informatica didactica 6 (2006) - http://www.informatica-didactica.de

Zur Ordnungswirkung fundamentaler Ideen der Informatik am Beispiel der theoretischen Schulinformatik

Eckart Modrow

Max-Planck-Gymnasium
Theaterplatz 10
37073 Göttingen
emodrow@astro.physik.uni-goettingen.de

Zusammenfassung

Wir untersuchen Schwills „fundamentale Ideen der Informatik“ [Schw93a] auf ihre Brauchbarkeit als Ordnungsschema für Unterrichtsinhalte und modifizieren und erweitern sie im Ergebnis. Am Beispiel der eher „theoretischen“ Teile des Informatikunterrichts werden die Konsequenzen aufgezeigt. Es folgen drei Beispiele zu konkretem ideenorientierten Unterricht für verschiedene Altersstufen.

Abstract

We analyze the concept of “fundamental ideas of computer science” proposed by Schwill [Schw93a] whether it is suitable to organize subjects in schools. After modifying and extending the concept we apply it to the more “theoretical” subjects of computer science in schools. We conclude with three examples of idea-oriented lessons for different levels.

Einleitung

In einem LOG IN-Artikel schreibt Schelhowe 1997 [Sche97]: *„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.“* Noch kürzer findet man z. B. bei Herschel [Her74]: *„Nichts ist praktischer als eine gute Theorie.“* Ist es aber wirklich dieser Eindruck, den Schülerinnen und Schüler mitnehmen, nachdem sie sich mit theoretischen Fragestellungen beschäftigt haben? Bevor man – wie oft gefordert – die Theorieanteile im Informatikunterricht erhöht, 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 sehr genau klarmachen, was es bewirken will und auch 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. Statt also durch die Ausbildung theoretischer Modelle Klarheit in ein unübersichtliches Konglomerat kaum zusammenhängender Fakten zu bringen, werden die aus einer Theorie folgenden Methoden praktisch geübt, wobei die Praxisrelevanz solcher Fertigkeiten immer mehr abnimmt. In der Informatik, mit Maschinen als Thema, die entsprechende Algorithmen ausführen können, wäre ein vergleichbares Vorgehen obskur. Im Sinne von Schelhowe müssen wir also 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 danach in diesem Gebiet „besser zurechtfinden“. Im folgenden wird versucht, zu diesem Zweck die Fundamentalen Ideen der Informatik im theoretischen Bereich zu identifizieren und daraufhin zu untersuchen, ob und in welcher Form sie als Ordnungsschema taugen.

1. Zu den Kriterien für fundamentale Ideen

Fundamentale Ideen strukturieren ein Fach sowohl wissenschaftlich wie erkenntnistheoretisch. 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“* [Hey95] und tragen so zur kulturellen Kohärenz bei. Sie *„schaffen Beziehungsnetze, prägen* [fachlichen] *Details eine verbinden-*

de Struktur auf und unterstützen so den nichtspezifischen Transfer" [Schw96]. Fundamentale Ideen sind also außerordentlich hilfreich – aber was sind „fundamentale Ideen"? Betrachten wir zuerst ihre Wirkungen.

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 und ermöglichen durch diese Reduktion Lernen. Die Ideen eines Faches verdeutlichen damit die mit diesem Fach verbundenen Sichten auf die Welt, 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 [Kla96] 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 demokratischen 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. 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. Das 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. 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.

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 Schwill [Schw93a] ist eine fundamentale Idee der Informatik ein Denk-, Handlungs-, Beschreibungs- oder Erklärungsschema, das vier Kriterien erfüllt: 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 *Sinnkriterium* (Die Idee besitzt eine Verankerung im Alltagsdenken und eine lebensweltliche Bedeutung.) und das *Zeitkriterium* (Die Idee ist in der historischen Entwicklung der Informatik deutlich wahrnehmbar und bleibt längerfristig relevant.). Die Bedeutung dieser Kriterien wird besonders deutlich, wenn man sie in Hinsicht auf konkreten Unterricht diskutiert.

Das Horizontalkriterium sortiert reines Spezialwissen aus, 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 dann deutlich werden, wenn wirklich unterschiedliche Anwendungsbereiche vorkommen, in denen die Idee relevant ist. 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.

Das Vertikalkriterium ist noch mehr als das vorherige ein Kriterium für Lehrerinnen und Lehrer, denn die Heranwachsenden überblicken nicht ihren Bildungsgang als Ganzes, 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. 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.

Nach Schwill können durch Betrachtung der geschichtlichen Entwicklung fundamentale Ideen identifiziert werden. Das Zeitkriterium sichert die Kontinuität des Unterrichts und den Wert der Kenntnisse und Erfahrungen der Unterrichtenden und verhindert fachliche „Moden“. Offensichtlich wird es im Bereich der Schulinformatik kaum beachtet.

Schwill interpretiert das Sinnkriterium so, dass der Kontext fundamentaler 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. 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. Wir sollten deshalb 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 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 Alternativen, zwischen denen Fachdidaktiker eine Entscheidung treffen müssen. Schwill verzichtet auf diese Entscheidung, und deshalb enthält sein Katalog (fast) alle in Frage kommenden Ideen, 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. Zu den fundamentalen Ideen der Informatik

Schwill nennt als Masterideen die *Algorithmisierung*, die *strukturierte Zerlegung* und die *Sprache*. Untersuchen wir die dazugehörigen drei Ideenbäume (s. Anhang), dann sehen wir, dass Algorithmisierung und strukturierte Zerlegung direkt zu sehr handfesten Unterideen führen. Innerhalb der Bäume können wir die einzelnen Ebenen als Präzisierungen der vorangegangenen auffassen, als zunehmende fachliche Durchdringung der übergeordneten Ideen. Die genannten Masterideen selbst können in ihrer Allgemeinheit m. E. 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, und dass sich das Fachliche verliert, wenn die Fundamentalität zunimmt. 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.

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 deshalb zu fragen, welche der Äste und welche Verzweigungstiefe für eine hinreichend genaue Rekonstruktion der angestrebten informatischen 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 und den Wegen zu diesen. Die Lernenden sollten z. B. wissen, dass durch Reihung, Alternative und Iteration Algorithmen so formuliert werden, dass Computer sie abarbeiten können. Zur Formulierung können aber auch Reihung, Alternative und Rekursion herangezogen werden. Es ist sicher nicht unbedingt notwendig, beide Möglichkeiten aufzuführen, und keine der beiden ist qualitativ besser; 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.

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 sie 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* 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. 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 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 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. Durch die ihr zugeordneten Bereiche formale Sprachen, Automaten und Schaltwerke berührt sie gut erprobte schulische Standardthemen. Netzwerke unterschiedlicher Klassen, gekoppelte Automaten und deren Anwendungen liefern Zugang zu aktuellen und auch für die Unterrichteten spannenden Themen. 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 Richtlinien – überlassen bleiben.

3. Fundamentale Ideen in 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 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 (...)"* [Her94, S. 38]. Bussmann und Heymann [Bus87, S. 29] 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."* Schubert [Schu99, S. 8] schreibt: *„Worin besteht die Theorie der Informatik? Einen Zugang bildet die Verbindung von Algorithmen, Sprachen und Maschinenmodellen."* Den gemeinsamen Aspekt aller dieser Äußerungen bilden die *informatikspezifischen Maschinenmodelle* (Automaten), die im Gegensatz zu den statisch beschriebenen Zusammenhängen in mathematischen Formulierungen das dynamische Verhalten eines Systems als Prozess 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

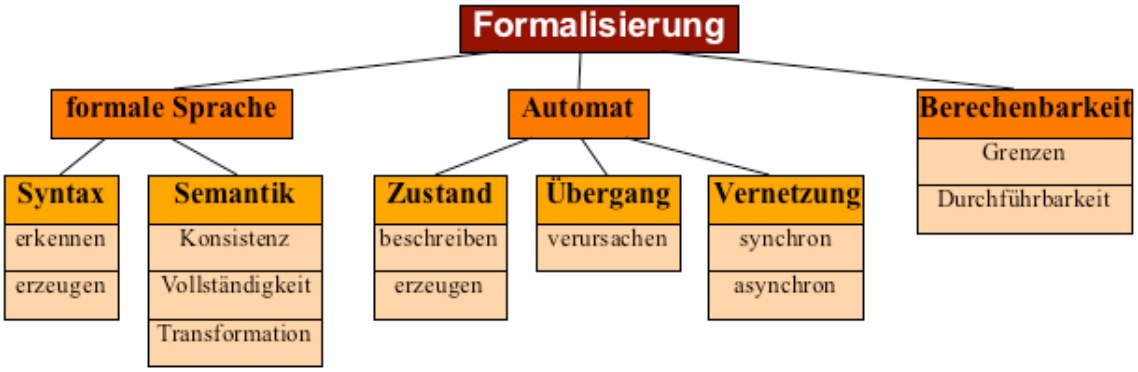
Schule geeignet. Es findet in der Fachwissenschaft auf sehr unterschiedlichen Gebieten und sehr unterschiedlichen Niveaus Anwendung (Horizontal- und Vertikalkriterium), wird seit den Anfängen der Informatik benutzt und besitzt gerade durch seine Anschaulichkeit eine lebensweltliche Bedeutung (Zeit- und 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. Damit entfällt für diese Automatenklassen, mit denen in der Schule überwiegend gearbeitet wird, weitgehend auch der Einwand von Rechenberg [Rec97, S. 32] und Schelhowe [Sche97, S. 29], 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 sequenziell 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*.

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 noch 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, 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.



4. 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 müssen einerseits 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 gehen unsere Maschinen aus einem Zustand in unterschiedliche Folgezustände über. 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 und zwischen Maschinen, vor allem die Anwendung auf so einfache Konstrukte, wie wir sie meist behandeln. Da wir es noch nicht mit Programmier-

sprachen 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 dieser 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, aber auch Technikmodelle, deren Motoren durch Relais geschaltet werden, Turtlegrafik-Umgebungen, Sprachspiele („Zufallsgedichte“, „Eliza“, …) 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. 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 so viele Erfahrungen mit den Einzelbausteinen dieses Weges (Codierungen, Turingmaschinen, …) zu machen, dass eine geschlossene Darstellung mit ihren verblüffend weit tragenden Aussagen gut machbar wird. Die numerischen und durch Iterationen und Kombinationen auftretenden Grenzen sind allerdings nahe liegend und durch Experimente direkt erfahrbar. Sie bieten ebenso wie die zellulären Automaten hochinteressante Probleme schon für den Anfangsunterricht.

5. Konsequenzen für die theoretische Schulinformatik

Auf die ordnende Wirkung fundamentaler Ideen wurde schon eingegangen. Wenn aber Erfahrungen geordnet werden sollen, dann müssen sie erstmal vorhanden sein. Es muss also genügend „Stoff“ bereit liegen, anhand dessen theoretisches Arbeiten verdeutlicht werden kann, und er muss genügend umfangreich sein, um die ordnende Wirkung zu erfahren. Weil Theorien mit Begriffen arbeiten, die aus konkreten Erfahrungen abstrahiert werden, ist die *fachspezifische Begriffsbildung* eine notwendige Vorstufe der eigentlichen theoretischen Arbeit. 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. 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 erklären.

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, anderer-

seits eine Technikwissenschaft zu sein. Betonen wir beide Aspekte, dann können wir auch auf dem Gebiet der Theorie konstruktiv arbeiten, und umgekehrt sollte *„der Geist der Theorie von der ersten Stunde an präsent (sein)“* [Nie02]. 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 Ideenbäumen 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 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 in Programme oder Schaltungen übersetzen, so dass die exakte Beschreibung eines geeigneten Systems schon 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. Der Kurs arbeitet konstruktiv, die Arbeitsmittel werden der Theorie entliehen. Dementsprechend sollten keine „nicht direkt hilfreichen“ Beweise, etwa zur Äquivalenz von Automatenklassen, geliefert werden. 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.).

Literatur

[Bus87] Bussmann, H., Heymann, H.-W.: Computer und Allgemeinbildung. Neue Sammlung 27 (1987) 2-39

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

[Her94] Herget, W.: Ziele und Inhalte des Informatikunterrichts – zum Vergleich. In: H. Hischer (Hrsg.): Mathematikunterricht und Computer – Neue Ziele oder neue Wege zu alten Zielen? Franzbecker (1994) 28-40

[Hey95] Heymann, H.-W.: Zielsetzungen eines künftigen Mathematik- und Informatikunterrichts – Überlegungen aus bildungstheoretischer Sicht. In: H. Hischer, M. Weiß (Hrsg.): Fundamentale Ideen - Zur Zielorientierung eines künftigen Mathematikunterrichts unter Berücksichti-

gung der Informatik. Franzbecker (1995) 46-57

[Kla96] Klafki, W.: Neue Studien zur Bildungstheorie und Didaktik. Zeitgemäße Allgemeinbildung und kritisch-konstruktive Didaktik. 4. Auflage. Weinheim: Beltz 1996.

[Mod04] Modrow, E.: Technische Informatik mit Delphi. http://www.emu-online.de, 2004 Zugriff: 23.12.2005

[Mod05] Modrow, E.: Theoretische Informatik mit Delphi. http://www.emu-online.de, 2005 Zugriff: 23.12.2005

[Nie02] Nievergelt, J.: Folien zur Vorlesung, http://www.tedu.ethz.ch/didaktik/id1_material.html, 2002

[Rec97] Rechenberg, P.: Quo vadis Informatik? LOG IN 17,1 (1997) 25-32

[Sche97] Schelhowe, H.: Auf dem Weg zu einer Theorie der Interaktion? LOG IN 17, 5 (1997) 27-33

[Schu99] Schubert, S.: Begleitmaterial zur Vorlesung Einführung in die Didaktik der Informatik. WS 1999/2000, Skript 1999

[Schw93] Schwill, A.: Fundamentale Ideen in Mathematik und Informatik. Skript 1993

[Schw93a] Schwill, A.: Fundamentale Ideen der Informatik. Zentralblatt für Didaktik der Mathematik 1 (1993) 20-31

[Schw96] Schwill, A.: Vorlesungen zur Didaktik der Informatik. Skript 1996/97

Anhang

Die fundamentalen Ideen der Informatik werden in der folgenden Version benutzt: