arbeite gerne im gleichen Tempo wie alle anderen.				

zu meinen Vorkenntnissen in Informatik (vor Beginn des Oberstufenunterrichts):

Ich habe Computer benutzt zum .... (bitte ankreuzen)	trifft zu
Schreiben von Texten.	
Surfen im Internet.	
Chatten im Internet.	
Musik hören / Musik aus dem Netz laden. (Unzutreffendes ggf. streichen)	
EMail	
Gestaltung von Grafiken.	
Gestalten eigener HTML-Seiten.	
Programmieren in _____	
und _____	

Weshalb haben Sie Informatik gewählt?

Treffen die Aussagen auf Sie zu? (bitte ankreuzen)	trifft zu
Ich erhoffe mir gute Zensuren.	
Mich interessiert das Fach.	
Ich erhoffe mir bessere Berufschancen.	
Mich interessiert .. Programmieren.	
.. Hardware.	
.. theoretische Informatik.	
.. selbstständiges Arbeiten mit komplexen Fragestellungen und Daten.	
sonstige Gründe:	

zu meinen Erwartungen an den Unterricht:

Treffen die Aussagen auf Sie zu? (bitte ankreuzen)	trifft zu	
	all-gemein	für In-forma-
Der Unterricht soll vom Lehrer klar in Einzelschritten geführt werden.		
Der Unterr. soll größere Abschnitte enthalten, in denen ich selbstständig Aufg. bearbeite.		
Ich möchte mir selbst Aufgaben wählen, die in den Unterricht eingehen.		
Die Unterrichtsinhalte sollen nur vom Lehrer vorgegeben werden.		
Die Unterrichtsinhalte sollen sich auch aus den bearbeiteten Problemen ergeben.		
Alle Inhalte sollen im Unterricht eingeführt und dort vollständig behandelt werden.		
Ich möchte mir Inhalte auch selbst erschließen, z. B. aus Büchern oder aus dem Internet.		
Aufgaben und Lösungsanforderungen sollen im Detail eindeutig definiert sein, so dass ich genau weiß, was von mir erwartet wird.		
Aufgabenstellungen sollen so offen sein, dass ich eigene Ideen entwickeln und umsetzen kann.		
Mein Interesse am Fach endet nach dem Unterricht und der Bearbeitung der Aufgaben.		
Ich möchte im Unterricht Anregungen gewinnen, die ich über den Unterricht hinaus verfolgen will.		
Ich möchte einen möglichst breiten Überblick über fachliche Inhalte und Methoden erhalten, der dafür nicht besonders tief gehen muss.		
Ich möchte weniger Stoff behandeln, mich dafür aber exemplarisch mit einzelnen Problemen sehr gründlich auseinandersetzen.		

Falls Sie schon Informatikunterricht hatten:

Wie schätzen Sie die folgenden Aussagen ein. (bitte ankreuzen)	trifft zu	trifft nicht zu
Informatik ist leichter als andere Fächer.		
Informatik ist schwerer als andere Fächer.		
Ich arbeite im Informatikunterricht selbständiger als in anderen Fächern.		
Ich lerne im Informatikunterricht viel, verglichen mit anderen Fächern.		
Ich lerne im Informatikunterricht wenig, verglichen mit anderen Fächern.		
Meine Informatikzensuren sind besser als mein Zensurendurchschnitt.		
Ich würde das Fach wieder wählen.		

In den anderen Fächern entfielen die Fragen zu den informatikspezifischen Vorkenntnissen.

Nachwort

In dieser Arbeit habe ich versucht, die Rahmenbedingungen zu beschreiben, unter denen ein fachlich fundierter, aber vorrangig an der Allgemeinbildung orientierter Informatikunterricht möglich ist. Das Ergebnis lässt sich knapp beschreiben:

> *Die fachlichen Inhalte müssen geeignet ausgewählt und so tief gehend unterrichtet werden, dass die Lernenden in der Lage sind, problemorientiert und aktiv auf dem jeweiligen Teilgebiet zu arbeiten. In dieser selbstständigen Schülerarbeit verwirklicht das Fach dann seine allgemein bildenden Ziele.*

Verfehlt der Unterricht das Ziel, seinen Schülerinnen und Schülern den Computer als ein Werkzeug zugänglich zu machen, mit dem sie ihre eigenen Ideen realisieren können, so folgen daraus nicht nur fachliche Probleme, sondern auch und gerade die allgemein bildenden Ziele sind dann kaum noch zu erreichen. Erforderlich sind also Unterrichtende, die einerseits fachlich hinreichend ausgebildet wurden – wie in anderen Fächern auch –, und die andererseits genügend eigene Erfahrungen haben, um den Lernenden *bei deren Arbeit* eine Hilfe zu sein.

Der Konstruktivismus, der mir nicht nur wegen seiner quantentheoretischen Wurzeln und der Ähnlichkeit zur evolutionären Passung sympathisch ist[458], wurde in dieser Arbeit „pragmatisch“ als Lerntheorie betrachtet. In dieser Form hat er die angenehme Wirkung, das Interesse auf die Adressaten des Unterrichts, die Schülerinnen und Schüler, zu fokussieren. *Deren Lernfortschritte* sind mir wichtig – und nicht irgendwelche Systematiken, die ohne eine solche Priorität formuliert worden sind. Die Resultate konstruktivistischer Überlegungen nehmen *beide* am Unterricht beteiligten Parteien in die Pflicht, und sie definieren sehr klar auch die *Unterkante* des Akzeptablen für beide Seiten: Die Lernenden müssen aktiv an ihrem Lernfortschritt arbeiten. Ohne eine solche Bereitschaft ist der Unterricht sinnlos. Von den Lehrenden muss verlangt werden, dass sie die Schülerinnen und Schüler als Individuen sehen und diese auch so behandeln. Eine Präsentation des „Stoffs“ ohne Bezug zu dem subjektiven Kenntnis- und Erfahrungsstand der Lernenden ist auf die Dauer gesehen wirkungslos und nicht zu akzeptieren. In der Schulrealität sind diese beiden Mindestanforderungen deutlich spürbar, denn die etwas älteren Schülerinnen und Schüler der Sekundarstufen scheitern in der Regel nicht an intellektuellen Defiziten, sondern an fehlender Lernbereitschaft, und Unterricht erreicht die Lernenden nicht mehr, wenn es misslingt, die Brücke zur Lebenswelt zu schlagen. In Wahlfächern wie der Informatik wird dieses durch „Abwählen“ dann sehr deutlich.

Die Schulinformatik ist in der bemerkenswerten Lage, seit ca. 25 Jahren als abiturrelevantes Fach Grund- und vereinzelt Leistungskurse anzubieten, mündliche und schriftliche Reifeprüfungen abzunehmen – und trotzdem immer noch über keine einheitlichen Standards und (von Ausnahmen abgesehen) über keine fachlich ausgebildeten Lehrkräfte zu verfügen. Das Fehlen von Informatiklehrerinnen und –lehrern hat natürlich damit zu tun, dass nach der Einstellungswelle der 70er-Jahre kaum noch Kolleginnen und Kollegen in die Schulen kamen[459]. Es hat aber auch damit zu tun, dass die Ergebnisse der universitären Fachdidaktiken die Berufspraktiker kaum erreichen. Offensichtlich wird die Ausbildung noch immer nach dem Verlassen der Studienseminare als abgeschlossen angesehen, lebenslanges Lernen findet kaum statt. Das kann nicht allein als Vorwurf an die Kollegien genommen werden, denn Fortbildung ist ja nicht (nur) deren Privatsache. Es fehlen offensichtlich die Rahmen-

[458] sondern besonders wegen seines Verzichts auf absolute Wahrheiten.
[459] und da gab es ja noch kaum eine „Informatik“

bedingungen, unter denen systematisch Innovationen aus den Universitäten in die Schulen gelenkt werden.

Der derzeitige rapide Generationswechsel bei den Unterrichtenden kann zusammen mit sinkenden Schülerzahlen dazu führen, dass nach einer Übergangszeit die vorhandenen Stellen besetzt sind und die dann jüngeren Kolleginnen und Kollegen wiederum kaum Einstellungschancen haben. Die „Pädagogen-Welle“ der 70er wird wohl etwas flacher, aber im Prinzip ähnlich reproduziert werden. Da die derzeit eingestellten Kolleginnen und Kollegen in der Regel zwar Anwenderkenntnisse besitzen, aber nur selten ausgebildete Informatiker sind, wird so der Bedarf an einer berufsbegleitenden Lehrerweiterbildung im Bereich der Informatik weiter bestehen. Auch wenn diese ein Hochschulstudium nicht ersetzen kann, muss m. E. die Hochschule trotzdem darin eingebunden werden, um wenigstens die fachlichen Mindeststandards zu halten. Dafür sind klare Spielregeln vonnöten, um die Reibungsverluste zu minimieren: Einerseits muss klar sein, was im Informatikunterricht dauerhaft zu unterrichten ist, andererseits müssen in diesem Rahmen Aktualisierungen möglich sein, und auch die können m. E. überwiegend nur von Universitäten geliefert werden. Gerade hierfür habe ich in dieser Arbeit versucht, geeignete Kriterien zu finden.

Was bleibt also zu tun?

Aus konstruktivistischer Sicht ist zu prüfen, ob die hier entwickelten Ideen und deren Konsequenzen „passen“, also einer kritischen Überprüfung in der Unterrichtswirklichkeit standhalten. Das tun sie im Einzelfall sicherlich. Sind sie aber auch als Standardmodell tauglich – besonders in ihrem Anspruch auf selbstständiges Arbeiten? In der „virtuellen Lehrerweiterbildung Informatik in Niedersachsen VLIN“[460] wird versucht, entsprechenden Unterricht zu verbreiten – so gut es unter den ziemlich beschränkten Bedingungen halt geht. Ergebnisse über die Auswirkungen werden aber erst in einigen Jahren vorliegen. Weiterhin wäre das Auswahlverfahren für Unterrichtsinhalte soweit anzuwenden, dass sich ein Kanon von Unterrichtsbeispielen für eine ganze Kursfolge ergibt – und diese wäre dann zu evaluieren.

Der von mir als Beispiel herangezogene Bereich der technischen und theoretischen Informatik deckt in dieser Form zwei Halbjahre Unterricht ab. An den zentralen Inhalten dieser Gebiete wird sich mittelfristig kaum etwas ändern. Gesucht sind aber immer wieder neue, interessante und vor dem Hintergrund der fachlichen Entwicklung aktuelle Beispiele. Nehmen wir als drittes Halbjahr einen Kurs über „Datenstrukturen“, dann haben wir ein weiteres dauerhaftes Thema gefunden. Listen, Stapel, Bäume, Netze und ihre Anwendungen sind unabhängig von der Realisierung eine unerschöpfliche Quelle für eigenständige Schülerarbeiten. Ein vierter Kurs über Dateien, Datenbanken und Simulationen rundet den fachlichen Kanon der Kursstufe ab. Wählen wir für den Einführungsunterricht in die Algorithmik die erprobten und gut dokumentierten Themen der statischen und bewegten Grafik, Zeichenkettenverarbeitung und zweidimensionale Felder, z. B. als Pixelbilder, dann hat sich in den vergangenen Jahren inhaltlich kaum etwas geändert – außer bei den Werkzeugen. Für so einen Unterricht ist m. E. eine nachhaltige Lehreraus- und -weiterbildung effizient möglich. Wird diese durch eine kontinuierliche Materialienentwicklung unterstützt, dann sollte auch die Informatik für „normal“ engagierte Lehrerinnen und Lehrer unterrichtbar sein – also ohne ungebührliche Anforderungen zu stellen.

In diesem Sinne: Packen wir es an!

[460] Materialien findet man unter *www.vlin.de*.

Literaturverzeichnis

[Abe93] Abelson, H. / Sussman, G.: Struktur und Interpretation von Computerprogrammen
Springer 1993

[Alb83] Albert, J. / Ottmann, Th.: Automaten, Sprachen und Maschinen für Anwender
BI 1983

[Amb91] Ambros, Wolfgang: Didaktik und Anti-Didaktik
LOG IN 4, 1991

[Arg94] Argyris, J. / Faust, G. / Haase, M.: Die Erforschung des Chaos
Vieweg 1994

[Bau90] Baumann, Rüdeger: Didaktik der Informatik
Klett 1990

[Bau91] Baumann/Koerber: Informatik in der Schule der 90er Jahre
LOG IN 6, 1991

[Bau92] Baumann, Rüdeger: Informatik für die Sekundarstufe II, Bd. 1
Klett, 1992

[Bau93] Baumann, Rüdeger: Informatik für die Sekundarstufe II, Bd. 2
Klett, 1993

[Bau94] Baumann, Rüdeger: Algorithmen – wozu?
LOG IN 4, 1994

[Bau95] Baumann, Rüdeger: Probleme des Anfangsunterrichts
LOG IN 1,1995

[Bau96a] Baumann, Rüdeger: Didaktik der Informatik
Klett 1996

[Bau96b] Baumann, Rüdeger: Informatik für die Sekundarstufe II, Bd. 1/2
Klett 1996/97

[Bau98] Baumann/Wegner: Brauchen wir eine Bildungsinformatik?
LOG IN 2, 1998

[BeB02] Beats Biblionetz
http://beat.doebe.li/bibliothek/, 2002

[Ben95] Bender, Peter: Wo im Fächerkanon der allgemeinbildenden Schule soll die Informatik angesiedelt werden?
in [His95]

[Ben02] Ben-Ari, Mordechai: Constructivism in Computer Science Education
Journal of Computers in Mathematics and Science Teaching, in press, 2002

[Ber96] Bernstein, H., PC-Elektronik-Labor Bd. 1 bis 4
Franzis 1996/97

[Ber98] Berger, Peter: Informatische Weltbilder
LOG IN 3/4, 1998

[Boy00] Boyle, Tom: Constructivism: A Suitable Pedagogy for Information and Computing Sciences?
http://www.ics.ltsn.ac.uk/pub/conf2000/Papers/www.unl.ac.uk/simt/aim/boyle/boylecv.htm, 2000

[Bra93] Braitenberg, Valentin: Experimente mit kybernetischen Wesen
Rowohlt 1993

[Bra95] Brauer/Brauer: Informatik – das neue Paradigma
LOG IN 4, 1995

[Bra97] Brandl, Werner: Lernen als „konstruktiver" Prozess: Trugbild oder Wirklichkeit?
schulmagazin 5 bis 10, Heft 5/1997

[Bro69] Brockhaus Enzyklopädie, 17. Auflage
Brockhaus 1969

[Bru67] Bruner, Jerome: Toward a Theory of Instruction
Harvard Press 1967

[Bru70] Bruner, Jerome: Der Prozeß der Erziehung
Schwann 1970

[Bur94] Burkert, Jürgen: Umorientierung des Informatikunterrichts
LOG IN 4, 1994

[Bus87] Bussmann, H. / Heymann, H.-W.: Computer und Allgemeinbildung
Neue Sammlung 1 1987

[Con94] Convey, P. / Highfield, R.: Der Pfeil der Zeit in der Selbstorganisation des Lebens
Rowohlt 1994

[Cra88] Cramer, Friedrich: Chaos und Ordnung, die komplexe Struktur des Lebendigen
Deutsche Verlagsanstalt 1988

[Dah00] Dahlke, M. / Rosenthal, K.-H.: Konstruktivistische Didaktik
http://www.konstruktivistische-didaktik.de/start.htm, 2000

[Dan00] Daniels / Berglund / Petre: Some thoughts on international projects in the undergraduate education
http://www.docs.uu.se/~matsd/EPCoS.html, 2000

[Dav94] Davis, Philip J. und Hersh, Reuben : Erfahrung Mathematik
Birkhäuser 1994

[DLP02] Disney Learning Partnership: Constructivism as a Paradigm for Teaching and Learning
http://www.thirteen.org/edonline/concept2class/month2/explor_sub1.html, 2002

[Dob97a] Doberenz/Kowalski: Borland Delphi 3 für Einsteiger und Fortgeschrittene
Hanser 1997

[Dob97b] Doberenz/Kowalski: Borland Delphi 3 für Profis
Hanser 1997

[Dob98] Doberenz/Druckenmüller: Java
Hanser 1998

[Dor99] Dorninger, Christian: Neue Medien und der Konstruktivismus
http://paedpsych.jk.uni-linz.ac.at:4711/LEHRTEXTE/Lehrtexte.html, 1999

[Dud97] Schüler Duden Informatik
Meyer 1997

[Eb398] Eberle, Franz: Das Potential des Internet für Aus- und Weiterbildung: Mythos und Realität
Informatik/Informatique 6/98

[Eig75] Eigen, Manfred / Winkler, Ruthild: Das Spiel
Piper 1975

[Ele97] Electronics Workbench: Handbuch u. Technische Referenz
Interactiv Image Technologies 1997

[Ent96] Entsminger, G.: The Way of Delphi
Prentice Hall 1996

[Ess92] Essl, Karlheinz: Kompositorische Konsequenzen des Radikalen Konstruktivismus
http://www.essl.at/bibliogr/rad-konstr.html#fn1, 1992

[Fel97] Feldmann/Prüssel: Logik-Simulation mit dem PC
Franzis 1997

[Fla96] Flanagan, David: Java in a Nutshell
O'Reilly 1996

[For97] Forneck, Hermann: Eine neue Konzeption informationstechnischer Allgemeinbildung
LOG IN 6, 1997

[Fri95] Friedrich, Steffen: Grundpositionen eines Schulfaches
LOG IN 5/6, 1995

[Fri96] Friedrich/Schubert/Schwill: Informatik in der Schule – ein Fach im Wandel
LOG IN 2, 1996

[Gas92] Gasper / Leiß / Spengler / Stimm: Technische und theoretische Informatik
BSV 1992

[Ger95] Gerhard, M. / Schuster, H.: Das digitale Universum - Zelluläre Automaten als Modelle der Natur
Vieweg 1995

[GI01] Gesellschaft für Informatik: Empfehlungen für ein Gesamtkonzept zur informatischen Bildung an allgemein bildenden Schulen
Beilage LOG IN 2, 2001

[Gle90] Gleick, James: Chaos
Knaur 1990

[Hei 68] Heimann, P. / Otto, G. / Schulz, W.: Unterricht – Analyse und Planung
Schroedel 1968

[Hen96] Henze, N. / Nejdl, W.: Constructivism in Computer Science Education: Evaluating a Teleteaching Environment for Project Oriented Learning
http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/1998

[Hen98] Henn, Hans-Wolfgang: Warum manchmal Katzen vom Himmel fallen …
in [His00]

[Her74] Herschel, Rudolph: Einführung in die Theorie der Automaten, Sprachen und Algorithmen
Oldenbourg 1974

[Her94] Herget, Wilfried: Ziele und Inhalte des Informatikunterrichts – zum Vergleich
in [His94]

[Her98] Herget, Wilfried: Modellbildung einmal anders – Beispiele zwischen Informatik und Mathematik
in [His00]

[Her99] Hermes/Schumacher: Arbeitshefte Informatik Java
Klett 1999

[Hey95] Heymann, Hans-Werner: Zielsetzungen eines künftigen Mathematik- und Informatikunterrichts
in [His95]

[His91] Hischer, Horst: Neue Technologien als Anlass einer erneuten Standortbestimmung für den Mathematikunterricht, 1991
http://hischer.de/uds/forsch/publikat/hischer/artikel/madid91/index.htm

[His94] Hischer, Horst / Weiss, Michael (Hrsg.): Mathematikunterricht und Computer
Franzbecker 1995

[His95] Hischer, Horst / Weiss, Michael (Hrsg.): Fundamentale Ideen
Franzbecker 1995

[His00] Hischer, Horst (Hrsg.): Modellbildung, Computer und Mathematikunterricht
Franzbecker 2000

[Hof79] Hofstadter, Douglas R.: Gödel, Escher, Bach
Klett-Cotta 1979

[Hof91] Hofstadter, Douglas R.: Metamagicum
Greif 1991

[Hop96] Hoppe/Luther: Informatik und Schule
LOG IN 1, 1996

[Hub97] Hubwieser/Broy: Ein neuer Ansatz für den Informatikunterricht am Gymnasium
LOG IN 3/4, 1997

[Hub00] Hubwieser, Peter: Didaktik der Informatik
Springer 2000

[Hum95] Humberger, J. / Reichel, H.-Ch.: Fundamentale Ideen der angewandten Mathematik
BI 1995

[Hum00] Humbert, Ludger: Umsetzung von Grundkonzepten der Informatik zur fachlichen Orientierung im Informatikunterricht
www.informatica-didactica.de/InformaticaDidactica/Issue1/Humbert, 2000

[Jac87] Jacobs, Konrad: Resultate – Ideen und Entwicklungen in der Mathematik, Bd.1
Vieweg 1987

[Kip98] Kippenhahn, Rudolph: Verschlüsselte Botschaften
Rowohlt 1998

[Kla76] Klafki, Wolfgang: Aspekte kritisch-konstruktiver Erziehungswissenschaft
Beltz 1976

[Kla85] Klafki, Wolfgang: Neue Studien zur Bildungstheorie und Didaktik
Beltz 1985

[Kla96] Klafki, Wolfgang: Neue Studien zur Bildungstheorie und Didaktik
Beltz 1996

[Kla98a] Klafki, Wolfgang: Grundzüge kritisch-konstruktiver Erziehungswissenschaft
Script 1998

[Kla98b] Klafki, Wolfgang: Selbsttätigkeit als Grundprinzip des Lernens in der Schule
Script 1998

[Kla98c] Klafki, Wolfgang: Kriterien einer guten Schule
Script 1998

[Koe95] Köhler, Hartmut: Computereinsatz: Krämergeist, Unreife und Vermessenheit zugleich – Gedanken zur Allgemeinbildung, in [His95]

[Koe00] Koerber/Peters: Informatische Bildung in Deutschland – Die Wurzeln der Zukunft
LOG IN 2, 2000

[Kop65] Kopp, Ferdinand: Didaktik in Leitgedanken
Auer 1965

[Krü00] Krüger, Guido: GoTo Java 2
Addison-Wesley 2000

[Küc00] Küchlin, W. / Weber, A.: Einführung in die Informatik, Objektorientiert mit Java
Springer 2000

[Lav98] v. Lavergne, Harro: Thesen zur aktuellen Orientierungslosigkeit
LOG IN 5, 1998

[LeB01] LeBlanc, Mark: Ethical Issues in Computing
http://www2.wheatonma.edu/academic/academicdept/MathCS/faculty/mleblanc, 2001

[Leh92] Lehmann, Gabriele: Ziele im Informatikunterricht
LOG IN 1, 1992

[Lei92] Leiß/Spengler/Stimm: Technische und theoretische Informatik
BSV 1992

[Lem00] Lemay, Laura: Java 2 Platform
SAMS 2000

[Mag00] Magenheim, Johann S.: Informatiksystem und Dekonstruktion als didaktische Kategorien, Skript 2000

[Man91] Mandelbrot, Benoît: Die fraktale Geometrie der Natur
Birkhäuser 1991

[Mat95] Matcho/Faulkner: Using Delphi
Que 1995

[Mic98] Microsoft Visual J++ 6.0 Programmierhandbuch
Microsoft Press 1998

[Mod87] Modrow, Eckart: BCD-Ziffernerkenner
LOG IN 7, 1987

[Mod91] Modrow, Eckart: Zur Didaktik des Informatik-Unterrichts, Band 1
Dümmler 1991

[Mod92a] Modow, Eckart: Zur Didaktik des Informatik-Unterrichts, Band 2,
Dümmler 1992

[Mod92b] Modrow, Eckart: Automaten-Schaltwerke-Sprachen
Dümmler 1992

[Mod93] Modrow, Eckart: Datenbankaspekte im Informatikunterricht
Computer und Unterricht 10, 1993

[Mod94] Modrow, Eckart: Anfangsunterricht mit Mäusen, Knöpfen, Schaltern und anderen „Spielzeugen“, INFORMATIK betrifft uns 2, 1994

[Mod95a] Modrow, Eckart: Physikobjekte im Informatikunterricht
INFORMATIK betrifft uns 1, 1995

[Mod95b] Modrow, Eckart: Ansatzpunkte zu Änderungen im Mathematikunterricht aus Sicht der Informatik
in [His95]

[Mod96a] Modrow, Eckart: Dateien-Datenbanken-Datenschutz
Dümmler 1996

[Mod96b] Modrow, Eckart: Von Bällen, Billardtischen und den Grenzen der Simulation
INFORMATIK betrifft uns 3, 1996

[Mod98] Modrow, Eckart: Informatik mit Delphi, Band 1
Dümmler 1998

[Mod00] Modrow, Eckart: Informatik mit Delphi, Band 2
Dümmler 2000

[Mon98] Monnerjahn, Rolf: Modellbildung und Simulation – durch Nachahmung zum Verständnis?
in [His00]

[Mül98] Müller, Klaus: Warum bleibt Wissen träge? Ein Standpunkt zum Konstruktivismus im Unterricht
http://idw-online.de/public/zeige_einrichtung.html?eid=105, 1998

[Neu98] Neupert, Pohlmann, Schubert: Positionen zur informatischen Bildung an deutschen Schulen, MNU/GI 1998

[Nie02] Nievergelt, Jürg: Folien zur Vorlesung
http://www.tedu.ethz.ch/didaktik/id1_material.html, 2002

[Nöb79] Nöbauer, W. / Timischl, W.: Mathematische Modelle in der Biologie
Vieweg 1979

[Pei98a] Peitgen, Heinz-Otto: Bausteine des Chaos
Rowohlt 1998

[Pei98b] Peitgen, Heinz-Otto: Bausteine der Ordnung
Rowohlt 1998

[Pen91] Penrose, Roger: Computerdenken
Spektrum Verlag 1991

[Pet88] Peterson, Ivars: Mathematische Expeditionen
Spektrum Verlag 1988

[Ple98] Pleil, Thomas: Konstruktivismus im Unterricht: Warum bleibt Wissen träge?
http://idw-online.de/public/zeige_einrichtung.html?eid=105, 1998

[Pos91] Postman, Neil: Das Technopol
Fischer 1991

[Pot98] Potts, Steve: Java 1.2 How-To
SAMS 1998

[Pri81] Prigogine, I. / Stengers, I.: Dialog mit der Natur
Piper 1981

[Rec97] Rechenberg, Peter: Quo vadis Informatik?
LOG IN 1, 1997

[Rei95] Reiser/Wirth: Programmieren in Oberon
Addison-Wesley 1995

[Rei00] Reich, Kersten: Konstruktivismus
http://www.uni-koeln.de/ew-fak/Paeda/hp/reich/index.html, 2000

[Sche93] Schefe, P. / Hastedt, H. / Dittrich, Y. / Keil, G. (Hrsg.): Informatik und Philosophie
BI 1993

[Sche97] Schelhowe, Heide: Auf dem Weg zu einer Theorie der Interaktion?
LOG IN 5, 1997

[Schm71] Schmitt, Alfred: Automaten – Algorithmen - Gehirne
Suhrkamp 1971

[Schö95] Schöning, Uwe: Theoretische Informatik – kurzgefasst
Spektrum Verlag 1995

[Schr91] Schroeder, Manfred: Fraktale, Chaos und Selbstähnlichkeit
Spektrum 1991

[Schu99] Schubert, Sigrid: Begleitmaterial zur Vorlesung Einführung in die Didaktik der Informatik WS 199/2000, Script 1999

[Schu01] Schulte, Carsten: Vom Modellieren zum Gestalten – Objektorientierung als Impuls für einen neuen Informatikunterricht?
http://www.informatica-didactica.de/InformaticaDidactica/Schulte2001.htm, 2001

[Schw93a] Schwill, Andreas: Fundamentale Ideen in Mathematik und Informatik
Script 1993

[Schw93b] Schwill, Andreas: Objektorientierte Programmierung
LOG IN 4, 1993

[Schw93c] Schwill, Andreas: Programmierstile
LOG IN 4, 1993

[Schw95] Schwill, Andreas: Fundamentale Ideen in Mathematik und Informatik
in [His95]

[Schw96] Schwill, Andreas: Vorlesungen zur Didaktik der Informatik
Script 1996/97

[Schw00] Schwill, Andreas: Didaktisch-methodische Ansätze der Informatikausbildung
Script 2000

[Sha76] Shannon, C./ Weaver, W.: Mathematische Grundlagen der Informationstheorie
Oldenbourg 1976

[Sta98] Standish, Thomas: Data Structures in Java
Addison-Wesley 1998

[Ste01] Steelman report: Computing Curricula 2001 – Computer Science
The Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery
http://www.acm.org/sigsec/cc2001/steelman, 2001

[Tho98] Thomas, Marco: Nebenläufigkeit im Schulfach Informatik
Script Potsdam 1998

[Thi97] Thissen, Frank: Das Lernen neu erfinden – konstruktivistische Grundlagen einer Multimedia-Didaktik
LEARNTEC 97, Tagungsband, 1997

[Tho00] Thomas, Marco: Einführung in die Didaktik der Informatik
Begleitmaterial zur Vorlesung, Potsdam 2000/01

[Tuc96] Tucker, Alan B.: Strategic Directions in Computer Science Education
ACM Computing Surveys 28(4), December 1996

[Vol85] Volpert, Walter: Zauberlehrlinge
Beltz 1985

[Vos00] Vossen/Witt: Grundlagen der theoretischen Informatik mit Anwendungen
Vieweg 2000

[Wag80] Wagenschein, Martin: Naturphänomene sehen und verstehen
Klett 1980

[Wäl99] Wällnitz, Elke: Vorlesung Didaktik der Informatik
Script Chemnitz 1999

[Wei85] Weizsäcker, Carl Friedrich v.: Aufbau der Physik
Hanser 1985

[Wei86] Weizsäcker, Carl Friedrich v.: Die Einheit der Natur
DTV 1986

[Wer98] Werning, Rolf : Konstruktivismus. Eine Anregung für die Pädagogik?
Pädagogik 7-8/98

[Wir84] Wirth, Niklaus: Compilerbau
Teubner 1984

[Wol97] Wolff, Dieter: Lernstrategien: Ein Weg zu mehr Lernerautonomie
http://www.ualberta.ca/~german/idv/wolff1.htm, 1997

Materialien 1: Java-Quelltext zu Beispiel 1

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Applet1 extends Applet implements ActionListener
{
   // Oberflächenelemente
   Label lanf        = new Label("Anfangszeichen:");
   Label lein        = new Label("Eingabe:");
   Label lversch     = new Label("Verschlüsselt:");
   Label lentsch     = new Label("Entschlüsselt:");
   Button bversch    = new Button("verschlüsseln!");
   Button bentsch    = new Button("entschlüsseln!");
   TextField tanf    = new TextField(1);
   TextField tein    = new TextField(40);
   TextField tversch = new TextField(40);
   TextField tentsch = new TextField(40);

   char zustand = 'B'; //Elemente des Automaten: aktueller Zustand

   private char u(char e) //Überführungsfunktion
   {
      return e;
   }

   private char g(char e) //Ausgabefunktion
   {
      int neu = (charToInt(zustand) ^ charToInt(e));
      return intToChar(neu);
   }

   private int charToInt(char c)   //wandelt Zeichen in Zahlen, 'A'--> 0
   {
      return (int)c - (int)'A';
   }

   private char intToChar(int i)   //wandelt Zahlen in Zeichen
   {
      char c = (char)(i + (int)'A');
      return c;
   }

   public void init() //konfiguriert die Oberfläche
   {
      setLayout(null);
      lanf.setBounds(10,10,80,20); add(lanf);
      tanf.setBounds(90,10,15,20); add(tanf);
      lein.setBounds(10,50,80,20); add(lein);
      tein.setBounds(90,50,400,20); add(tein);
      lversch.setBounds(10,90,80,20); add(lversch);
      tversch.setBounds(90,90,400,20); add(tversch);
      lentsch.setBounds(10,130,80,20); add(lentsch);
      tentsch.setBounds(90,130,400,20); add(tentsch);
      bversch.addActionListener(this);
      bversch.setBounds(200,170,80,20); add(bversch);
      bentsch.addActionListener(this);
      bentsch.setBounds(300,170,80,20); add(bentsch);
      tanf.setText("B");
   }
```

```
  public void actionPerformed(ActionEvent e) //Verteiler auf die Unterprogramme.
  {
    if(e.getActionCommand().equals("verschlüsseln!"))
       verschluessele();
    else entschluessele();
  }

  private void verschluessele() //verschlüsselt die Eingabe
  {
    String eingabe,ausgabe,anfang;

    eingabe =  tein.getText();
    ausgabe = "";
    anfang = tanf.getText();
    if (anfang.length()<1)
    {
        zustand = 'B';
        tanf.setText("B");
    }
    else zustand = anfang.charAt(0);
    for(int i=0;i<eingabe.length();i++)
    {
        char c  = eingabe.charAt(i);
        ausgabe = ausgabe + g(c);
        zustand = u(c);
    }
    tversch.setText(ausgabe);
  }

  private void entschluessele() //entschlüsselt
  {
    String eingabe,ausgabe,anfang;

    eingabe =  tversch.getText();
    ausgabe = "";
    anfang = tanf.getText();
    if (anfang.length()<1)
    {
        zustand = 'B';
        tanf.setText("B");
    }
    else zustand = anfang.charAt(0);
    for(int i=0;i<eingabe.length();i++)
    {
        char c  = eingabe.charAt(i);
        char a  = g(c);
        ausgabe = ausgabe + a;
        zustand = u(a);
    }
    tentsch.setText(ausgabe);
  }
}
```

Materialien 2: Delphi-Quelltexte zu Beispiel 3

```
unit uZellulaererAutomat;  //Steuerprogramm

interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, ExtCtrls, StdCtrls;

type  TForm1 = class(TForm)
      Panel1: TPanel;
      Ende: TButton;
      Start: TButton;
      Label1: TLabel;
      gAnzeige: TEdit;
      Stop: TButton;
      Neustart: TButton;
      Timer1: TTimer;
      procedure EndeClick(Sender: TObject);
      procedure StartClick(Sender: TObject);
      procedure StopClick(Sender: TObject);
      procedure NeustartClick(Sender: TObject);
      procedure Timer1Timer(Sender: TObject);
      procedure FormCreate(Sender: TObject);
      end;

var Form1: TForm1;

implementation
uses uWelt;
{$R *.dfm}

const dt = 10; //Zeitintervall für die Timer-gesteuerte Simulation

var generation: integer = 0;

procedure init; //initialisiert das System
begin
generation := 0;
Form1.gAnzeige.text := IntToStr(generation);
welt.init;
end;

procedure TForm1.EndeClick(Sender: TObject);  //Programmende
begin
halt
end;

procedure TForm1.StartClick(Sender: TObject);  //Simulation starten
begin
Timer1.Interval := dt;
end;

procedure TForm1.StopClick(Sender: TObject);  //Simulation stoppen
begin
Timer1.Interval := 0;
end;

procedure TForm1.NeustartClick(Sender: TObject);  //neue Simulation
begin
init
end;
```

```
procedure TForm1.Timer1Timer(Sender: TObject);   //Aufruf einer Aktionskette in der „Welt“
begin
welt.handle;
generation := generation + 1;
Form1.gAnzeige.text := IntToStr(generation);
end;

procedure TForm1.FormCreate(Sender: TObject);   //einmal die „Welt“ erzeugen
begin
welt := twelt.create(Form1);
init
end;

initialization
randomize;

end.

(*-------------------------------------------------------------------------*)

unit uAutomat; //Vorlage für die Gitterautomaten

interface
uses Graphics, ExtCtrls, Controls, SysUtils;

type
  tPartner = class(tPanel)   //die Gitterautomaten
    public
    zustand, neuerZustand, punkte, xpos,ypos: integer;
    wasDerNachbarTat                        : array[1..8] of boolean;
    constructor create(aOwner: tWincontrol; x,y: integer);
    procedure init;
    procedure zeigeDich;
    procedure handelMitDenNachbarn;
    procedure wechseleZustand;
    function handelsergebnisMit(p: tPartner; lage: integer): integer;
    function kooperiertMit( lage: integer): boolean;
     end;

implementation
uses uWelt;

constructor tPartner.create(aOwner: tWincontrol; x,y: integer);
begin
inherited create(aOwner);
SetBounds(11*x,11*y+40,10,10);
parent   := aOwner;
name := 'Px'+IntToStr(x)+'y'+IntToStr(y);
caption := '';
xpos := x;
ypos := y;
end;

procedure tPartner.init;
var i: integer;
begin
zustand := random(3)+1;   //Strategie festlegen
neuerZustand := zustand;
for i := 1 to 8 do wasDerNachbarTat[i] := true;
punkte := 0;
zeigeDich
end;
```

```
procedure tPartner.zeigeDich;
begin
case zustand of
  1: color := clGreen;   //der Naive: kooperiert immer
  2: color := clRed;     //der Betrüger: kooperiert nie
  3: color := clYellow; //TitForTat
  end;
show
end;

procedure tPartner.handelMitDenNachbarn;
var xlinks,xrechts,yoben,yunten: integer;
begin
if xpos = 0  //eine "Zylinderwelt" in beiden Richtungen
  then xlinks  := uwelt.xmax
  else xlinks := xpos-1;
if xpos = uwelt.xmax then xrechts  := 0 else xrechts := xpos+1;
if ypos = 0 then yoben  := uwelt.ymax else yoben := ypos-1;
if ypos = uwelt.ymax then yunten  := 0 else yunten := ypos+1;

punkte := handelsergebnisMit(welt.elemente[xlinks,ypos],1);
punkte := punkte + handelsergebnisMit(welt.elemente[xlinks,yoben],2);
punkte := punkte + handelsergebnisMit(welt.elemente[xpos,yoben],3);
punkte := punkte + handelsergebnisMit(welt.elemente[xrechts,yoben],4);
punkte := punkte + handelsergebnisMit(welt.elemente[xrechts,ypos],5);
punkte := punkte + handelsergebnisMit(welt.elemente[xrechts,yunten],6);
punkte := punkte + handelsergebnisMit(welt.elemente[xpos,yunten],7);
punkte := punkte + handelsergebnisMit(welt.elemente[xlinks,yunten],8);
end;

procedure tPartner.wechseleZustand;
var xlinks,xrechts,yoben,yunten,bestPunkte,bestZustand: integer;
begin
if xpos = 0  //eine "Zylinderwelt" in beiden Richtungen
  then xlinks  := uwelt.xmax
  else xlinks := xpos-1;
if xpos = uwelt.xmax then xrechts  := 0 else xrechts := xpos+1;
if ypos = 0 then yoben  := uwelt.ymax else yoben := ypos-1;
if ypos = uwelt.ymax then yunten  := 0 else yunten := ypos+1;

bestPunkte := punkte;   //feststellen, wer von den Nachbarn die meisten Punkte hat
bestZustand := zustand;
if bestPunkte < welt.elemente[xlinks,ypos].punkte then begin
  bestpunkte := welt.elemente[xlinks,ypos].punkte;
  bestZustand := welt.elemente[xlinks,ypos].zustand;
  end;
if bestPunkte < welt.elemente[xlinks,yoben].punkte then begin
  bestpunkte := welt.elemente[xlinks,yoben].punkte;
  bestZustand := welt.elemente[xlinks,yoben].zustand;
  end;
if bestPunkte < welt.elemente[xpos,yoben].punkte then begin
  bestpunkte := welt.elemente[xpos,yoben].punkte;
  bestZustand := welt.elemente[xpos,yoben].zustand;
  end;
if bestPunkte < welt.elemente[xrechts,yoben].punkte then begin
  bestpunkte := welt.elemente[xrechts,yoben].punkte;
  bestZustand := welt.elemente[xrechts,yoben].zustand;
  end;
if bestPunkte < welt.elemente[xrechts,ypos].punkte then begin
  bestpunkte := welt.elemente[xrechts,ypos].punkte;
  bestZustand := welt.elemente[xrechts,ypos].zustand;
  end;
```

```
if bestPunkte < welt.elemente[xrechts,yunten].punkte then begin
  bestpunkte := welt.elemente[xrechts,yunten].punkte;
  bestZustand := welt.elemente[xrechts,yunten].zustand;
  end;
if bestPunkte < welt.elemente[xpos,yunten].punkte then begin
  bestpunkte := welt.elemente[xpos,yunten].punkte;
  bestZustand := welt.elemente[xpos,yunten].zustand;
  end;
if bestPunkte < welt.elemente[xlinks,yunten].punkte then begin
  bestpunkte := welt.elemente[xlinks,yunten].punkte;
  bestZustand := welt.elemente[xlinks,yunten].zustand;
  end;
neuerZustand := bestZustand;   //da zeigt sich der Opportunismus
end;

function tPartner.handelsergebnisMit(p: tPartner; lage: integer): integer;
var inverseLage        : integer;
    derAndereKooperiert: boolean;
    ichKooperiere      : boolean;
begin
case lage of
  1..4: inverseLage := 4+lage;
  5..8: inverseLage := lage-4;
  end;
derAndereKooperiert := p.kooperiertMit(inverseLage); //nachsehen, was der Partner tut
wasDerNachbarTat[lage] := derAndereKooperiert;
ichKooperiere := kooperiertMit(lage);
if ichKooperiere                        //jetzt reagieren
  then if derAndereKooperiert
    then result := random(9)+2
    else result := 0
else
  if derAndereKooperiert
    then result := random(20)
    else result := random(2);
end;

function tPartner.kooperiertMit( lage: integer): boolean;
begin
case zustand of
  1: result := true;                    //der Naive: kooperiert immer
  2: result := false;                   //der Betrüger: kooperiert nie
  3: result := wasDerNachbarTat[lage]; //TitForTat
  end
end;

end.

(*-------------------------------------------------------------------------*)
```

```
unit uWelt;   //die Welt der Gitterautomaten: der zelluläre Automat

interface
uses Graphics,ExtCtrls,Controls,uAutomat;

const xmax = 49; ymax = 49;  //Gittermaße

type
  tWelt = class
    public
    elemente: array[0..xmax,0..ymax] of tPartner;
    constructor create(aOwner: tWincontrol);
    procedure init;
    procedure handle;
    end;

var welt: tWelt;

implementation

constructor tWelt.create(aOwner: tWincontrol);
var i,j: integer;
begin
for i := 0 to xmax do
  for j := 0 to ymax do
    elemente[i,j] := tPartner.create(aOwner,i,j); //das dauert!
end;

procedure twelt.init;
var i,j: integer;
begin
for i := 0 to xmax do
  for j := 0 to ymax do
    elemente[i,j].init;
end;

procedure tWelt.handle;
var i,j: integer;
begin
for i := 0 to xmax do  //alle Automaten treiben Handel mit den Nachbarn
  for j := 0 to ymax do
    elemente[i,j].handelMitDenNachbarn;
for i := 0 to xmax do  //alle Automaten überprüfen die Umgebung
  for j := 0 to ymax do
    elemente[i,j].wechseleZustand;
for i := 0 to xmax do  // und wechseln ggf. ihren Zustand
  for j := 0 to ymax do
    elemente[i,j].zustand := elemente[i,j].neuerZustand;
for i := 0 to xmax do  //und zeigen sich wieder
  for j := 0 to ymax do
    elemente[i,j].zeigeDich;
end;

end.
```

Materialien 3: Delphi-Quelltexte zu Beispiel 4

```
unit uTurtle;
interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, Menus, StdCtrls, ExtCtrls, OleCtrls, SPIRITLib_TLB;

type TLEGO = class(TForm)
       MainMenu1: TMainMenu;
       Datei1: TMenuItem;
       Oeffnen: TMenuItem;
       Speichern1: TMenuItem;
       N1: TMenuItem;
       Ende1: TMenuItem;
       Programm: TMenuItem;
       LEGO: TMenuItem;
       Anzeige: TMenuItem;
       Neu1: TMenuItem;
       Editor: TMemo;
       OpenDialog1: TOpenDialog;
       SaveDialog1: TSaveDialog;
       Editieren: TMenuItem;
       bersetzen1: TMenuItem;
       Ausfhren2: TMenuItem;
       Verbindungstest1: TMenuItem;
       ZielBildschirm1: TMenuItem;
       ZielRCX1: TMenuItem;
       Backup: TMemo;
       helpmemo: TMemo;
       Image1: TImage;
       Befehle: TMemo;
       Quelltext1: TMenuItem;
       Quelltext2: TMenuItem;
       Bild1: TMenuItem;
       Bildlschen1: TMenuItem;
       Spirit1: TSpirit;
       Konstantenangeben1: TMenuItem;
       Bewegungskonstanteeingeben1: TMenuItem;
       Timer1: TTimer;
       procedure Ende1Click(Sender: TObject);
       procedure Neu1Click(Sender: TObject);
       procedure OeffnenClick(Sender: TObject);
       procedure Speichern1Click(Sender: TObject);
       procedure EditierenClick(Sender: TObject);
       procedure bersetzen1Click(Sender: TObject);
       procedure FormCreate(Sender: TObject);
       procedure Ausfhren2Click(Sender: TObject);
       procedure Quelltext1Click(Sender: TObject);
       procedure Quelltext2Click(Sender: TObject);
       procedure Bild1Click(Sender: TObject);
       procedure Bildlschen1Click(Sender: TObject);
       procedure EditorKeyUp(Sender: TObject; var Key: Word;
                                                    Shift:TShiftState);
       procedure Verbindungstest1Click(Sender: TObject);
       procedure ZielBildschirm1Click(Sender: TObject);
       procedure ZielRCX1Click(Sender: TObject);
       procedure Bewegungskonstanteeingeben1Click(Sender: TObject);
       procedure Konstantenangeben1Click(Sender: TObject);
       procedure Timer1Timer(Sender: TObject);
       procedure EditorMouseUp(Sender: TObject; Button: TMouseButton;
                                     Shift: TShiftState; X, Y: Integer);
  end;
```

```
var LEGO: TLEGO;

implementation
{$R *.dfm}

type
tScreenTurtle = class
   x,y : single;
   w,d: integer;
   constructor init(xn,yn: single; dn: integer);
   procedure turn(i: integer);
   procedure move(i: integer);
   procedure show;
   procedure hide;
   end;

tLEGOTurtle = class
   UZeit, dt: integer;
   M1an,M2an: boolean;
   constructor init;
   procedure turn(i: integer);
   procedure move(i: integer);
   function PruefeVerbindung: boolean;
   end;

tKeller = class
   speicherband: string;
   startzeichen: char;
   constructor init(c: char);
   function erstes: char;
   function pull: char;
   procedure push(c: char);
   end;

var
  DateiGespeichert: boolean=false; //Zustand der Quelldatei
  DateiGeaendert: boolean = false;
  DateiUebersetzt: boolean = false;
  Dateiname: string='NeueDatei.logo';
  Ausgabeziel: (RCX,Screen)=Screen; //Umschalten zwischen Bildschirm und LEGO-RCX
  breite,hoehe: integer; //Bildschirmdaten
  turtle: tScreenTurtle; // die beiden Ausgabegeräte
  LTurtle: tLEGOTurtle;

//---------------------------------------------------- Methoden der Screenturtle ----------------------------------------------------
constructor tScreenTurtle.init(xn,yn: single; dn: integer);
begin
inherited create;
x := xn; y := yn; w := 0; d := dn;
show;
end;

procedure tScreenTurtle.Show;
begin
with LEGO.Image1.Canvas do
  begin
  pen.mode := pmNotXOR;
  pen.Color := clBlack;
  brush.color := clRed;
  brush.Style := bsSolid;
  ellipse(round(x-d),round(y-d),round(x+d),round(y+d));
  moveto(round(x),round(y));
```

```
  lineto(round(x+2*d*cos(pi/180*w)),round(y+2*d*sin(pi/180*w)));
  pen.mode := pmCopy;
  end;
end;

procedure tScreenTurtle.hide;
begin show end;

procedure tScreenTurtle.move(i: integer);
begin
hide;
with LEGO.Image1.Canvas do
  begin
  moveto(round(x),round(y));
  x := x + i*cos(Pi/180*w);
  y := y + i * sin(Pi/180*w);
  Lineto(Round(x),round(y));
  end;
show;
end;

procedure tScreenTurtle.turn(i: integer);
begin
hide;
w := (w+i) mod 360;
show;
end;

//---------------------------------------------------- Methoden der LEGOturtle ----------------------------------------------------
constructor tLEGOTurtle.init;
begin
inherited create;
UZeit := 60; dt := 10;
if PruefeVerbindung then LEGO.Spirit1.deletetask(1);
M1an := false; M2an := false;
end;

function tLEGOTurtle.PruefeVerbindung: boolean;
begin
LEGO.Spirit1.InitComm;
if not LEGO.Spirit1.PBAliveOrNot
  then begin
     showmessage('Die Verbindung zum RCX ist unterbrochen!');
     result := false
     end
  else result := true
end;

procedure tLEGOTurtle.move(i: integer);
begin
if not(M1an and M2an) then begin
   LEGO.spirit1.on_('02');
   LEGO.Helpmemo.Lines.add('RCX: On_(''02'')');
   M1an := true;
   M2an := true;
   end
else if not M1an then begin
        LEGO.spirit1.on_('0');
        LEGO.Helpmemo.Lines.add('RCX: On_(''0'')');
        M1an := true;
        end
     else begin
       LEGO.spirit1.on_('2');
```

```
       LEGO.Helpmemo.Lines.add('RCX: On_(''2'')');
       M2an := true;
       end;
LEGO.spirit1.wait(2,round(i*dt*0.01));
LEGO.Helpmemo.Lines.add('RCX: Wait(2,'+IntToStr(i*dt)+')');
end;

procedure tLEGOTurtle.turn(i: integer);
begin
if i < 0 then begin
  if M2an then begin
   LEGO.spirit1.off('2');
   LEGO.Helpmemo.Lines.add('RCX: Off(''2'')');
   M2an := false;
   end;
  if not M1an then begin
   LEGO.spirit1.on_('0');
   LEGO.Helpmemo.Lines.add('RCX: On_(''0'')');
   M1an := true;
   end
  end
else begin
  if M1an then begin
   LEGO.spirit1.off('0');
   LEGO.Helpmemo.Lines.add('RCX: Off(''0'')');
   M1an := false;
   end;
  if not M2an then begin
   LEGO.spirit1.on_('2');
   LEGO.Helpmemo.Lines.add('RCX: On_(''2'')');
   M2an := true;
   end
  end;
LEGO.Spirit1.Wait(2,round(UZeit/360*i));
LEGO.Helpmemo.Lines.add('RCX: Wait(2,'+IntToStr(round(UZeit/360*i))+')');
end;

//------------------------------------------------- Kellermethoden ----------------------------------------------------
constructor tKeller.init(c: char);
begin
inherited create;
speicherband := c;
startzeichen := c;
end;

function tKeller.erstes: char;
begin
if speicherband<>'' then result := speicherband[1] else result := ' '
end;

function tKeller.pull: char;
begin
if speicherband<>'' then begin
  result := speicherband[1];
  delete(speicherband,1,1)
  end
else result := ' '
end;

procedure tKeller.push(c: char);
begin
speicherband := c + speicherband
end;
```

```
//------------------------------------------------- Hilfsmethoden ----------------------------------------------------

//---------------------------------- Kopiert die Zeilen einer Memokomponenten in eine andere ---------------------------
procedure Kopiere(quelle,ziel:tMemo);
var i: integer;
begin
ziel.lines.clear;
for i := 0 to Quelle.lines.count do
   ziel.lines.add(quelle.lines.strings[i]);
end;

//------------------------------------------------- Anfangswerte festlegen -------------------------------------------
procedure TLEGO.FormCreate(Sender: TObject);
begin
Editor.Lines.Clear;
Backup.lines.clear;
breite := image1.Width;
hoehe := image1.Height;
Turtle := tScreenTurtle.init(breite div 2, hoehe div 2,5);
LTurtle:= tLEGOTurtle.init;
end;

//------------------------------------------------- Eventhandler  des Menüs ------------------------------------------

//------------------------------------------------- Neue Datei anlegen -----------------------------------------------
procedure TLEGO.Neu1Click(Sender: TObject);
begin
if not DateiGespeichert and (Editor.lines.count>0) then begin
  if MessageDlg('Wollen Sie Ihr Programm speichern?',
                      mtInformation,[mbYes,mbNo,mbAbort],0)=mrYes
    then begin
      Savedialog1.FileName := Dateiname;
      if Savedialog1.Execute then begin
        Dateiname := Opendialog1.FileName;
        if Dateiname <> '' then begin
          if pos('.logo',Dateiname)=0 then Dateiname := Dateiname+'.logo';
          Editor.lines.SaveToFile(Dateiname);
          DateiGespeichert := true;
          end;
      end
    end
  end;
Editor.Lines.Clear;
Dateiname := 'NeueDatei.logo';
DateiGespeichert := false;
DateiUebersetzt := false;
end;

//------------------------------------------------- Datei öffnen -----------------------------------------------------
procedure TLEGO.OeffnenClick(Sender: TObject);
begin
if not DateiGespeichert and (Editor.lines.count>0) then begin
  if     MessageDlg('Wollen     Sie     Ihr     Programm     speichern?',
mtInformation,[mbYes,mbNo,mbAbort],0)=mrYes
    then begin
      Savedialog1.FileName := Dateiname;
      if Savedialog1.Execute then begin
        Dateiname := Opendialog1.FileName;
        if Dateiname <> '' then begin
          if pos('.logo',Dateiname)=0 then Dateiname := Dateiname+'.logo';
          Editor.lines.SaveToFile(Dateiname);
          DateiGespeichert := true;
```

```
          end;
      end
    end
  end;
if Opendialog1.Execute then begin
  Dateiname := Opendialog1.FileName;
  if Dateiname <> '' then begin
    Editor.lines.LoadFromFile(Dateiname);
    DateiGeaendert := false;
    DateiUebersetzt := false;
    end;
  end
end;

//------------------------------------------------- Datei speichern ----------------------------------------------------
procedure TLEGO.Speichern1Click(Sender: TObject);
begin
Savedialog1.FileName := Dateiname;
if Savedialog1.Execute then begin
  Dateiname := Savedialog1.FileName;
  if Dateiname <> '' then begin
  if pos('.logo',Dateiname)=0 then Dateiname := Dateiname+'.logo';
    Editor.lines.SaveToFile(Dateiname);
    DateiGespeichert := true;
    end;
  end
end;

//------------------------------------------------- Programm beenden ---------------------------------------------------
procedure TLEGO.Ende1Click(Sender: TObject);
begin
if not DateiGespeichert then begin
  if          MessageDlg('Wollen          Sie          Ihr          Programm
speichern?',mtInformation,[mbYes,mbNo,mbAbort],0)=mrYes
    then begin
      Savedialog1.FileName := Dateiname;
      if Savedialog1.Execute then begin
        Dateiname := Opendialog1.FileName;
        if Dateiname <> '' then begin
          if pos('.logo',Dateiname)=0 then Dateiname := Dateiname+'.logo';
          Editor.lines.SaveToFile(Dateiname);
          DateiGespeichert := true;
          end;
      end
    end
  end;
halt
end;

//------------------------------------------------- Programm editieren -------------------------------------------------
procedure TLEGO.EditierenClick(Sender: TObject);
begin
Editor.show;
Helpmemo.Hide;
Image1.Hide;
Befehle.Hide;
if Dateiname ='' then begin
   Editor.Lines.Clear;
   Dateiname := 'NeueDatei.logo';
   DateiGespeichert := false;
   DateiUebersetzt := false;
   end
   else if Backup.Lines.count > 0 then begin
```

```
     Kopiere(Backup,Editor);
     Backup.lines.Clear;
     end;
Editor.show;
end;

//------------------------------------------------- Programm übersetzen ----------------------------------------------
procedure TLEGO.bersetzen1Click(Sender: TObject);
type tZustand =(s0,s1,s2,s3,s4,s5,s6,s7,se,sf);
var s                  : tZustand;
    c                  : char;
    Zeile,fZeile,Befehl: string;
    keller             : tKeller; //Keller
    i,j,fnr            : integer;
    anfang             : boolean;

function u(s: tZustand; c: char): tZustand; //Überführungsfunktion des Parsers
begin
case s of
  s0: case c of
        'W'         : result := s2;
        'V','R','L': result := s1;
        '$'         : result := se;
        else begin
          result := sf;
          fnr := 1;
          end
        end;
  s1: case c of
        '0'..'9'   : result := s7;
        else begin
          result := sf;
          fnr := 2;
          end
        end;
  s2:case c of
        '0'..'9'   : result := s3;
         else begin
          result := sf;
          fnr := 2;
          end
        end;
  s3: case c of
        '0'..'9'   : result := s3;
        '('         : begin
                      keller.push('(');
                      result := s4;
                      end;
        else begin
          result := sf;
          fnr := 3;
          end
        end;
  s4: case c of
        ')'         : begin
                      if keller.erstes ='(' then begin
                        keller.pull;
                        if keller.erstes=keller.startzeichen
                           then result := s0
                        else result := s4;
                        end
                      else begin
```

```
                      result := sf;
                      fnr := 6
                      end
                    end;
        'V','R','L': result := s5;
        'W'        : result := s2;
        else begin
          result := sf;
          fnr := 4;
          end
        end;
  s5: case c of
        '0'..'9'   : result := s6;
        else begin
          result := sf;
          fnr := 2;
          end
        end;
  s6: case c of
        '0'..'9'   : result := s6;
        ')'        : begin
                     if keller.erstes ='(' then begin
                         keller.pull;
                         if keller.erstes=keller.startzeichen
                            then result := s0
                         else result := s4;
                         end
                       else begin
                         result := sf;
                         fnr := 6
                         end
                       end;
        'V','R','L': result := s5;
        'W'        : result := s2;
        else begin
          result := sf;
          fnr := 5;
          end
        end;
  s7: case c of
        '0'..'9'   : result := s7;
        'V','R','L': result := s1;
        'W'        : result := s2;
        '$'        : result := se;
        else begin
          result := sf;
          fnr := 5;
          end
        end;
  end;
end;

begin
s := s0;
keller := tKeller.init('#');
anfang := true;
Kopiere(Editor,Backup);
Kopiere(Editor,helpmemo);
helpmemo.lines.add('$');
Editor.lines.Clear;
Befehle.lines.clear;
i := 0;
Zeile := helpmemo.lines.Strings[i];
```

```
Befehl := '';
while not (s in [se,sf]) do
  begin
  if zeile <>'$' then Editor.lines.Add(zeile);
  fZeile := '';
  for j := 1 to length(Zeile) do
     begin
     c := UpCase(Zeile[j]);
     fZeile := fZeile + ' ';
     if c in ['V','R','W','L','0'..'9','(',')','$'] then begin
       s := u(s,c);
       if s in [s1,s2,se] then
         if anfang then anfang := false
         else if (befehl <>'') and (keller.erstes=keller.startzeichen) then
                 begin Befehle.lines.add(Befehl); Befehl := '' end;
       Befehl := Befehl+c;
       end;
     if s in [se,sf] then break;
     end;
  i := i + 1;
  Zeile := helpmemo.lines.strings[i];
  end;
if keller.erstes<>keller.startzeichen then begin
  s := sf;
  fnr := 7
  end;
if s=se then begin
  Editor.lines.clear;
  if Befehl <>'' then Befehle.lines.add(Befehl);
  for i := 0 to Befehle.Lines.Count do
      Editor.lines.add(Befehle.lines.strings[i]);
  Editor.Lines.add('ok! fehlerfrei übersetzt!');
  DateiUebersetzt := true
  end
else begin
   DateiUebersetzt := false;
   delete(fzeile,length(fzeile),1);
   fzeile := fzeile + '^: ';
   case fnr of
     1: fzeile := fzeile + 'Befehl erwartet!';
     2: fzeile := fzeile + 'Zahl erwartet!';
     3: fzeile := fzeile + 'Zahl oder "(" erwartet!';
     4: fzeile := fzeile + 'Befehl oder ")" erwartet!';
     5: fzeile := fzeile + 'Befehl oder Zahl erwartet!';
     6: fzeile := fzeile + 'falsche Klammerstruktur!';
     7: fzeile := fzeile + 'es fehlen schliessende Klammern!';
     end;
   Editor.lines.Add(fzeile);
   end
end;

//---------------------------------------------------- Programm interpretieren ----------------------------------------------------
procedure TLEGO.Ausfhren2Click(Sender: TObject);
var i: integer;

procedure interpretiere(s: string); //rekursiver Interpreter
var Klammerzahl, i, zahl: integer;
    befehl,c: char;
    Klammerinhalt: string;
```

```
function HoleZeichen: char;
var c: char;
begin
c := s[1]; Delete(s,1,1);
result := c;
end;

function HoleZahl: integer;
var c: char; i: integer;
begin
i := 0;
c := HoleZeichen;
while c in ['0'..'9'] do begin
  i := 10*i+ord(C)-ord('0');
  c := HoleZeichen;
  end;
s := c + s;
result := i
end;

begin
s := s + '$';
befehl := HoleZeichen;
while befehl <> '$' do
  begin
  zahl := HoleZahl;
  case befehl of
    'V': if Ausgabeziel = Screen then Turtle.move(zahl)
         else LTurtle.move(zahl);
    'R': if Ausgabeziel = Screen then Turtle.turn(zahl)
         else LTurtle.turn(zahl);
    'L': if Ausgabeziel = Screen then Turtle.turn(-zahl)
         else LTurtle.turn(-zahl);
    'W': begin
         Delete(s,1,1);
         Klammerzahl := 1;
         Klammerinhalt := '';
         while Klammerzahl > 0 do begin
           c := HoleZeichen;
           case c of
             ')': Klammerzahl := Klammerzahl - 1;
             '(': Klammerzahl := Klammerzahl + 1;
             end;
           Klammerinhalt := Klammerinhalt + c;
            end;
         Delete(Klammerinhalt,length(Klammerinhalt),1);
         if Ausgabeziel = Screen then
            for i := 1 to zahl do interpretiere(Klammerinhalt)
         else begin
            spirit1.loop(2,zahl);
            Helpmemo.lines.Add('RCX: Loop(2,'+IntToStr(zahl)+')');
            interpretiere(Klammerinhalt);
            spirit1.endloop;
            Helpmemo.lines.Add('RCX: EndLoop');
            end
         end;
      end;
  befehl := HoleZeichen;
  end;
 end;
```

```
begin
if not DateiUebersetzt then
  begin
  showmessage('Übersetzen Sie erst Ihr Programm!');
  exit
  end;
if Ausgabeziel=Screen then begin
  Editor.Hide;
  Befehle.Hide;
  Image1.show;
  end
  else begin
    spirit1.PlayTone(1000,100); //Anfang der RCX-Task
    spirit1.DeleteTask(1);
    spirit1.BeginOfTask(1);
    Editor.Hide;
    Image1.Hide;
    Helpmemo.lines.clear;
    helpmemo.Show;
    Helpmemo.Lines.add('Datenübertragung zum RCX beginnt ...');
    Helpmemo.Lines.add('----------------');
    Helpmemo.Lines.add('RCX: PlayTone(1000,100)');
    Helpmemo.Lines.add('RCX: DeleteTask(1)');
    Helpmemo.Lines.add('RCX: BeginOfTask(1)');
    end;
for i := 0 to Befehle.lines.count do
    interpretiere(Befehle.lines.strings[i]);
if Ausgabeziel=RCX then begin
  spirit1.off('02');
  Helpmemo.Lines.add('RCX: Off(''02'')');
  LTurtle.M1an := false;
  LTurtle.M2an := false;
  spirit1.EndOfTask;
  spirit1.PlaySystemSound(2);
  spirit1.StartTask(1);
  Helpmemo.Lines.add('RCX: EndOfTask(1)');   //Ende der RCX-Task
  Helpmemo.Lines.add('RCX: PlaySystemSound(2)');
  Timer1.Interval := 2000;   //kurz warten vor der Ausführung
  end;
end;

//------------------------------------------------- Verbindung zum RCX prüfen ----------------------------------------
procedure TLEGO.Verbindungstest1Click(Sender: TObject);
begin
if LTurtle.PruefeVerbindung
  then begin spirit1.PlayTone(600,100); Showmessage('Verbindung ok!') end
  else showmessage('...keine Verbindung zum RCX!');
end;

//------------------------------------------------- Zeitkonstante für Translation ------------------------------------
procedure TLEGO.Bewegungskonstanteeingeben1Click(Sender: TObject);
var h: integer; s: string;
begin
h := LTurtle.dt;
if MessageDlg('Soll sich der RCX einen Schritt vorwärts bewegen?',
               mtInformation,[mbYes,mbNo,mbAbort],0)=mrYes
  then if LTurtle.PruefeVerbindung then begin
          spirit1.On_('02');
          spirit1.Wait(2,h);
          spirit1.Off('02')
          end
        else showmessage('Die Verbindung zum RCX ist unterbrochen!');
```

```
try s := inputbox('Zeitintervall:',
       'Geben Sie ein neues Zeitintervall für Bewegungen an!',IntToStr(h));
    LTurtle.dt := StrToInt(s);
  except LTurtle.dt := h;
    Showmessage('Eingabefehler: alter Wert bleibt erhalten.')
  end;
end;

//------------------------------------------------- Zeitkonstante für Rotation ----------------------------------------------
procedure TLEGO.Konstantenangeben1Click(Sender: TObject);
var h: integer; s: string;
begin
h := LTurtle.UZeit;
if MessageDlg('Soll sich der RCX drehen?',
               mtInformation,[mbYes,mbNo,mbAbort],0)=mrYes
  then if LTurtle.PruefeVerbindung then begin
          spirit1.On_('0');
          end
        else showmessage('Die Verbindung zum RCX ist unterbrochen!');
try s := inputbox('Drehzeit:','Geben Sie die Zeit für eine volle
                                  Umdrehung Ihres Modells an!',IntToStr(h));
    LTurtle.UZeit := StrToInt(s);
  except LTurtle.UZeit := h;
    Showmessage('Eingabefehler: alter Wert bleibt erhalten.')
  end;
end;

//------------------------------------------------- übersetzen Text anzeigen ------------------------------------------------
procedure TLEGO.Quelltext1Click(Sender: TObject);
begin
Befehle.show;
Editor.hide;
Image1.Hide;
end;

//------------------------------------------------- Quelltext anzeigen ------------------------------------------------------
procedure TLEGO.Quelltext2Click(Sender: TObject);
begin
Editor.show;
Befehle.Hide;
Image1.Hide;
end;

//------------------------------------------------- Bild anzeigen -----------------------------------------------------------
procedure TLEGO.Bild1Click(Sender: TObject);
begin
Editor.hide;
Befehle.Hide;
Image1.show;
end;

//------------------------------------------------- Bild löschen ------------------------------------------------------------
procedure TLEGO.Bildlschen1Click(Sender: TObject);
begin
Turtle.hide;
with Image1.Canvas do begin
  brush.style := bsSolid;
  brush.color := clWhite;
  rectangle(0,0,Image1.width,Image1.height);
  end;
  Turtle.show;
end;
```

```
//------------------------------------------------- Umschalten auf Screen ---------------------------------------------
procedure TLEGO.ZielBildschirm1Click(Sender: TObject);
begin
Ausgabeziel := Screen;
end;

//------------------------------------------------- Umschalten auf RCX ------------------------------------------------
procedure TLEGO.ZielRCX1Click(Sender: TObject);
begin
Ausgabeziel := RCX;
end;

//------------------------------------------------- Änderungen merken -------------------------------------------------
procedure TLEGO.EditorKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
DateiUebersetzt := false;
DateiGespeichert := false;
if Backup.Lines.count > 0 then begin
  Kopiere(Backup,Editor);
  Backup.lines.Clear;
  end;
end;

procedure TLEGO.EditorMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
if Backup.Lines.count > 0 then begin
  Kopiere(Backup,Editor);
  Backup.lines.Clear;
  end;
end;

//------------------------------------------------- nach einer Pause RCX-Programm starten -----------------------------
procedure TLEGO.Timer1Timer(Sender: TObject);
begin
spirit1.StartTask(1);
Helpmemo.Lines.add('RCX: StartTask(1)');
Timer1.interval := 0;
end;

end.
```

Lebenslauf

Name:	Modrow	
Vorname:	Eckart	
Amtsbezeichnung:	Studiendirektor	
Geburtsdatum:	18.2.1948	
Geburtsort:	Reinbek, Kreis Stormarn	
Familienstand:	verheiratet seit 13.12.1974, zwei Kinder	
Ehefrau:	Monika Modrow, geb. Dobbratz	
Religionszugehörigkeit:	ohne	
Bildungsgang:	Grundschule:	1954 – 1956, Brunsmark, Krs. Hzgt. Lauenburg 1956 – 1958, Wahlstedt, Kreis Segeberg
	Gymnasium:	1558 – 1966, Dahlmannschule Bad Segeberg
	Studium:	1969 – 1974, Physik, TU Braunschweig
Prüfungen:	Reifeprüfung:	19.10.1966, Dahlmannschule Bad Segeberg
	Diplomphysiker-Vorprüfung:	22.10.1971, TU Braunschweig
	Diplomphysiker-Hauptprüfung:	8.11.1974, TU Braunschweig
	2. Staatsexamen:	11.6.1976, Studienseminar Emden
Lehrbefähigung:	Mathematik, Physik ab 4.7.1979 besondere Lehrbefähigung Informatik	
Wehrdienst:	1.1.1967 – 31.12.1968, Lt. d. R.	

Dienstlicher Werdegang:

1.2.1975 – 31.7.1976	Referendar am Studienseminar Emden
1.8.1976 – 18.2.1984	Studienrat am Otto-Hahn-Gymn. Göttingen
19.2.1984 – 30.6.1992	Oberstudienrat am Otto-Hahn-Gymn. Gö.
1.7.1992 –	Studiendirektor am Max-Planck-Gymn. Gö.
ab 1.8.2000	Teilabordnung an die Universität Göttingen zur Konzeption und Durchführung der „Virtuellen Lehrerweiterbildung Informatik in Niedersachsen“ VLIN
ab 15.10.2002	Lehrauftrag zur Informatikdidaktik an der Universität Göttingen

27.10.2002, Scheden