

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 einzusetzen 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 in einem sehr trägen System wie dem öffentlichen Schulwesen langfristig existieren kann. Diese Frage stellt sich verschärft in der aktuellen Situation, wird aber auch bei verbesserter Lehrerausbildung virulent bleiben. Schulfächer brauchen Standards, die üblicherweise durch die Universität gesetzt werden, 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 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.

Anregungen zu diesem Problemkreis können in Klafkis Ausführungen zur Handlungsforschung und zur schulnahen Curriculumentwicklung 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“ 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 Unterrichts-

tenden 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. 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.“

Im Schulinformatikbereich übersteigt die für die Verbreitung mancher neuer Inhalte erforderliche Zeit oft die Dauer, in der diese Inhalte für den Unterricht relevant sind. 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 Umfang zu betreiben haben, sondern dass eine Art Handlungsforschung 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.“* Speziell für den Informatikunterricht hält er die Anforderungen an Lehrer „derzeit“ nicht für erfüllbar. 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. Entsprechend selten werden sie dann angemessen unterrichtet. 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 Simu-

lation 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. 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.

Breibt man schulnahe Curriculumentwicklung im genannten Sinne, dann reduziert sich die *Handlungsforschung* weitgehend auf den Erwerb von Erfahrungen mit neuen Werkzeugen, das Erlernen neuer technischer Details und die damit meist verbundene Materialentwicklung. Wirklich neue Konzepte tauchen eher selten auf – wie in anderen Fächern auch. Solche Änderungen im Detail und besonders die Materialentwicklung kann effizient durch ortsnahe Lehrerfortbildung 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 – und zwar in Kenntnis des offenen Lernzielkatalogs. In diesem Rahmen können die Universitäten dann auch „richtige“ Handlungsforschung betreiben, „*Forschung beim Unterrichten unter Einmischung*“. Gefordert ist deshalb immer noch eine „*schulnahe Curriculumentwicklung, gekoppelt mit neuen Formen der praktischen Lehrerfortbildung, und zwar mit einem unterrichtsnahen in-service-training*“. 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. 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“ und tragen so zur kulturellen Kohärenz bei. Sie „schaffen Beziehungsnetze, prägen [fachlichen] Details eine verbindende Struktur auf und unterstützen so den nichtspezifischen Transfer“. 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. 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 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 Faches 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. 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. 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 Faches innerhalb des Kanons der Wissenschaften zu diskutieren – nachdem genügend Kenntnisse erworben worden sind, um diese Diskussion seitens der Schülerinnen und Schüler sinnvoll zu führen.

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.“* 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 *„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). 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 werdenden“* universellen Ideen allerdings *„selbst die Inhalte eines ernst gemeinten Informatikunterrichts sein [müssten]“* – und damit *„sei die allgemein bildende Schule erheblich überfordert“*. Er macht es sich damit m. E. ziemlich einfach, weil einerseits das „Wirksamwerden“ der universellen Ideen wohl nur schwer nach-

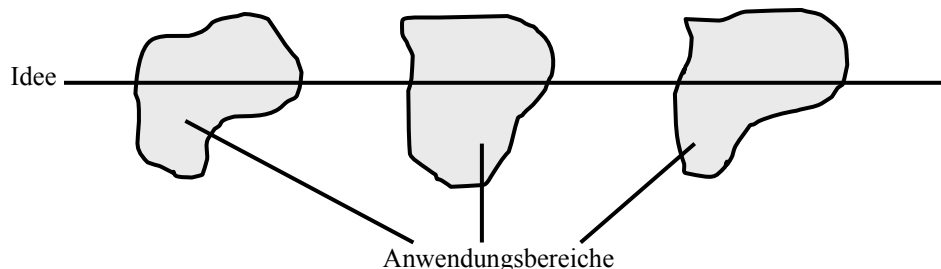
zuprüfen ist, und er andererseits pauschal den doch sehr konstruktiven Schwillischen Ideen die Realisierbarkeit abspricht – und sich somit einer inhaltlichen Diskussion entzieht. Insbesondere übersieht er, dass – selbst wenn die Mathematik wirklich die Informatik umfassen würde – dieses für die Unterrichteten unerheblich wäre, solange die prinzipiell vorhandenen Möglichkeiten nicht wirklich realisiert werden, also die spezifischen Stärken des Schulfaches Informatik auch wirklich im Mathematikunterricht zu erfahren sind.

Sigrid Schubert und Wilfried Herget sehen Schwill's Kriterien für fundamentale Ideen wesentlich positiver, halten die Diskussion darum aber offensichtlich noch nicht für abgeschlossen, „da er einen sehr eingeschränkten Bereich der Informatik in die Auswahl einbezieht. Aspekte der technischen und angewandten Informatik fehlen darin.“ 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.“

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.

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:

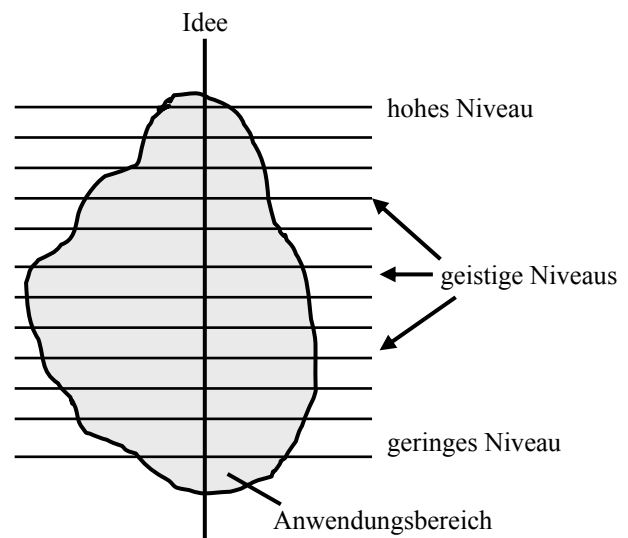


Das Kriterium sortiert reines Spezialwissen aus, ist offensichtlich sinnvoll. Es ist aber eigentlich ein Kriterium für „erfahrene Informatiker“, da es breites Fachwissen voraussetzt, aus dem die gemeinsame Idee durch Abstraktion gewonnen werden kann. Im Unterricht wird den Schülerinnen und Schülern diese Eigenschaft einer Idee nur dann deutlich werden, wenn wirklich unterschiedliche Anwendungsbereiche vorkommen, in denen die Idee relevant ist – also mindestens zwei. 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, 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.

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. 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.

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. 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 vorthoretisch 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.

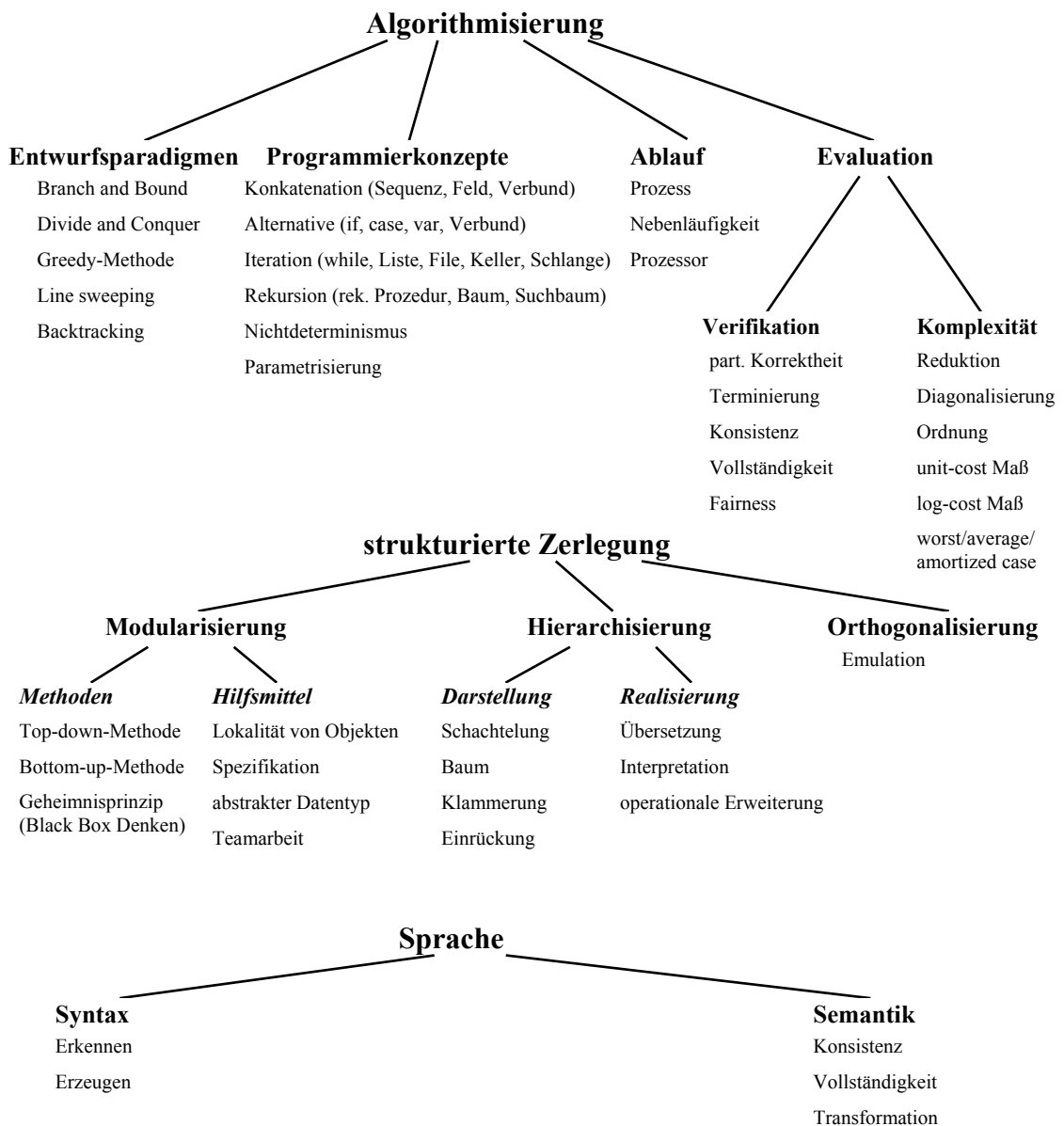
Der Unterricht muss dann so angelegt werden, dass sich diese – wenigen – fundamentalen Ideen bei den Schülerinnen und Schülern bilden können, er muss Kenntnisse und Erfahrungen vermitteln, die anhand dieser Ideen zu ordnen sind, und er muss diese Ideen zu einem geeigneten Zeitpunkt explizit thematisieren, 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 Schwillischen 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 Schwill's Sinnkriterium wird von ihm – übrigens sehr knapp – nur eingesetzt, um die Eigenschaft einer Idee, vorthoretisch 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, 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*. 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.



Schwill bezeichnet alle in diesen drei Bäumen auftauchenden Ideen ausdrücklich als fundamental, 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.

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, weil einerseits strukturierte Zerlegung sicher als Teil des Algorithmisierungsprozesses aufgefasst werden kann, und andererseits z. B. die algorithmischen Grundstrukturen Anfang oder Ende 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* nicht spezifisch für die Auswahl mathematischer Ideen sind, sondern in Hinsicht auf die Mathematik noch einer erheblichen Präzisierung bedürfen, können *Algorithmisierung* und *strukturierte Zerlegung* 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. Fundamentalideen 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.

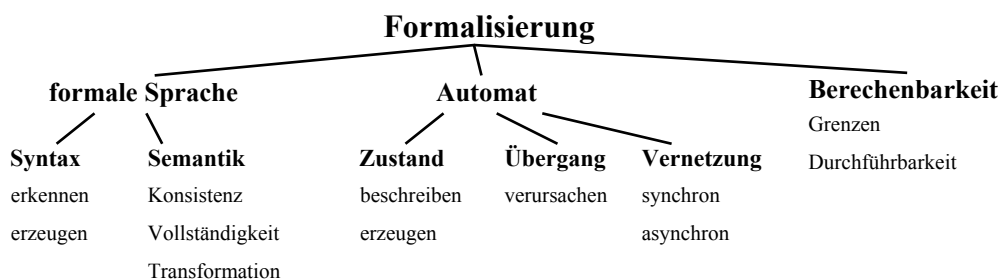
Betrachten wir einmal zum Vergleich die *Algorithmisierung* und die *strukturierte Zerlegung*, angewandt auf die Physik: Ebenso wie in der Informatik können wir die betrachteten Systeme modellieren, die erkannten Gesetzmäßigkeiten formal beschreiben und so z. B. Voraussagen über die zeitliche Entwicklung des Systems machen – natürlich jeweils mit den fachspezifischen Methoden und Werkzeugen. Wir können die Systeme zerlegen, unter verschiedenen Aspekten vereinfachen, 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. 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. 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, 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 jeweiligen Gültigkeitsbereich, 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* 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 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 mindestens 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 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** 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 durch-

föhrbaren 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 Persönlichkeit 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. Im Bereich der Schule föhrt 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 des-

halb – 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, der herausstellt, dass sich bei allen vordergründigen Unterschieden die Ziele relativ wenig von einander unterscheiden, 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, 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 mathema-

tischen 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 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 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 „algorithmienorientierte 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, 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. 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 (im Sinne der ITG) oder nur ein Spezialistenfach für wenige ist. Die damit verbundenen Entscheidungen zwischen

Pflichtfach \leftrightarrow Wahlfach
Oberflächenwissen \leftrightarrow vertieftem Wissen
Anwenderkenntnissen \leftrightarrow 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*“, „*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*“), 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.

- Einerseits wird aktive Schülerarbeit und Projektunterricht als grundlegend für das Fach herausgestellt,

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 andererseits verschiedene Programmierparadigmen kennen gelernt und beurteilt werden sollen.

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

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*“ (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) herabgestuft. Eine wenig überraschende Konsequenz ist, dass „*die am Versuch beteiligten Schüler (...) beängstigend wenig Interesse erkennen* [ließen]“ – und dann braucht man eben ein Pflichtfach.

Ich halte eine fachliche „Orientierung“ des Informatikunterrichts – gleich welcher Art – für kontraproduktiv. 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. Ein Fach sollte den durch allgemeindidaktische Überlegungen gesteckten Rahmen vollständig ausfüllen und nicht versuchen, ihn aus fachlichen Überlegungen weiter einzuschränken. 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* prägen und schon gar nicht als *Startpunkt* gewählt werden. Weiterhin bezweifle 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 ist der ungute Einfluss der (hier: Informations-)Orientierung nachzuvollziehen: Nach erfreulichen Aussagen zu Motivation und Kreativitätsförderung und einer einleuchtenden Begründung für die Bedeutung des Informationsbegriffs im Informatikunterricht, der Entwicklung eines Grundschemas der Informationsverarbeitung 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“ 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.*“ Er verzichtet weitgehend auf die Implementierung der entwickelten Methoden und so m. E. auf den aus allgemeindidaktischer Sicht entscheidenden Grund für ein Schulfach Informatik (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. 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 weiterführende Einheiten auftauchen. Ich halte es für besser, wenn Unterrichtsinhalte den Unterrichteten *zum aktuellen Zeitpunkt* hilfreich und sinnvoll erscheinen 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, und er läuft m. E. Gefahr, die Schülerinnen und Schüler zu verlieren, denn sonst wäre seine Forderung nach Pflichtunterricht, weil „*die Schüler nur in einem solchen Fach zu derartigen Anstrengungen bereit sind*“, kaum zu erklären. 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.

Ähnliches lässt sich zur Didaktik Rüdiger Baumanns 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 – in der Schule wenig Bedeutung erlangt hat. Baumann beginnt mit den Bezugswissenschaften und einer Handlungstheorie, die er formal zu formulieren versucht 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, angelehnt an Kategorien der allgemeinen Didaktik, ein ganzes Arbeitsprogramm enthält, das aber nirgends angegangen wird. 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.“* 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 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. 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.

Einen ganz anderen Weg als die beiden genannten Didaktiker zeigen die Ergebnisse der „fachdidaktischen Gespräche der TU Dresden“, die aktuell zusammen mit Hubwiesers informationszentrierter Didaktik offensichtlich in die „Empfehlungen für ein Gesamtkonzept zur informatischen Bildung an allgemein bildenden Schulen“ 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 Informatiksystemen 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 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.

2.4 Zum Programmieren und der OOP

„Informatikunterricht soll kein Programmierkurs sein. Warum eigentlich nicht?“ Dieses Zitat von Sigrig 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. 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.

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.

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.“* 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 seiner Zergliederung Lösungsentwürfe und deren Realisation mit dem Computer folgen.“* 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, 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“. 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.“ 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, 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. Dieses Erkenntnis und die Fähigkeit zum begründeten Vergleich besitzen einen höheren Bildungswert als Vertiefungen in einem einzelnen Paradigma.*“ 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.

Die Wahl der Sprache ist sicherlich wichtig, 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, 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. 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. 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, 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 Verstecken ggf. die Entwicklung eines gültigen Computermodells bei den Lernenden. Ben-Ari spricht von einem „objektorientierten Paradoxon“. Er hat natürlich Recht damit, dass die Lernenden in jedem Fall ein Computermodell entwickeln werden, und auch die Notwendigkeit eines solchen Modells ist offensichtlich: Trotzdem ist das m. E. kein Einwand gegen OOP-Sprachen. Ben-Ari sagt nämlich nichts über die Art des zugrunde zu legenden Computermodells. Folgen wir seiner Argumentation, dass es falsch ist, unbekannte Eigenschaften zu verstecken, 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 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.

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 einfügen 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. 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. Auch die Effizienz der entwickelten Unterrichtsmaterialien wird erhöht, wenn deren Bewertung erleichtert wird und damit die Nutzungsmöglichkeiten steigen. 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 *tendenzielle* Beurteilung sehr wohl möglich, denn sonst wäre z. B. eine Evaluation von Unterricht und seinen Inhalten aussichtslos.

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.
- 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. Die Zielrichtung ist aber eine andere als in eher fachlich orientierten Unterrichtsphasen.

Bevor auf Details der fachlichen Planung eingegangen werden kann, muss m. E. unbedingt geklärt sein, welche **Unterrichtsmethoden** in welchem Umfang eingesetzt werden. 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, ... ernst, dann *sind* das schon weitgehend die Unterrichtsinhalte. Die hierfür bereitgestellte Zeit steht für systematischen Fachunterricht nicht mehr zur Verfügung, 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 gewannen die Vorschläge erheblich an Relevanz gegenüber Alternativen, die die Rahmenbedingungen nicht berücksichtigen. **Wir hätten den gesuchten Praxisbezug gewonnen.**

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. Eine Zusammenstellung solcher Unterrichtseinheiten spiegelt dann die fachliche Sicht des Unterrichtenden auf sein Fach, so wie sie den Unterrichteten erscheint. 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.

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 Unterrichtenden 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 angegebenen 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 %)