Vom Modellieren zum Gestalten -

Objektorientierung als Impuls für einen neuen Informatikunterricht?

Carsten Schulte

Didaktik der Informatik
Universität Paderborn
Fürstenallee 11
D-33102 Paderborn
E-Mail: carsten@uni-paderborn.de

Zusammenfassung:

Modellieren ist kein neues Thema für die Fachdidaktik Informatik. Grundlegende Ziele, Konzepte und Methoden wurden im Zuge des anwendungsorientierten Informatikunterrichts festgeschrieben. Hierzu zählen die Projektmethode, der Versuch der integrativen Behandlung der gesellschaftlichen Auswirkungen und die Orientierung des Unterrichts an den Phasen des Software life cycles (siehe die Beiträge in (Arlt, 1981); die Zusammenfassungen in (Eberle, 1996) und (Forneck, 1992)). Modellieren ist dennoch ein aktuelles Thema. Das hängt zu einem nicht geringen Teil mit der Objektorientierung zusammen, die verbunden ist mit neuen (oft grafischen) Notationen, sowie neuen Verfahren und Sichtweisen auf die Modellierungsphase bei der Softwareentwicklung. All dies berührt natürlich den Informatikunterricht. In diesem Artikel gehe ich der Frage nach, welche Chancen für die Unterrichtskonzeption der Paradigmenwechsel in der Softwaretechnik (Quibeldey-Cirkel, 1994, S. 2-10), die Hinwendung zur Objektorientierung, mit sich bringt. Insbesondere werden zwei Aspekte, das Trainieren von Denkfähigkeiten sowie die Thematisierung der gesellschaftlichen Auswirkungen von Informatik betrachtet.

Abstract:

Modelling is not a new topic within didactics of computer science. Fundamental learning objectives, concepts and teaching methods were developed in the so called 'application oriented computer science lessons' in the early 80's. Main aspects of this concept are the project method, the sequence of lessons along the steps of the software life cycle and the attempt of integrating subjects of computers and society (see contributions in (Arlt, 1981); summaries in (Eberle, 1996) and (Forneck, 1992)). Nevertheless modelling is a current topic. It is often connected with object orientation, with new (graphical) notations, and with new methods and views on the modelling phase in the software engineering process. This development obviously affects computer science education in schools. The paper discusses opportunities for instruction arising from the paradigm shift (Quibeldey-Cirkel, 1994, p. 2-10) in software engineering. The main focus lies on training thinking skills and studying social effects of computer science.

1. Modellieren im anwendungsorientierten Informatikunterricht

Um die Auswirkungen abschätzen zu können, die eine Übertragung der Neuerungen in der Fachwissenschaft auf den Informatikunterricht nach sich ziehen könnte, soll zunächst skizziert werden, auf welcher Basis der heutige Informatikunterricht entstanden ist.

Das Anfertigen einer derartigen Skizze erfordert methodisch die Analyse der vorhandenen Quellen zur Informatikdidaktik, zu Unterrichtsinhalten, -methoden und vor allem der damit verbundenen Zielen, kurz: eine umfangreiche empirische Untersuchung. Ein sauberes methodisches Vorgehen würde also nicht nur den hier zur Verfügung stehenden Raum übersteigen. Daher werde ich mich auf eine recht grobe Skizze beschränken, die sich aus der Analyse der veröffentlichten Literatur zum Themenkomplex des Modellierens ableiten lässt. Um das Nachvollziehen zu erleichtern, können in der Online-Version des Artikels jeweils dort, wo auf Literatur eingegangen wird, kurze Zitate aus der angesprochenen Literatur aufgerufen werden - die wichtigen Zitate befinden sich direkt im Text.
Wie sieht also die Grundlage des typischen Informatikunterrichts aus, bezogen auf den Schwerpunkt des Artikels, das Modellieren? Um das festzustellen, müssen wir uns die Entwicklung des in der Didaktik so genannten anwendungsorientierten Ansatzes ansehen:
Im anwendungsorientierten Ansatz wollte man das "algorithmische Denken" sowie die "Auswirkungen der Datenverarbeitung auf die Gesellschaft" integrativ behandeln, um ein Auseinanderfallen des Informatikunterrichts "in einen Programmierkurs und einen Gesellschaftskundeunterricht" zu verhindern (Riedel, 1981, S.38). Beide Bereiche werden als konstitutiv angesehen, der Informatikunterricht dient als Denkschulung und als Vorbereitung auf eine informatisierte Welt.
Die Denkschulung erfolgt durch Problemlösen, das sich an den Methoden des Software-Engineering orientiert. Ein Beispiel dafür ist die Formulierung "Vom Problem zum Programm", so der Untertitel eines Schulbuchs (Balzert, 1987). Darin wird formuliert, "[i]m täglichen Leben steht man ständig vor Problemen, die man lösen muß" (S.13). Und etwas später (S.16f):

> "Die Beispiele und Erläuterungen zeigen, daß das Lösen von Problemen ein umfangreiches Gebiet umfaßt. Ziel der Informatik ist es, Lösungsverfahren für bestimmte Probleme zu finden und so zu beschreiben, daß sie von DVAn abgearbeitet werden können. Dadurch ergeben sich Einschränkungen hinsichtlich des Problemkreises als auch hinsichtlich der Lösungsbeschreibungen.[1]"

Zunächst orientiert sich der Problembegriff am täglichen Leben, um dann eingegrenzt zu werden auf algorithmisch lösbare Probleme. Der Einsatz der DVAn ist orientiert am zeitgemäßen (Erstauflage 1976, zweite Auflage 1983, zitiert aus dem unveränderten Druck 1987) Schema 'Eingabe - Verarbeitung - Ausgabe'. Mit der Konzentration des Computereinsatzes auf die Perspektive der automatischen Verarbeitung wird die Lösungsbeschreibung auf einen Algorithmus reduziert. Daneben sind Aspekte wie Benutzungsschnittstellen, Abfangen von Fehlern (etwa durch falsche Eingaben), Dokumentation und Wartung zunächst zweitrangig.

Die Schüler sollen Verfahren zur Problemlösung lernen:

> "Als ein Leitfaden auf dem Weg vom Problem zur Lösung wird ein Schema zum Problemlösen benutzt, das einen gewissen Rahmen bzw. eine Hilfestellung beim Problemlösen bietet und für viele Problembereiche ein sinnvolles systematisches Vorgehen ermöglicht" (S.17).

[1] Der Artikel ist nach den Regeln der neuen Rechtschreibung verfasst, die Zitate sind jedoch originalgetreu übernommen, also oft in der alten Rechtschreibung gesetzt.

In der Konsequenz werden die Begriffspaare problemlösendes Denken und algorithmisches Denken sowie Softwareentwicklung und Problemlösen zu Synonymen. Der Weg ist geebnet, Softwareentwicklung zum Zwecke der Denkschulung im Unterricht zu thematisieren. Das Problemlöse-Schema deckt sich mit einem Vorgehensmodell für die Softwareentwicklung. Der Unterricht wird als Projekt organisiert, das in Phasen abläuft, die sich am Schema des 'Software-Engineering' orientieren.

Nun, wo bleibt das Modellieren? Modellieren wird zu einem Begriff, der die planerischen Anteile (und Phasen) der Softwareentwicklung betont und als Gegensatz zum einfachen Codieren und zum so genannten 'Hacken' in Stellung gebracht wird. Dazu beachte man auch, wie im folgenden Zitat nebenbei auch der enge Zusammenhang zwischen Modellieren und Problemlösen angedeutet wird:

> "Informatikunterricht soll kein Programmierkurs sein. Warum eigentlich nicht? [...] Problemlösen (Modellieren und Strukturieren) unter Anwendung von Informatikprinzipien und -methoden gilt als erstrebenswert. Die Programmiersprache soll im Hintergrund (Mittel zum Zweck) bleiben. Das aber ist Programmierung (nicht zu verwechseln mit Codierung)" (Schubert, 1991, S.27).

Ebenfalls diese Passage zitierend streicht Hubwieser heraus, dass die Anforderung darin liege, die planerischen Aspekte der Softwareentwicklung zu betonen, ohne dass die "Modellierung und Strukturierung" einerseits zu "philosophischen Exkursen" verkommt bzw. andererseits "spezifische Eigenheiten der verwendeten Programmiersprache in den Mittelpunkt des Unterrichts rücken" (Hubwieser, 1999, S.24f).

In einer weiteren Variante unterscheidet Eberle zwischen "relativ programmunabhängige[r] Umsetzung mittels der Strukturierungshilfsmittel wie Programmablaufplan, Struktogramm, Pseudocode oder auf einer höheren Ebene Datenflussplan" und der "Implementierung in einen Programmcode". Das "eigentliche" Problemlösen sei demnach die "Umsetzung der alltagssprachlich formulierten Problemstellungen in formale Beziehungen". Dabei falle "dem antizipativen Verständnis der zeitlichen Abläufe (Prozeduren)" besondere Bedeutung zu. Und daran zeige sich ebenfalls der Unterschied zur Mathematik (Eberle, 1996, S.329). Auch hier zeigt sich also die weitgehende Gleichsetzung von Problemlösen, Modellieren und Softwareentwicklung.

Halten wir den ersten Pinselstrich unserer Skizze fest: Anfang der Achtziger Jahre fällt eine Art didaktische Grundsatzentscheidung, in deren Folge das Unterrichtsziel (Vermittlung von Problemlösefähigkeiten), der Unterrichtsinhalt (Modellieren) und die Anbindung an die Wissenschaft Informatik (Softwareentwicklung) an einem Ort zusammenfallen. Diese Entscheidung hat über 20 Jahre Bestand.

Wenn nun feststeht, was man im Informatikunterricht zu welchem Zwecke macht, ist die Frage: wie soll man es machen? Welcher Unterrichtsablauf ergibt sich?

Nun, der Unterrichtsablauf ist in Phasen gegliedert, die dem Softwareentwicklungsprozess nachempfunden sind:

> "Der Problemlöseprozeß kann in folgende Phasen gegliedert werden [...]:
> - Problem- und Zielformulierung,
> - Problemanalyse und Modellbildung,
> - Algorithmierung,
> - Codierung und Implementation,
> - Benutzungsphase." (Riedel S.38)

Nebenbei: Im obigen Zitat ist es praktisch möglich, anstelle des Wortes Problemlöseprozess in der ersten Zeile die Worte Softwareentwicklung oder sogar Modellieren einzusetzen.

Dieses unterrichtliche Vorgehen wird durch die Techniken 'Zergliedern in Teilprobleme' und 'Schrittweise Verfeinerung' unterstützt (vgl. Balzert, 1987, S.31-35. Eberle, 1996 S.95f (Koerber), 98f (Schubert), 104 (van Lück)).
Soweit hat sich nun ein recht abgerundetes Bild ergeben. Aber man wollte mit dem anwendungsorientierten Ansatz mehr erreichen: Gesellschaftliche Auswirkungen der Informatik sollten durch eine Hinwendung zur Modellierung und zu softwaretechnischen Methoden integrativ behandelt werden können.
Diese Frage ist im unterrichtlichen Gang jedoch ein Anhängsel geblieben. Dieses zeigt etwa die Kritik Baumanns, der eine Art Versozialwissenschaftlichung des Informatikunterrichts fürchtet (vgl. etwa Baumann, 1996, S.181).
Auch Forneck kommt in einer Untersuchung des anwendungsorientierten Ansatzes zu dem Schluss, dass "es in den untersuchten Unterrichtsreihen nicht gelingt, nach einer Algorithmisierung und Programmierung diese Tätigkeiten auf gesellschaftliche Fragen zurückzubeziehen" (Forneck, 1992, S.229).
Dies zeigt (für unsere Skizze), dass die Fragen nach den Auswirkungen oft nicht aus den informatischen Inhalten und Herangehensweisen motiviert werden und daher auch nicht auf der Grundlage der Unterrichtsinhalte des Informatikunterrichts beantwortet werden können.
Zudem ergibt sich die Frage, wie das Verhältnis zwischen Informatik und Gesellschaft gesehen werden kann bzw. soll.
Zum Teil steckt in der Aussage, die Wirkungen der Informatik sollen betrachtet werden, bereits eine Konzeption dieses Verhältnisses: Demnach ist die Informatik Ausgangspunkt von Veränderungen in der Gesellschaft. Das läuft im Grunde auf einen Technik-Determinismus (vgl. Chandler, 1996) hinaus:

> "According to technological determinists, particular technical developments, communications technologies or media, or, most broadly, technology in general are the sole or prime antecedent causes of changes in society, and technology is seen as the fundamental condition underlying the pattern of social organization.
> Technological determinists interpret technology in general and communications technologies in particular as the basis of society in the past, present and even the future. They say that technologies such as writing or print or television or the computer 'changed society'. In its most extreme form, the entire form of society is seen as being determined by technology: new technologies transform society at every level, including institutions, social interaction and individuals. At the least a wide range of social and cultural phenomena are seen as shaped by technology. 'Human factors' and social arrangements are seen as secondary."
> (http://www.aber.ac.uk/media/Documents/tecdet/tdet02.html)

Hier haben wir nun den nächsten Pinselstrich für unsere Skizze: Das Thema 'Informatik und Gesellschaft' wird technisch-deterministisch konzipiert. Im Grunde ist das nicht überraschend, wo doch sämtliche (eigentlich ja - da die zu vermittelnden Fähigkeiten bezeichnend - kognitive) Problemlösefähigkeiten auf formale softwaretechnische Methoden abgebildet werden.

In diesem Zusammenhang bekommt auch das Wort Baumanns von der 'Versozialwissenschaftlichung' einen anderen Klang: Wenn man tatsächlich die gesellschaftlichen Fragen nur ungenau, unscharf, unformal und im gewissen Sinne nicht formal-methodisch, nicht technisch-deterministisch behandeln kann, dann führt deren Thematisierung im Informatikunterricht zwangsläufig zu einem Bruch im unterrichtlichen Ablauf, der ja bislang (im Sinne unserer groben Skizze sogar ausschließlich!) von den formalen Schritten der Softwareentwicklung geprägt ist.

Zusammenfassung: Modellieren im anwendungsorientierten Ansatz beginnt mit der Setzung, den Unterrichtsablauf an das Wasserfallmodell der Softwareentwicklung zu koppeln. Damit wird der Begriff des Problemlösens nach und nach gleichbedeutend mit dem Begriff Softwareentwicklung. Problemlösendes Denken wird gleichgesetzt mit algorithmischem Denken.

Schließlich kann die unterrichtliche Softwareentwicklung als Denkschulung bezeichnet werden. Das zweite Standbein des Ansatzes besteht in der Absicht, das Thema der gesellschaftlichen Auswirkungen in den Unterrichtsablauf (also in das Erstellen von Problemlösungen) zu integrieren. Diese Absicht konnte jedoch nicht zufrieden stellend umgesetzt werden. Zudem beruht sie auf einer Weltsicht, die Technik und Gesellschaft in ein unangemessenes deterministisches Verhältnis setzt.

2. Modellieren und Denkschulung

Nicht zuletzt stellt sich die Frage, ob der Unterrichtsablauf tatsächlich die intendierten kognitiven Problemlösefähigkeiten herausbildet. Diese Frage wird als das Transferproblem bezeichnet: Wenn ein Schüler im Unterricht gezeigt hat, dass er die Schritte der Softwareentwicklung an einigen Softwareentwicklungsproblemen durchführen kann, diese Probleme also lösen kann, kann er dann auch Probleme aus anderen Bereich lösen? Sind seine Fähigkeiten transferierbar? Kann er beispielsweise das Problem lösen, ob er nach der Schule ein Informatikstudium aufnehmen soll?

Die Transferforschung bezüglich der These, Softwareentwicklung fördere Problemlösefähigkeiten, zusammenfassend zieht Eberle folgende Schlussfolgerung:

> "Beim Programmieren sind die Problemlöse- und Denkfähigkeiten an die Strukturen der Programmiersprache gebunden[2]. Diese wiederum sind nicht auf die Problemlösestrukturen aller Probleme generalisierbar. Während wir an anderer Stelle [...] vor allem das erste negative Ergebnis betont haben, gilt es auch, deswegen nicht einfach den formalen Bildungswert dieser Inhalte in Frage zu stellen, sondern in positiver Weise den gefundenen spezifischen Transfer zu betonen: Demnach ist Programmieren/Algorithmik geeignet, a) spezifische Problemlösefähigkeiten für andere Bereiche herauszubilden und b) wie in anderen Fächern auch einen aufgrund des jetzigen Forschungsstandes nicht größeren, aber auch nicht kleineren Beitrag zur Entwicklung allgemeiner Problemlösefähigkeiten zu leisten." (Eberle, 1996, S. 212)

Eberle klärt leider nicht sein Verständnis der transferierbaren Problemlösefähigkeit, in seinen Empfehlungen zur Methodik schränkt er Problemlösen bezeichnenderweise ein auf die "Lösung algorithmischer Problemstellungen" (Eberle, 1996, S. 329).

Die von Eberle herangezogenen Untersuchungen (Eberle, 1996, S.201-212) beziehen sich hauptsächlich auf Logo, aber auch auf Pascal, Fortran, Basic, zum Teil auf Standardanwendungen und Modellbildungswerkzeuge.
Insgesamt ordnet Eberle sein Konzept zum Problemlösen (wohlgemerkt: nach Analyse auch späterer Ansätze) in den anwendungsorientierten Ansatz ein. Demnach soll die "inhaltliche Struktur der informatischen Problemlösung als gleichzeitige Ablaufstruktur für den Unterricht" dienen (Eberle, 1996, S.331. Vgl. auch die Empfehlung S. 339 und das Beispiel auf S. 397).
Trotz manchmal anders lautender Berichte scheint sich demnach der in den achtziger Jahren entwickelte anwendungsorientierte Ansatz zumindest in Bezug auf das algorithmische Problemlösen bewährt zu haben - ansonsten hätte Eberle diesen Ansatz in 1996 nicht so direkt in seine Konzeption und seine Empfehlungen für den Informatikunterricht übernehmen können. Andererseits wird deutlich, wie sehr der Begriff Problemlösen an den Algorithmus-Begriff und an das imperative Programmieren bzw. die strukturierte Softwareentwicklung nach dem Wasserfallmodell gebunden ist[3].

---

[2] Weiter hinten (S.426f) schränkt Eberle allerdings die Bedeutung der Programmiersprache als nachrangig ein. Die Programmiersprache diene nur als "Vehikel zur Visualisierung algorithmischer Grundstrukturen".

[3] Andere Sprachkonzepte, insbesondere deklarative, werden zwar eingefordert (etwa Schubert 1991), dienen aber eher als Ergänzung des imperativen Ansatzes (vgl. auch Baumann, 1996, S.239ff).

In der Softwaretechnik jedoch haben sich die Methoden stark der Objektorientierung zugewandt: Objektorientierte Sprachen haben rein imperative abgelöst (vgl. Quibeldey-Cirkel, 1994. S.12), objektorientierte Vorgehensmodelle haben das Wasserfallmodell abgelöst (vgl. Quibeldey-Cirkel, 1994. S.105)[4]. Die ursprüngliche Idee des anwendungsorientierten Ansatzes, das Problemlösen im Informatikunterricht an die Methoden der Informatik anzukoppeln, ist jedoch nicht in Frage gestellt worden.
Welche Änderungen im Informatikunterricht könnte nun der Wechsel zu objektorientierten Methoden nach sich ziehen? Diese Änderungen könnten die Unterrichtsziele, -methoden und -inhalte betreffen.
Zunächst stellt sich die Frage nach unterschiedlichen Vorgehensweisen in der Softwareentwicklung, die ein verändertes unterrichtliches Vorgehen und andere Unterrichtsmethoden zur Folge haben können.
Ein für den Vergleich verschiedener objektorientierter Vorgehensmodelle in der Softwaretechnik entwickeltes Schema lässt eine bekannte Phasierung erkennen: "Für den neutralen Vergleich wurden die fünf Phasen Voruntersuchung, Systemanalyse, Entwurf, Erzeugung und Einführung gewählt" (Noack/Schienmann, 1999, S.170). Die groben Ablaufstrukturen der beiden hier unterschiedenen Paradigmen lassen zunächst also keine Unterschiede erkennen. Im Detail allerdings ergeben sich Abweichungen.
In den objektorientierten Vorgehensmodellen sind explizit Schleifen bzw. Iterationen möglich, der Problemlöseprozess wird stärker als ein heuristischer Prozess aufgefasst: als Suchen und Finden von Lösungsideen, die überprüft, verändert und verworfen werden können. Demgegenüber stellt sich das Wasserfallmodell eher als ein logisch-deduktiver Prozess dar.
Objektorientierte Vorgehensmodelle versuchen stärker (in Abgrenzung zum algorithmischen Problemlösen gesehen) Softwarequalitätseigenschaften wie Benutzbarkeit, Wiederverwendbarkeit, Flexibilität und Wartbarkeit zu berücksichtigen, und sie weisen der Phase Voruntersuchung (Anforderungsanalyse) eine stärkere Bedeutung zu, denn: "Man weiß, daß der Anfangszustand verändert werden muß, kennt aber den Zielzustand bloß vage oder gar nicht, allenfalls sind globale Zielkriterien bekannt, Komparative wie 'besser' oder 'effizienter'" (Quibeldey-Cirkel, 1999, S.47).
Betrachten wir zunächst die Auswirkungen der veränderten Vorgehensweise auf die Anforderungen, die an Softwareentwickler gestellt werden, demzufolge im Unterricht besonders trainiert und als Unterrichtsziele explizit ausgewiesen werden könnten. Diese induktive Art der Bestimmung von Unterrichtszielen reicht natürlich als Begründung für deren Setzung nicht aus. Bevor ich einige Anknüpfungspunkte für eine bildungstheoretische Begründung nenne, lege ich dar, wie sich die Unterrichtsziele im Bereich der Problemlösefähigkeiten ändern:
Fokussiert man die formalen Bildungsziele auf das problemlösende Denken, dann kann folgende Hypothese aufgestellt werden: Man kann zwischen einfachen und komplexen Problemen sowie (damit verknüpft) zwischen linear-analytischem und vernetztem Denken unterscheiden[5]. Objektorientierte Methoden fordern und fördern Letzteres stärker als algorithmenorientierte Herangehensweisen.
Was ist vernetztes Denken? In einer Arbeitsdefinition kann vernetztes Denken "als ein situationsbezogenes Denken verstanden werden, bei dem die in einer spezifischen Denksituation relevanten Elemente zueinander in Relation gesetzt, integriert und zu einem Gesamtzusammenhang verflochten werden" (Möller, 1999 S.28). Gegenüber "linear-analytischem - auf logische-deduktive Kausalketten zurückführbarem - Denken" berücksichtigt vernetztes Denken Wechselwirkungen einzelner Teilaspekte und Gesamtzusammenhänge (Möller, 1999, S.28).

---

[4] Dort heißt es: "..Zum anderen verläuft der objektorientierte Entwurfsprozess iterativ: Die zahlreichen Durchläufe verschmelzen zunehmend die Phasenübergänge."
[5] Krasemann spricht in einem ähnlichen Zusammenhang von strukturellem Denken. Krasemann, 1997, S.331.

Durch die Thematisierung von Fragen, die über die algorithmische Problemlösung hinausgehen (Softwarequalitätskriterien und Einbettung in Anwendungskontext) kann eine objektorientierte Methodik möglicherweise komplexere Denkweisen fordern und fördern: Die Anzahl der zu beachtenden Kriterien wäre größer, das Erkennen der Abhängigkeiten zwischen den Kriterien sowie deren Gewichtung wären gefordert[6].
Eine andere Sichtweise auf Software bietet sich an: Gegenüber einer Datenstruktur, auf der Operationen angewendet werden, entsteht eine Welt aus interagierenden Objekten, die durch Ereignissteuerung nichtlineares Verhalten ermöglicht. Ein Objekt wird dabei typisiert als Klasse definiert. Einander ähnelnde Klassen werden in eine hierarchisierende Vererbungsbeziehung gestellt.
Vor diesem Paradigma könnte objektorientiertes Modellieren konzeptuell vernetztes Denken trainieren (vgl. Möller, 1999), denn die interagierenden Objekte sollen ein benutzbares Softwaresystem ergeben, so dass eine sinnvolle Vernetzung der Objekte hergestellt werden muss.
Möglicherweise erfordert also bereits die objektorientierte Quelltext-Organisation eine Art vernetztes Denken: Gegenüber imperativer Software, die wie eine Art Orchester funktioniert, dass zentral von einem Dirigenten geleitet wird, funktioniert ein objektorientiertes Programm eher wie ein Jazz-Ensemble aus individuellen Solisten, deren Zusammenarbeit die Gesamtfunktionalität ergibt[7]. Die Organisation dieser Zusammenarbeit erfordert die Vernetzung der Objekte und damit vom Entwickler vernetztes Denken.
Die eigentliche Chance der Objektorientierung dürfte jedoch eher in solchen objektorientierten Vorgehensweisen der Softwareentwicklung liegen, die versuchen, die Kunden einzubeziehen und kooperativ (= Entwickler und Kunden) Ziele und Anforderungen an die zu entwickelnde Software zu bestimmen. Die Kunden werden in den Entwicklungsprozess einbezogen, um die unscharfen oder unklaren Anforderungen zu bestimmen.
Durch die Übertragung dieser Unklarheit in den Unterricht besteht die Möglichkeit zu einem kreativen Problemfindungsprozess. Lewis/Petrina/Hill vermuten, dass durch ein solches kreatives Element des 'Problem posing', welches nicht nur Problemfindung, sondern auch Umformulierung von Problemen einbezieht, der Unterricht lernwirksamer wird. Sie zitieren entsprechende Studien:

> "These studies have shown that when technological problem solving was taught through design and was open-ended, thereby allowing for problem posing and exploration, student creativity was fostered, and this enriched student learning."

Entsprechende Problemklassen sind zu unterscheiden:

> "well structured problems, which required convergent thinking and could be solved via algorithms and ill-structured problems, which required divergent thinking and solution via heuristics".

Die unterschiedlichen Problemklassen erfordern in der Konsequenz nicht nur unterschiedliche Denkstrategien, sondern entsprechend unterschiedliche Unterrichtsabläufe. Lewis/Petrina/Hill folgern, dass Unterricht sich von festen Schemata und vom Lehrer vorgegebenen Problemen wegbewegen sollte hin zu

> "context-bound and situation-specific models [...] where the student as learner plays an important and active role". (Lewis/Petrina/Hill, 1998)

---

[6] Die Anzahl der Kriterien, deren Vernetzung und Gewichtung sind Kriterien für komplexes Denken.
[7] Die Metapher 'Orchester vs. Jazz-Ensemble' stammt aus: Bellin, David; Simone, Susan Suchmann: The CRC card book. Addison Wesley, 1997. S. 8f.

Um den Erwerb von kognitiven Fähigkeiten zu stärken, schlägt Johnson (vgl. Johnson, 1992, S.32f.) fünf Prinzipien vor, die in unserem Zusammenhang für den Informatikunterricht interessant sein können:

1. Help Students Organize Their Knowledge
2. Build On What Students Already Know
3. Facilitate Information Processing
4. Facilitate "Deep Thinking"
5. Make Thinking Processes Explicit

Die verschiedenen Prinzipien korrespondieren recht gut mit verschiedenen Aspekten der durch Modellieren beabsichtigten formalen Bildung im Sinne des Trainings von kognitiven Kompetenzen: Prinzip 1 kann als eine Begründung des Ansatzes von Hubwieser verstanden werden, Notationen und Techniken der Informatik als Werkzeuge zum Umgang mit Information zu vermitteln. Gerade wenn ungenau definierte Probleme bearbeitet werden müssen, spielt die Art der Repräsentation des Problems eine wichtige Rolle beim Finden der Lösung. Prinzip 2 zeigt auf, dass die Auswahl lerngruppenadäquater Probleme eine entscheidende Grundlage für den unterrichtlichen Verlauf ist. Prinzip 3 spielt darauf an, die einzelnen Modellierungstechniken, Notationen und Werkzeuge so zu vermitteln und zu verankern, dass deren Nützlichkeit und Übertragbarkeit von den Lernenden erkannt wird, damit sie später in entsprechenden Situationen zugreifbar sind. Unter Prinzip 4 schlägt Johnson vor, bestehendes Material weiterzuentwickeln, da eine solche Aufgabe erfordert, das vorhandene Material zu verstehen. Re-Engineering-Aufgaben bieten sich an. Das fünfte Prinzip schließlich meint, dass nicht nur Problemlöse- bzw. Modellierungsverfahren vermittelt werden sollen, sondern auch "how, when, where, and why the strategy should be employed". Dieses Prinzip schließt m. E. das Nutzen von Erfahrungswissen, sprich: von Entwurfsmustern, mit ein und zeigt auf, an welcher Stelle und wie Muster in den Unterricht integrierbar sind.

Objektorientierte Ansätze der Modellierung bieten sich demnach dort an, wo Probleme 'offen' sind, wo unterschiedliche Lösungen und gegebenenfalls sogar unterschiedliche Beurteilungskriterien möglich sind. Das heißt, dieses Vorgehen bietet sich an, um die Gestaltung von Software zu thematisieren. Damit ändern sich Unterrichtsziele und -abläufe. Die Unterrichtsmethodik muss sich wegbewegen von der bewährten, am Wasserfallmodell orientierten, Sequenzierung in entsprechende Unterrichtsphasen.

Zusammenfassung: Es ist nicht sicher, dass durch Programmieren transferierbare Problemlösefähigkeiten vermittelt werden. Die Softwaretechnik hat sich zudem weitgehend von Wasserfallmodellen verabschiedet. Im Informatikunterricht scheint das anwendungsorientierte Vorgehen dennoch bevorzugt zu werden. Der Wechsel zu objektorientierten Methoden kann jedoch andere Problemlösefähigkeiten fördern: Vernetztes Denken. Diese Vermutung gründet sich zu einem gewissen Grad auf die vernetzte objektorientierte Quelltextorganisation, stärker jedoch auf iterative Softwareentwicklungsmethoden, die Rolle von Softwarequalitätskriterien und den Gedanken, dass Software nicht ohne Berücksichtigung des Einsatzkontextes entwickelt werden kann. Unterrichtsmethodisch kann dieses Lernziel durch die Gelegenheit zu kreativen Problemfindungsprozessen und Prinzipien zum Training kognitiver Fähigkeiten unterstützt werden. Damit wird auch die Bindung des Unterrichtsablaufs an Wasserfallmodelle in Frage gestellt. Welche alternativen Modelle für den Unterrichtsablauf in Frage kommen, welche anderen Abläufe aus der Softwaretechnik Pate für Unterrichtsabläufe sein können, ist offen.

3. Modellieren und Softwaretechnik

Wie sehen denn andere, in der Objektorientierung verwendete Vorgehensmodelle aus? Oder, anders gefragt: Welche Aspekte aus der Softwaretechnik sind für die bisher erarbeitete Unterrichtskonzeption nutzbar?

Quibeldey-Cirkel behauptet: "Als 'schädlich' für den Entwurf komplexer Systeme gilt das algorithmische Denken an sich" (Quibeldey-Cirkel, 1995), denn "algorithmisches Denken schließt den Kunden vom Entwurfsprozess aus".
Ein Modell für die Entwicklung von Software zur Unterstützung von Geschäftsprozessen (vgl. Deiters, 1997) zeigt beispielhaft das mögliche Vorgehen: Zunächst erfolgt eine Analyse des Ist-Zustands, dessen Ergebnis ein Geschäftsprozess-Modell ist. Auf der Basis dieser Ist-Analyse wird ein Soll-Modell entwickelt, welches die Geschäftsprozesse zu optimieren versucht - und erst auf der Basis dieses Modells beginnt die 'eigentliche' Softwareentwicklung. Die Ergebnisse einer Phase, die einzelnen Modelle sind eher 'Modelle für' als 'Modelle von' (vgl. Schefe, 1999, S.132f): Sie dienen als Grundlage für den nächsten Schritt, sind Vorbild für die weitere Arbeit anstelle Abbild der Ausgangslage. Das trifft selbst auf die Ist-Analyse zu, denn in der Ist-Analyse werden diejenigen Aspekte betrachtet, von denen man annimmt, dass sie im weiteren Verlauf wichtig sind.
Aus Sicht der Softwaretechnik sind neben dem Abbilden der Kundenanforderungen allgemeine Softwarequalitätskriterien, Erfahrungswissen sowie die Eigenschaften der Werkzeuge zu berücksichtigen. Aus Sicht der Auftraggeber werden nicht (nur) vorhandene Tätigkeiten oder Arbeitsabläufe automatisiert, also 'in den Rechner' abgebildet. Es werden Infrastrukturen geschaffen, die in vorhandene Abläufe eingreifen und diese ändern. Dies gilt zwar nicht für jegliche Projekte, jedoch für die E-Type-Systems: Für Software, die in einen Kontext eingebettet wird (Dazu mehr im nächsten Abschnitt).
Die sich ergebende Frage, ob Software Realität abbildet oder Arbeitsplätze gestaltet, soll hier nicht in ausschließender Art diskutiert werden. Es geht um die Suche nach Sichtweisen auf Informatik und Softwaretechnik, die geeignet sind, die unterrichtlichen Ziele zu stützen. Ich vermute, dass eine Konzentration auf abbildende Aspekte den Softwareentwicklungsprozess zu sehr aus der Sicht der Softwareentwickler darstellt, so dass eine einschätzende Beurteilung auch im Sinne der Diskussion von Auswirkungen nicht so leicht in den Unterrichtsablauf zu integrieren ist.
Doch bevor das Thema der gesellschaftlichen Auswirkungen im nächsten Abschnitt diskutiert wird, zeichne ich im Folgenden inner-informatische Argumente gegen eine rein abbildende Sichtweise nach. Man kann eine Art abbildende Sichtweise auf folgenden Aspekt richten: Demnach bildet Software "soziales Verhalten in Daten" (Keil-Slawik, 2000) ab. Passender ist jedoch die Ansicht Schefes, wonach die stattgefundene Tätigkeit als normierende Beschreibung von Handlungszusammenhängen (Schefe, 1999, S.122 und S.132) aufgefasst wird. Diese Ansicht berücksichtigt stärker den oben angesprochenen Unterschied von Ist-Modell und Soll-Modell. Man könnte den Begriff Abbilden auch auf das Abbilden des geplanten Systems beziehen. Anstelle von 'Abbilden des geplanten Systems' trifft m. E. der Begriff Gestaltung den Sachverhalt besser. Dazu Keil-Slawik: "Die der Software zugrunde liegenden Modelle menschlichen Verhaltens verändern dieses mehr oder weniger stark [...] Eine Rückbezüglichkeit entsteht" - diese Formulierung fokussiert eher auf ungewollte Nebenwirkungen, die aus dem Software-Einsatz entstehen können, negiert so aber gerade die Tatsache, dass verändernde Wirkungen gewollt sind und im Soll-Modell gestaltet werden[8].

[8] Wobei offen bleibt, ob nun der Softwareentwickler oder der Auftraggeber der Gestalter dieser verändernden Wirkungen ist. Möglicherweise führt der Softwareentwickler nur aus, möglicherweise findet eine Art 'partizipative Systemgestaltung' statt.

Entwurfsmuster, wie von Gamma et. al. vorgestellt, haben gerade den Zweck, die Qualität eines Entwurfs (insbesondere bezüglich Flexibilisierung und Wartbarkeit) dadurch zu erhöhen, dass der Entwurf auf allgemein bekannte und bewährte Muster aufbaut und somit auf genaues Abbilden verzichtet. Sie stellen fest:

> "Die strikt an der realen Welt orientierte Modellierung führt zu einem System, das vielleicht die heutige Realität wiedergibt, aber nicht unbedingt die von morgen. [...] Abstraktionen sind der Schlüssel dazu, einen Entwurf flexibel zu machen". (Gamma et. al., 1996, S.14)

Und diese Abstraktionen führen oft zu Klassen, "für die es keine Entsprechung in der realen Welt gibt" (Gamma et. al., 1996, S.14).

Insgesamt muss man jedoch feststellen, dass die Sichtweise, informatisches Modellieren sei Abbilden von Realität, sehr oft anzutreffen ist. Wedekind u.a. vermuten dazu:

> "Der Grund, warum dieser schlichte abbildungstheoretische Realismus trotz seiner offensichtlichen Schwächen so weit (nicht nur in den Naturwissenschaften) verbreitet ist, liegt wohl drin, daß der entscheidende Schritt, der dem Modellieren vorausgeht, als relativ unproblematisch gilt". (Wedekind, 1998, S.267)

Dabei gelte für den Naturwissenschaftler:

> "[I]hre Realität ist immer durch disziplinspezifische Versuchs- und Beobachtungskontexte gegeben, [...] so daß man bei dem Stichwort 'Realität' im Grunde doch nur an *Beschreibungen* disziplinrelevanter Sachverhalte denkt" (Wedekind, 1998, S.267).

Bei der Softwareentwicklung sind diese disziplinrelevanten Sachverhalte die Anforderungen des Kunden bzw. Auftraggebers, die in Software abgebildet werden. Unter der Prämisse, die Anforderungsentwicklung sei ein der Softwareentwicklung vorgelagerter Schritt, würde man sich darum keine Gedanken machen. Wenn aber die Anforderungsermittlung Auswirkungen hat auf die Art, wie Software entwickelt wird, auf die Beurteilung der Softwarequalität, auf die Anforderungen an die Entwickler, dann kann die 'Analyse des Problembereichs' oder ein ähnlicher Schritt nicht ausgelassen werden, will man ein angemessenes Bild der Softwareentwicklung und insbesondere des Modellierens vermitteln.

In Bezug auf die Objektorientierung kommt hinzu, dass (im Vergleich mit dem strukturierten Ansatz) das Besondere gerade die Hinwendung zum Problembereich, zur Anwendungsdomäne, zum Realitätsausschnitt und in der Abkehr von den Besonderheiten der Maschine besteht. Darin liegt der Grund, weshalb die Übersetzungen zwischen Analyse, Design und Implementation(-s"-Modell") leichter fallen.
Verständlich, dass dieser Aspekt denn auch in den allermeisten Lehrbüchern zur Objektorientierung betont wird. Damit wird auf die Besonderheiten der Objektorientierung in Abgrenzung zum strukturierten Ansatz fokussiert. Im Grunde richten sich die entsprechenden Lehrbücher an einen bereits 'informatisch sozialisierten' Adressatenkreis. Für eine erste Begegnung mit Informatik jedoch gilt eine umgekehrte Richtung: Nicht von den bisher im Vordergrund stehenden maschinellen Aspekten soll der Blick auf das Modellieren des Anwendungsbereichs gelenkt werden, sondern gerade umgekehrt vom alltäglichen Erfahrungsbereich auf die Besonderheiten von Software[9]. Da dürfte helfen, zur Vermittlung der Besonderheiten von Software darauf hinzuweisen, dass Modellieren auch Gestalten und nicht nur Abbilden ist.
Ein Beispiel für ein solches Vorgehen kann aus dem Ansatz von Hubwieser abgeleitet werden, der den informationswissenschaftlichen Ansatz für den Informatikunterricht und insbe-

[9] Zu Besonderheiten von Software siehe auch die zitierten Arbeiten Keil-Slawik und Lehman.

sondere die Modellierung auf dem Modellierungsbegriff des hier zitierten Artikels von Wedekind et. al. aufbaut (Hubwieser, 1999, S.24). Hubwieser benutzt diesen Ansatz, um im Informatikunterricht den Umgang mit Informationen speziell unter dem Aspekt der von der Informatik entwickelten Beschreibungstechniken, Notationen, Sprachen und Vorgehensweisen einzuführen und einzuüben. Er legt einen Schwerpunkt auf grafische Notationen, wie sie auch von der Objektorientierung bevorzugt werden.
Das bedeutet für den Informatikunterricht, auch und zunehmend, in Werkzeuge einzuführen. Noack/Schienmann behaupten sogar: "Objektorientierte Anwendungsentwicklung ist ohne umfassende Werkzeugunterstützung nicht denkbar" (S.177). Im Informatikunterricht werden Modellierungswerkzeuge benötigt, um das zu lösende Problem und später die Problemlösung zu repräsentieren - meist grafisch. Aus der grafischen Darstellung der Problemlösung wird dann die Implementation erzeugt ('Das Modell umgesetzt', wie manchmal etwas hemdsärmelig gesagt wird).
Hier lauert nun eine weitere Schwierigkeit für eine Konzeption des Informatikunterrichts, auch wenn diese hier nur benannt werden kann, ohne eine Lösung anzubieten: Für die Beschäftigung mit den Werkzeugen der Informatik im Unterricht kann man zwei Ebenen unterscheiden: Zum einen die Ebene, auf der Ergebnisse der Informatik genutzt werden, und zum anderen die Ebene der informatischen Fragestellungen:

> "Es ist eine wesentliche Aufgabe der Informatik, Sprachkonzepte, Formalismen, Techniken und Werkzeuge zu entwickeln, die die Verständnisbildung so unterstützen, daß die Brücke zwischen der Welt der sinnfreien maschinellen Abläufe und ihrer sinnhaften Einbettung in menschliche Handlungszusammenhänge effektiv und verläßlich geschlagen werden kann und zwar sowohl auf der Seite der Herstellung als auch auf der Seite der Nutzung." (Keil-Slawik, 2000)

UML, als Beispiel für ein grafisches Modellierungswerkzeug, ist ein derzeit aktuelles Ergebnis des informatischen Bemühens, Hilfsmittel anzubieten für den beim Modellieren erforderlichen 'Brückenschlag' zwischen den Anforderungen des Kunden und der Realisierbarkeit der passenden Funktionalitäten auf der Maschine. Man kann nun überlegen, ob es Aufgabe des Informatikunterrichts ist, in die Nutzung von Ergebnissen in Form von Werkzeugen einzuführen oder durch das Umgehen mit solchen Werkzeugen Verständnis für wesentliche Aufgaben der Informatik zu wecken. Die beiden Aufgaben widersprechen sich natürlich nicht, sondern ergänzen sich.

Zusammenfassung: Objektorientierte Verfahrensweisen versuchen problemnah zu arbeiten und den Kunden in den Entwurfsprozess einzubeziehen. Beispielsweise werden Begriffe des Problembereichs in der Software benutzt. Daher wird oft darauf hingewiesen, dass Teile der Wirklichkeit abgebildet werden, Modellieren also Abbilden bedeutet. Diese Sichtweise ist jedoch in der Informatik umstritten und kann dazu verleiten, Objektorientierung als weiteres Problemlöseparadigma in den anwendungsorientierten Ansatz einzuordnen. Damit wäre in Bezug auf die Thematisierung der Wechselwirkungen zwischen Informatik und Gesellschaft nichts gewonnen.

4. Modellieren und gesellschaftliche Auswirkungen

Die im vorangegangenen Abschnitt mit Keil-Slawik angesprochenen Rückbezüglichkeiten, die Veränderungen menschlichen Verhaltens durch Softwareeinsatz, beschreiben bereits Aspekte der gesellschaftlichen Auswirkungen der Informatik.

Erkennbar werden diese in der Phase Voruntersuchung, in der es um die Anforderungen des Kunden geht. Software wird entsprechend den Wünschen des Kunden gestaltet, hier liegen

die Zwecke. An den Gestaltungsmethoden ist erkennbar: Software fügt sich in die Umgebung ein und ändert diese, so dass damit wiederum Änderungswünsche an die Software entstehen. Dementsprechend berücksichtigen die Methoden und die Qualitätskriterien, dass Software änderbar sein sollte.
Dies gilt nicht unbedingt für jede Software. Lehman unterscheidet zwischen verschiedenen Typen, von denen die 'E-Type-Systems' in diesem Rahmen interessieren: Software, über die man ohne Berücksichtigung des Einsatzkontextes keine Aussagen treffen kann. Es handelt sich um Software, die in soziale Kontexte eingebettet wird[10].
Aus zwei Gründen ist der Ansatz Lehmans in unserem Zusammenhang interessant: Zum einen stellt Lehman eine Taxonomie von Software-Typen auf, die nach dem Grad der Verzahnung der Software mit der Einsatzumgebung geordnet ist. Damit wird deutlich, dass Informatikunterricht einen bestimmten Typus von Software behandeln muss, soll eben diese Verzahnung vermittelt werden. Aufbauend auf dem Ansatz von Lehman kann man vermutlich Auswahlkriterien für entsprechende Unterrichtsbeispiele erarbeiten. Zum anderen gründet sich die Taxonomie auf langjährige empirische Studien, die hauptsächlich Software-Projekte untersuchen, die nach dem strukturierten Ansatz arbeiten. Damit wird deutlich, dass eben diese beobachtete Verzahnung von Software und Einsatzumgebung keineswegs, wie man eventuell vermuten könnte, auf Besonderheiten der Objektorientierung aufbaut. Man kann mit Lehman umgekehrt argumentieren, dass die Weiterentwicklung softwaretechnischer Methoden und Werkzeuge (auch) auf den sich aus dieser Verzahnung ergebenden Schwierigkeiten bei der Entwicklung von Software beruht.
Schefe spricht in diesem Zusammenhang sogar vom grundsätzlichen "Dilemma der Softwaretechnik [...], Unformalisierbares formal rekonstruieren zu müssen" (Schefe, 1999, S. 122). Zwischen den in der Anforderungsanalyse ermittelten unformalen Anforderungen und der formalen Spezifikation besteht eine nicht mathematisch fassbare Beziehung, die Schefe 'Sinnzuweisung' nennt, da es sich dabei um eine intentionale Bedeutungszuweisung handele. Die Aufgabe der Anforderungsanalyse sei daher, im Gegensatz zu einer naturwissenschaftlichen Abbildung eines Realitätsausschnitts, "bestimmte handlungsorientierte Konzeptualisierungen der Realität zu beschreiben. [...] Die Aufgabe ist, *Intentionen* zu verstehen, *Sinn* zu erfassen" (Hervorhebung im Orig., Schefe, 1999, S.123). Oder, um es mit Scott Ambler etwas handfester auszudrücken: Die Frage ist nicht mehr nur, wie man ein Software-System richtig konstruiert, sondern, wie man das richtige Software-System konstruiert[11]. Und diese Frage richtet sich an die Einsatzumgebung, wenn man so will, den gesellschaftlichen Kontext, den Handlungszusammenhang, in den die Software eingebettet ist.

> "Die geringe Wiederverwendbarkeit von Software hat ihre Ursachen nicht primär in unzulänglichen Programmiertechniken, sondern in Erkenntnissituationen. Es geht nicht um die Erkennung allgemeingültiger Gesetze und ('wiederverwendbarer') Lösungen, sondern um individuellen Situationen mit wandelbaren Zwecken anpassbare Mittel" (Schefe, 1999, S.123).

Software als Mittel zu sehen, das an individuelle Situationen und wandelbare Zwecke angepasst ist oder sein sollte, liefert den Schlüssel, um die Auswirkungen der Informatik so diskutieren zu können, dass die Diskussion informatisch geführt werden kann: Dies betrifft zum einen die Softwarequalitätseigenschaften, zum anderen die Frage nach der Angemessenheit der Mittel. Die letzte Frage bezieht sich auf die Eignung im gegebenen Kontext ebenso wie auf die Veränderungen im Kontext (Stichwort: Business-Reengineering). Damit werden E-Type-Systems als Mittel zur Gestaltung von Arbeitsplätzen eingeordnet. Diese Sichtweise ist der Grund, weshalb Modellieren als Gestalten, und nicht nur als Abbilden einer vorhandenen

---

[10] Daher das 'E-Type' - der Begriff 'embedded systems' bezeichnet Software, die in Maschinen eingebettet ist.
[11] "Developers are good at building systems right. What we're not good at is building the right system." (Ambler, 1995, S.1)

Realität (hier: als Abbilden von Arbeitsplätzen und Arbeitstätigkeiten) thematisiert werden sollte.

Die Gestaltung von Software berücksichtigt also den geplanten Einsatzkontext. Zu diesem Kontext gehören nicht nur die materiellen Gegebenheiten wie Hard- und Software, sondern auch die sozialen Gegebenheiten wie Arbeitsabläufe und verschiedene Rollen der Benutzer. Im Modellierungsprozess wird die geplante Software sozusagen kontextualisiert. Und dieser Prozess geht von zwei Seiten aus: Die Modelle (Anforderung, Analyse, Design) werden an die Umgebung angepasst - und andererseits kann auch die Umgebung angepasst werden: Arbeitsabläufe und Rollen der Benutzer können sich ändern. Die Sichtweise, Software (oder verallgemeinert: Informatik) gestaltet Arbeitsplätze, ist daher zu präzisieren, um nicht in einen Technik-Determinismus zu verfallen. Zwischen Software und Umgebung gibt es Wechselwirkungen, vernetzte Abhängigkeiten und nicht einseitige Ursache-Wirkungs-Beziehungen. In diesem Zusammenhang sollte man jedoch auch nicht in das andere Extrem fallen, dazu Chandler's Schlussbemerkung zum Technik-Determinismus:

> It is not very helpful to retreat to the extreme position that 'everything causes everything'. It is a great mistake to jump from the conclusion that the relationship between technology and society is not simple to the conclusion that the use of a particular technology in a specific context has no consequences at all. Any technological change which is great enough is likely to produce some social change. And some of these changes may be widespread and major. (http://www.aber.ac.uk/media/Documents/tecdet/tdet13.html)

Aus dieser Argumentation folgert Pannabecker:

> "The immediate task is not, [...] to find a single alternate metaphor but to recognize that there are different ways of approaching the study of technology and society" (Pannabecker, 1991, S.8).

Das bringt uns unter zwei Gesichtspunkten zurück auf den Anfang des Artikels: Modellieren als Denkschulung. Die eben skizzierte Sichtweise auf Software kann als eine Art Weltbild charakterisiert werden. Die Welt wird als ein vernetztes System aufgefasst, in dem zwischen den einzelnen Bestandteilen (hier unter anderem: Software und deren Benutzung) verschiedene Arten von Beziehungen herrschen.

So kann Informatikunterricht Gestaltungsaufgaben im Sinne der Klafki'schen Schlüsselprobleme thematisieren, welche - verallgemeinert aus der exemplarischen Behandlung von kleineren Aufgaben im Unterricht - im Grunde die Gesellschaft als Ganzes betreffen[12].
Hier nun finden wir bildungstheoretische Anknüpfungspunkte, die den oben dargestellten Wechsel der Unterrichtsziele legitimieren können. Zum einen betrifft das den Bereich des Trainings kognitiver Fähigkeiten, aber auch - wie wir in der Zwischenzeit gesehen haben - den Bereich des Verhältnisses zwischen Informatik (als Stellvertreter für Technik allgemein) und Gesellschaft:

> "Zur bildenden Auseinandersetzung gehört zentral die - an exemplarischen Beispielen zu erarbeitende Einsicht, daß und warum die Frage nach 'Lösungen' der großen Gegenwarts- und Zukunftsprobleme verschiedene Antworten ermöglicht [...]. Aus diesem Grundsachverhalt folgt allerdings keineswegs die umstandslose Anerkennung aller solcher Positionen als gleichberechtigt. Vielmehr stellt sich die Frage nach Kriterien, mit deren Hilfe die Geltung unterschiedlicher Lösungsvorschläge [...] beurteilt werden kann" (Klafki, 1996, S. 61).

[12] Vgl. dazu Klafki, insbesondere den Abschnitt 5 (S.56-69): Bildung im Medium des Allgemeinen: Konzentration auf epochaltypische Schlüsselprobleme. Klafki selbst nennt die "Gefahren und die Möglichkeiten der neuen technischen Steuerungs-, Informations- und Kommunikationsmedien" als Beispiel (S.59).

Daraus folgt, dass es "auch um die Aneignung von Einstellungen und Fähigkeiten [geht], deren Bedeutung über den Bereich des jeweiligen Schlüsselproblems hinausreicht" (Klafki, 1996, S.63f). Klafki nennt: Kritikbereitschaft und -fähigkeit, Argumentationsbereitschaft, Empathie und schließlich

"'vernetzendes Denken' oder 'Zusammenhangsdenken' [...]. Die Betonung dieser Fähigkeit ergibt sich zwingend aus neueren Zeit- und Gesellschaftsanalysen, die jene vielfältigen Verflechtungen herausgearbeitet haben, die heute, im Zeitalter hoch entwickelter Technik und ihrer möglichen Folgen sowie der damit verbundenen politischen und ökonomischen Wirkungszusammenhänge - zugespitzt formuliert - 'alles mit allem' verknüpfen" (Klafki, 1996, S.63f).

Als ein Ergebnis unseres Vergleichs zwischen der anwendungsorientierten Konzeption des Informatikunterrichts mit einer anderen, einer gestaltungs- oder systemorientierten Konzeption, stellen wir fest: Der spezifische Beitrag des Informatikunterrichts zur Allgemeinbildung ist, ob nun gewollt oder nicht, gebunden an eine bestimmte Sichtweise auf die typischen Probleme einer technisierten Gesellschaft, und auf mögliche Lösungen dafür. Man kann sogar vermuten, dass im Informatikunterricht implizit und unvermeidbar Ansätze für ein bestimmtes Weltbild vermittelt werden.

Unter Bezugnahme auf Frederick Vester weist etwa Möller darauf hin,

> "daß mit linearem bzw. vernetztem Denken konträre Weltbilder verknüpft sind. Lineares Denken kann auf ein technokratisches Weltbild zurückgeführt werden, demzufolge Teilaspekte der Wirklichkeit als isoliert nebeneinander stehend betrachtet werden und es für Wirklichkeitsphänomene eindeutige Erklärungen, für Probleme einfache Lösungen gibt."

Die Welt stellt sich demzufolge dar "als übersichtliches, geordnetes, zwar dynamisches, aber im wesentlichen präjudizierbares Gefüge".

In einer vernetzten Weltsicht dagegen wird die Welt

> "als System vernetzter natürlicher Abläufe begriffen, bei dem die zwischen Teilbereichen der Realität bestehenden Beziehungen mindestens ebenso wichtig für ein Gesamtverständnis sind wie die Teilbereiche selbst.[...] [Zwischen den Teilbereichen gibt es] Wechselwirkungsprozesse, Rückkopplungen, direkte oder indirekte (Kausal-)Beziehungen" (Möller, 1999, S.28)[13].

Die Art, in der Modellieren bzw. Problemlösen gelehrt wird, hängt also mit einem Weltbild, oder vielleicht besser: mit einem spezifischem Verständnis von Informatik zusammen. Das Verständnis der Informatik und der Einsatzweise von Computern hat sich von der Sichtweise des Ersetzens zu einer Sichtweise des Unterstützens gewandelt und bewegt sich damit weg von einer ausschließlich auf Algorithmen fixierten Sichtweise.

Der zweite Aspekt unter dem wir uns wieder auf den Anfang des Artikels beziehen, betrifft die Anforderungen an die Softwaregestaltung: Ereignisgesteuerte, objektorientierte Software muss so gestaltet werden, dass sie mit den Handlungsabläufen des Einsatzbereichs vernetzt

[13] Möller (S. 41) warnt: "Die gesellschaftlich-globalen Implikationen der Fähigkeit zum konzeptuellen vernetzten Denken könnten in Verbindung mit der Gegenüberstellung von vernetzten und linear-analytischem Denken [...] zu dem Schluß verleiten, daß es nur eine kategorisch richtige Denkform gibt, nämlich die des konzeptuell-vernetzten Denkens. Eine derart glorifizierende Sichtweise wird jedoch von DÖRNER überzeugend in Frage gestellt. DÖRNER fordert eine 'Flexibilität der Denkweisen', nicht eine Einseitigkeit, indem er betont: 'Manchmal ist das analytische Denken gefragt, das Denken im Detail und zu anderen Zeitpunkten braucht man die Synthese, das >Denken im großen Stil<, wo Details ganz bewußt übersehen werden müssen.' (1989a, S.49; vgl. auch 1989b, S.298f)"

ist. Diese Kontextualisierung bedarf vernetzten Denkens. Darin liegen die Schwierigkeiten des Entwurfs und des Lernens. Andererseits liegt hier auch ein erstes Anwendungsfeld für ein im unterrichtlichen Softwareentwurf trainiertes vernetztes Denken: Die Frage nach den Auswirkungen des Einsatzes einer Software und der Qualität der vorliegenden Software muss Wechselwirkungen zwischen der Software und dem Kontext erfassen.
Die Fragen betreffen die Flexibilität der Software in den Bereichen, in denen Änderungen erforderlich, wünschbar sind. Und sie betreffen die Passung in den Kontext: Hier ist zu sehen, dass die beobachtete Qualität der Software auch mit der Art der Benutzung zusammenhängt. Ebenfalls ist zu berücksichtigen:

> "[P]roblem solving and product design are not the same; the best result of a sound problem solving process is often something other than a new product" (Flowers, 1998).

Modellieren als Teil des Softwareentwicklungsprozesses wird eingebettet in die Gestaltung von Informatiksystemen im Sinne der Lehman'schen E-Type-Systems. Die Beurteilung des Systems sowie die Suche nach Verbesserungen gehen über die Frage der Konstruktion eines Software-Produktes hinaus. Einerseits wird über die Erstellung eines Produktes hinausgedacht, indem von Anfang an die Entwicklung weiterer Versionen berücksichtigt wird. Dementsprechend sind die Softwarequalitätskriterien Flexibilität und Wartbarkeit einzuordnen. Die Kriterien Benutzbarkeit und Aufgabenangemessenheit betreffen andererseits die Passung in den Anwendungskontext.

Zusammenfassung: Die Berücksichtigung verschiedener Kriterien und das Erkennen der Vernetzung von Software und Einsatzumgebung erschließt einerseits exemplarisch die Thematik der Wechselwirkungen zwischen Informatik und Gesellschaft; auch im Sinne der Klafki'schen Schlüsselprobleme. Andererseits wird hier das vernetzende Denken gefordert und gefördert.

5. Fazit

So wie beim Verhältnis von Informatik und Gesellschaft oft keine einfachen Ursache-Wirkungszusammenhänge greifen, kann man auch nicht von *den* Auswirkungen und *den* Möglichkeiten durch die Objektorientierung sprechen. Es gibt jedoch Hinweise, dass durch eine Hinwendung zu offenen Fragestellungen (im Gegensatz zu algorithmisch lösbaren Problemen) sowohl die kognitiven Lernziele (Denkfähigkeiten), die informatischen Lernziele (Fragestellungen und Werkzeuge, Methoden der Softwaretechnik) und der Aspekt der Wechselwirkungen zwischen Informatik und Gesellschaft profitieren können.
Objektorientierung kann helfen, diese Öffnung zu ermöglichen. Die Fragestellungen könnten sich auf die E-Type-Systems beziehen, für welche die Lehman'schen "Laws of Software Evolution" gelten. Damit sind die Fragestellungen in einen Kontext eingebettet und mit Re-Engineering-Verfahren kann im Unterricht die Benutzung und Weiterentwicklung der Software angegangen werden. Das könnte 'deep thinking' fördern. Wichtig wird sein, die jeweilige Phase und Ebene (Analyse, Design, Weiterentwicklung, Beurteilen der Lösungsidee, Suchen nach Lösungsideen, etc.) in ihrer Stellung und zu Aufgabe verdeutlichen. Informatische Notationen wie UML und entsprechende Softwareentwicklungsumgebungen sind dabei als Werkzeuge angesprochen. Durch die Arbeit in Gruppen und das Abwägen verschiedener Lösungsideen werden nicht nur vernetzte Denkfähigkeiten, das Betrachten des Problems aus verschiedenen Sichtweisen, sondern auch verschiedene Beziehungen zwischen Kunde und Auftraggeber und darin angelegt zwischen Technik und Gesellschaft erfahrbar.
Wäre das ein ganz neuer Informatikunterricht? Nein, eigentlich nicht. Die Ziele des anwendungsorientierten Ansatzes sind auch vor dem Hintergrund der Objektorientierung nicht gegenstandslos geworden. Vielmehr handelt es sich um eine Akzentverschiebung, die von Impulsen aus der Objektorientierung profitieren kann, aber nicht von diesen Impulsen abhängt.

Wäre dem so, dann gründete die Argumentation allein auf der Ebene der Ergebnisse der Wissenschaft Informatik, nicht jedoch auf der Ebene informatischer Forschungsfragen. Das Problem, wie man die im gegebenen Kontext richtige Software konstruiert (Ambler), wie man zwischen formaler Spezifikation und unformalen Anforderungen Sinn zuweisen kann (Schefe), bzw. wie man den Brückenschlag zwischen den beiden Welten hinbekommt (Keil-Slawik), ist die hier zentrale informatische Fragestellung.
Die Objektorientierung hat für diese Fragestellung keine Patentlösung gefunden, doch kann sie Impulse für Unterrichtsmethoden geben, um diese Frage im Rahmen des Informatikunterrichts gewinnbringend ansprechen zu können. Die "Dekonstruktion von Informatiksystemen als Unterrichtsmethode" (Hampel/Magenheim /Schulte, 1999) macht einen Schritt in diese Richtung.

6. Literatur

Ambler, Scott W.: The Object Primer. The Application Developer's Guide to Object-Orientation. Sigs Books, 1995.

Arlt, Wolfgang (Hrsg.): Informatik als Schulfach. Didaktische Handreichungen für das Schulfach Informatik. Oldenbourg Verlag München Wien 1981.

Balzert, Helmut: Informatik 1. Vom Problem zum Programm. Hueber-Holzmann, 1987. 2. Aufl. (1. Aufl. 1976)

Baumann, Rüdeger: Didaktik der Informatik. Klett, 1996. (Zweite, vollständig neu bearbeitete Auflage)

Beck, K., Cunningham, W.: A Laboratory For Teaching Object-Oriented Thinking. SIGPLAN Notices, Volume 24, Number 10, October 1989 (http://c2.com/doc/oopsla89/paper.html, 9.5.01)

Chandler, Daniel: Technological or Media Determinism. 1995. (http://www.aber.ac.uk/media/Documents/tecdet/tecdet.html, 9.5.01)

Deiters, Wolfgang: Prozeßmodelle als Grundlage für ein systematisches Management von Geschäftsprozessen. In: Informatik Forschung und Entwicklung 12, 1997. S. 52-60. (http://link.springer.de/link/service/journals/00450/papers/7012002/70120052.pdf, 9.5.01)

Eberle, Franz: Didaktik der Informatik bzw. einer informations- und kommunikationstechnologischen Bildung auf der Sekundarstufe II. Aarau: Verlag für Berufspädagogik Sauerländer, 1996.

Flowers, Jim: Problem Solving in Technology Education: A Taoist Perspective. In: Journal of Technology Education, Vol 10, Nr.1 1998.
(http://scholar.lib.vt.edu/ejournals/JTE/v10n1/flowers.html, 9.5.01)

Forneck, Hermann J.: Bildung im informationstechnischen Zeitalter. Verlag Sauerländer, 1992.

Gamma, E., Helm, R., Johnson, R., Vilissides, J.: Entwurfsmuster. Elemente wiederverwendbarer Software. Addison-Wesley, 1996.

Hampel, T., Magenheim, J., Schulte, C.: Dekonstruktion von Informatiksystemen als Unterrichtsmethode. In: Schwill, A. (Hrsg.): Informatik und Schule. Springer, 1999. S.149-164.

Hubwieser, Peter: Modellieren in der Schulinformatik. In: Log In 19, Nr.1, 1999. S. 24-29.

Johnson, Scott D.: A Framework for Technology Education Curricula Which Emphasizes Intellectual Processes. In: Journal of Technology Education, 1992 Vol.3, Nr.2 (http://scholar.lib.vt.edu/ejournals/JTE/v3n2/pdf/johnson.pdf, 9.5.01)

Keil-Slawik, Reinhard: Zwischen Vision und Alltagspraxis: Anmerkungen zur Konstruktion und Nutzung typographischer Maschinen. In: G.G. Voß, W. Holly und K. Boehnke (Hg.): Neue Medien im Alltag: Begriffsbestimmungen eines interdisziplinären Forschungsfeldes. Opladen: Leske & Budrich, 2000. S. 199-220.

Klafki, Wolfgang: Grundzüge eines neuen Allgemeinbildungskonzepts. Im Zentrum: Epochaltypische Schlüsselprobleme. S. 43 - 81. In: Klafki, Wolfgang: Neue Studien zur Bildungstheorie und Didaktik. Weinheim, Beltz. 5. Auflage 1996 (1. Auflage 1985).

Koerber, B., Sack, L., Schulz-Zander, R.: Prinzipien des Informatikunterrichts S. 28-35. In: Arlt.

Krasemann, Hartmut: Welche Ausbildung brauchen Informatiker? In: Informatik-Spektrum 20 (1997) 6, 328-334 (http://link.springer.de/link/service/journals/00287/papers/7020006/70200328.pdf, 9.5.01)

Lehman, M. M.: Programs, Life Cycles, and Laws of Software Evolution. In: Proceedings of the IEEE, Vol. 68, No.9, September 1980.

Lewis, T., Petrina, S., Hill, A.: Problem Posing - Adding a creative increment to technological Problem solving. In: Journal of Industrial Technology Education, Vol 36, Nr.1 1998.
(http://scholar.lib.vt.edu/ejournals/JITE/v36n1/lewis.html, 9.5.01)

Möller, Dirk: Förderung vernetzten Denkens im Unterricht. Grundlagen und Umsetzung am Beispiel der Leittextmethode. Lit Verlag, Münster 1999

Noack, J.,Schienmann, B.: Objektorientierte Vorgehensmodelle im Vergleich. Informatik-Spektrum 22, 1999, S. 166-180. (http://link.springer.de/link/service/journals/00287/papers/9022003/90220166.pdf, 9.5.01)

Pannabecker, J. R.: Technological impacts and determinism in technology education: Alternate metaphors from social reconstructivism. Journal of Technology Education, 1991,Vol 3, Nr.1
http://scholar.lib.vt.edu/ejournals/JTE/v3n1/pdf/pannabecker.pdf, 9.5.01)

Quibeldey-Cirkel, Klaus: Das Objekt-Paradigma in der Informatik. Teubner, Stuttgart, 1994

Quibeldey-Cirkel, Klaus: Entwurfsmuster: Design Patterns in der objektorientierten Softwaretechnik. Heidelberg u. a.: Springer-Verlag 1999.

Quibeldey-Cirkel, Klaus: Quo vadis Informatik? Aspekte einer objektorientierten Entwurfslehre. In: Objekt-Spektrum 2 (1995), Heft 1. S. 30-36. (http://www.ti.et-inf.uni-siegen.de/staff/Quibeldey/Schriften/Quo-Vadis.pdf, 9.5.01)

Riedel, Dieter : Ansätze einer Didaktik des Informatikunterrichts. S.36-41. In: Arlt.

Schefe, Peter: Softwaretechnik und Erkenntnistheorie. In: Informatik Spektrum 22, 1999. S. 122-135 (http://link.springer.de/link/service/journals/00287/papers/9022002/90220122.pdf, 9.5.01)

Schubert, Sigrid: Fachdidaktische Fragen der Schulinformatik und (un)mögliche Antworten. In: Gorny, Peter (Hrsg.): Informatik und Schule 1991. Springer-Verlag 1991. S.27-33.

Wedekind, H., Görz, G., Kötter, R., Inhetveen, R.: Modellierung, Simulation,Visualisierung. In: Informatik-Spektrum 21, 1998. S.: 265-272.
(http://link.springer.de/link/service/journals/00287/papers/8021005/80210265.pdf, 9.5.01)