Handout zur informatik-didaktischen Fortbildung von Lehrenden

Dr. Nicole Weicker
Universität Stuttgart
weicker@informatik.uni-stuttgart.de

15. April 2005

Spielregeln für diese Fortbildung

Vertraulichkeit Bringen Sie sich ein!

Offenheit Sagen Sie, wie es ist, wie sie es sehen!

Fairness Seien Sie kritisch und geben Sie Anmerkungen!

Teamorientierung Geben Sie sich innerhalb des Teams Unterstützung!

Initiative Tun und sagen Sie, was Sie wollen! (zu verstehen als aktives Einbringen)

Bereitschaft Seinen Sie bereit, neue Wege zu gehen!

Störungen haben Vorrang Sitzen Sie nichts aus!

Transfer Halten Sie die Ergebnisse in persönlicher Vereinbarung fest!

1 Lernziele

1.1 Allgemeines

Definition Ein Lernziel ist eine sprachliche Formulierung, die beschreibt, welche Lernergebnisse und welches Verhalten innerhalb einer festgelegten Zeitspanne erwartet werden bzw. erreicht werden sollen.

Abgrenzung: ein Lernziel ist nicht identisch mit dem Lernergebnis, vielmehr handelt es sich um ein vorgestelltes Lernergebnis.

Warum Lernziele explizit formulieren? Die Formulierung von Lernzielen bedeutet für die Lehrenden eine bewußte Vorbereitung auf den Unterricht. Lernziele helfen bei der Auswahl, der in Frage kommenden Inhalte, bei der Reduzierung der ausgewählten Inhalte auf das Wesentliche und bei der Darstellung der Inhalte für den Unterricht.

Neben dem Nutzen für die Lehrenden haben Lernziele auch Vorteile für die Schüler und Schülerinnen. Lernziele machen Lernen auf vielfältige Art möglich und planbar. Wichtig ist dabei, dass die Lernziele so *kleingearbeitet* werden, dass sie für die Praxis nützlich sind.

Formulierung von Lernzielen Lernziele sollten *eindeutig* und klar formuliert werden. Wichtig ist auch, dass sie erreichbar, also *realistisch* sind. Damit sowohl Lehrende wie auch Lernende einen Nutzen aus den Lernzielen ziehen können, ist es hilfreich, wenn die Lernziele *schriftlich* und im Präsens formuliert sind. Im Lernziel sollte enthalten sein, wie es *gemessen* werden kann, sprich, woran man erkennen kann, dass das Ziel erreicht ist. Lernziele sollten positiv formulieren, was erreicht werden soll.

1.2 Lernebenen

Allgemein werden drei Bereiche unterschieden, in denen Lernen unterschiedlich funktioniert und deren Lernziele unterschiedlich beschreiben werden. Diese Bereiche sind:

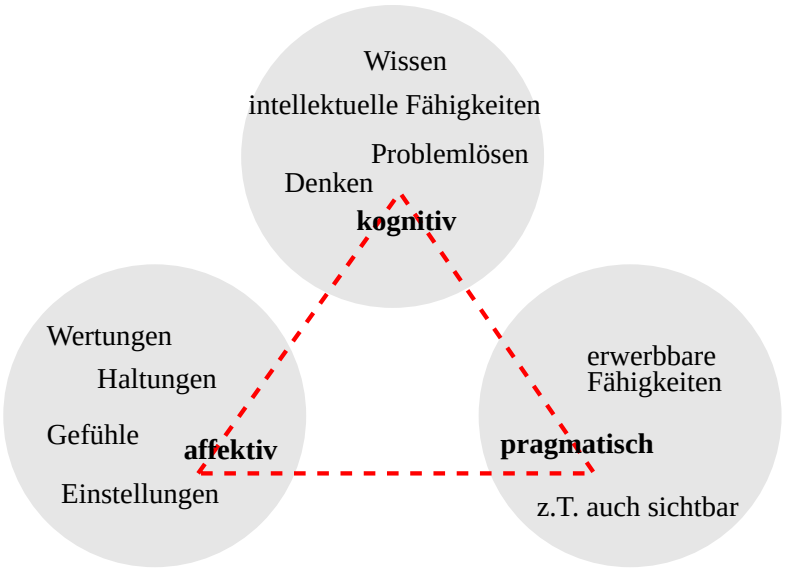

Abbildung 1: Lernbereiche

kognitiv-aktiv Der kognitiv-aktive Lernbereich lässt sich umschreiben als den Bereich, in dem es um die Erhellung des eigenen Daseins geht. Als kognitives Lernen bezeichnet man das Lernen im Bereich von Denken, Wissen, Problemlösung und intellektuellen Fähigkeiten. Die einzelnen Stufen der Taxonomie nach Bloom et al. (1974), in denen das kognitive Lernen erfolgt, sind: Kenntnisse, Verständnis, Anwendung, Transfer, Beurteilung (vgl. Abb. 2).

affektiv-pathetisch Der affektiv-pathische Lernbereich beschäftigt sich mit der Erfüllung des eigenen Daseins. Als affektives Lernen bezeichnet man das Lernen im Bereich der Emotionen, Wertungen, der Einstellungen und Haltungen. Die Taxonomie des affektiven Lernens sieht so: Aufmerksamwerden/Beachten, Reagieren, Werten, Organisation, Wertstruktur (vgl. Abb. 3).

pragmatisch-dynamisch Der pragmatisch-dynamische Lernbereich befasst sich mit der Bewältigung des eigenen Daseins. Pragmatisches Lernen ist das Lernen im Bereich der erwerbbaren Fertigkeiten, die zum Teil auch sichtbar sind. Die Taxonomie des pragmatischen Lernens ist: Imitation, Manipulation, Präzision, Handlungsgliederung, Naturalisierung (vgl. Abb. 4).

Beispiel zum kognitives Lernen

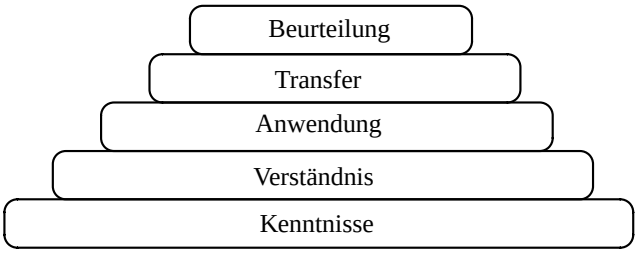

Abbildung 2: Taxonomie des kognitigen Lernens

Eine Schülergruppe möchte als Projekt ein solarbetriebenes Auto entwerfen und bauen. Ein Schüler schlägt vor, das zur Leistungsverbesserung der neue Kompensator XY von Z mit automatischer Energiespeicherung verwendet werden könnte. Die Schüler müssen ein Vorwissen zur Elektronik besitzen (Kenntnisse), die Gedanken des Erfinders Z nachvollziehen können und seine Verwendung des Kompensators soweit verstehen (Verständnis), dass sie an kleinen Beispielen, seinen Nutzen durchrechnen können (einfache Anwendung). Anschließend sollten sie die Leistung ihres Autos mit und ohne den neuen Kompensatores berechnen (Transfer). Erst danach werden sie in der Lage sein zu entscheiden, ob sie dieses Bauteil verwenden wollen oder nicht (Beurteilung).

Beispiel zum affektives Lernen

Abbildung 3: Taxonomie des affektiven Lernens

Ein Lehrer sagt einem Schüler, dass er sich in den Übungsgruppen kooperativer verhalten soll (Aufnahme). Der Schüler reagiert mit der Gegenfrage, auf welche Situation sich der Lehrer in seiner Aussage bezieht (Reaktion). Später stellt der Schüler für sich selbst fest, das der Lehrer mit seiner Aussage Recht hat (Werten) und nimmt sich vor, sein Verhalten zu ändern (Aufnahme des neuen Wertes in der eigenen Wertehierachie). Der Lehrer stellt in den nächsten Stunden fest, das sich das Verhalten des oben erwähnten Schülers tatsächlich verändert hat (Charakterisierung des Verhaltens durch den Wert)

Beispiel zum pragmatisches Lernen

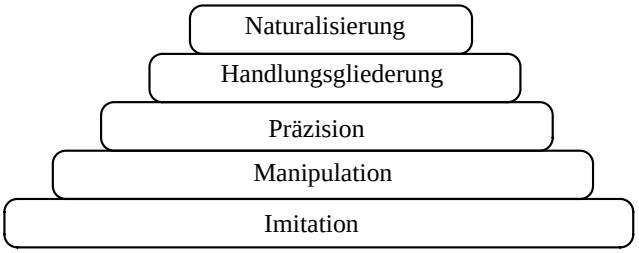

Abbildung 4: Taxonomie des pragmatischen Lernens

Ein Mensch möchte Tischler werden. Also begibt er sich zu einem Tischler, um das Tischlern von ihm zu lernen. Er schaut sich an, was der Tischler macht, lässt sich von ihm zeigen, wie man z.B. den Hobel ansetzt, um die Tischfläche zu glätten. Daraufhin nimmt er sich ein Stück Holz und probiert das erworbene Wissen aus (Imitation). Aber er hat sich nicht alle Feinheiten gemerkt und macht so kleinere Fehler oder ändert etwas anderes unabsichtlich (Manipulation). Dadurch sammelt er Erfahrung im Tischlergewerbe und versucht sein Tischlern mithilfe dieser Erfahrung zu verbessern. Er verändert nun die gelernte Strategie, indem er z.B. den Hobel anders hält oder anders benutzt (Präzision). Als nächstes soll er eine Kommode herstellen. Er benutzt seine Tischerfahrungen, um sein Vorgehen bei der Herstellung eines anderen Möbelstück wie einer Kommode möglichst gut zu planen und später auch umzusetzen

(Handlungsgliederung). Nachdem er seinen 250ten Tisch fertiggestellt hat, braucht er nicht mehr bei jedem Tisch nachzudenken, was er als nächstes zu tun hat. Seine Arbeitsvorgänge haben sich soweit automatisiert, dass er sich nebenher mit seinen Kollegen über das Wochenende unterhalten kann, ohne dadurch langsamer oder unsauberer zu arbeiten (Naturalisieren).

1.3 Schlüsselkompetenzen

Schlüsselkompetenzen oder auch Schlüsselqualifikationen sind „erwerbbare Fähigkeiten, Einstellungen und Wissenselemente, die bei der Lösung von Problemen und beim Erwerb neuer Kompetenzen in möglichst vielen Inhaltsbereichen von Nutzen sind, so dass eine Handlungsfähigkeit entsteht, die es ermöglicht, sowohl individuellen als auch gesellschaftlichen Anforderungen gerecht zu werden.“ (Orth, 1999).

Im schulischen Informatikunterricht sollten vor allem das Wissen, die Fähigkeiten und Fertigkeiten sowie die Haltungen und Einstellungen vermittelt werden, die einerseits notwendig sind, um in diesem Fach erfolgreich arbeiten zu können und andererseits auch unabhängig von der Informatik von Nutzen sein können. Für die Entscheidung, welche Schlüsselkompetenzen für die berufliche Handlungsfähigkeit speziell von Informatikern wichtig sind, ist es hilfreich, sich die Charakteristika der Informatik zu verdeutlichen.

2 Charakteristika der Informatik

Das Bild der Informatik ist vielfältig und bunt. Dies zeigt sich unter anderem darin, dass es bisher für diese noch relativ junge Wissenschaft keine einheitliche Definition gibt. Zwei seien hier beispielhaft aufgeführt:

- Definition aus dem Duden Informatik Claus and Schwill (2001):
  Informatik ist die Wissenschaft der systematischen Verarbeitung und Speicherung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von Computern.

- Definition aus dem Studien- und Forschungsführer Informatik Brauer and Münch (1996):
  Informatik ist die (Ingenieur-)Wissenschaft von der theoretischen Analyse und Konzeption, der organisatorischen und technischen Gestaltung sowie der konkreten Realisierung von (komplexen) Systemen aus miteinander und mit ihrer Umwelt kommunizierenden (in gewissem Maß intelligenten und autonomen) Agenten oder Akteuren, die als Unterstützungssysteme für den Menschen in unsere Zivilisation eingebettet werden müssen – mit Agenten/Akteuren sind Software-Module, Maschinen oder roboterartige Geräte gemeint.

Eine wesentliche Übereinstimmung, die sich bei nahezu allen Definitionsansätzen für Informatik findet, besteht darin, dass Informatik sich als Wissenschaft mit der Verarbeitung und Speicherung von Informationen befasst.

Eine der Haupteigenschaften der Informatik, die sich daraus ergibt, ist die *Immaterialität ihres Hauptprodukts der Software*. Software ebenso wie Information als solche wird zu ihrer Speicherung und Verarbeitung an Materie oder Energie gebunden, existiert jedoch nicht nur im Zusammenhang mit einem speziellen Träger.

Das eigentlich Spannungsfeld für den Informatiker entsteht daraus, die *Brücke zu schlagen* zwischen konkreten Anwendungen und dem abstrakten, immateriellen Konstrukt einer Software.

Ein weiteres wichtiges Charakteristikum der Informatik besteht darin, dass sie inzwischen mit nahezu allen anderen *Disziplinen zusammenarbeitet bzw. Dienstleistungen erbringt.*

Eine weitere informatik-spezifische Schwierigkeit ergibt sich daraus, dass die allermeisten *Projekte in der Informatik sehr groß* und unüberschaubar sind. Das bedeutet, dass eine Teamarbeit fast immer unumgänglich ist.

Die Hardwaresysteme, die Programmiersprachen und auch die auf dem Markt verfügbaren Tools entwickeln sich rasant. Das bedeutet für den Informatiker, dass sich sein *Handwerkzeug permanent verändert.*

Eine Besonderheit der Informatik stellt ihre direkte *gesellschaftliche Auswirkung* dar. Entwicklungen im Bereich der Informatik betreffen immer wieder nahezu jeden Menschen (z.B. Handy oder Internet).

2.1 Beispiel: Immaterialität von Software

Die Produkte der Informatik bestehen in erster Linie aus Software, die als Werkzeug zur Verarbeitung der immateriellen Größe „Information“ selbst immateriell ist. Aus dieser Eigenschaft der Immaterialität von Software ergeben sich eine Reihe von Folgerungen, die für Informatik charakteristisch sind (vgl. Abb. 5). Diese Folgerungen sind ebenso wie die Liste von Charakteristika der Informatik in Abschnitt 2 subjektiv und erheben damit weder den Anspruch auf eine Korrektheit im Sinne einer Beweisbarkeit noch auf Vollständigkeit. Vielmehr sollen sie ein mögliches Gesamtbild der Informatik aufspannen, das zur Reflexion und Diskussion anregt.

2.1.1 Abstraktion von Software

Aus der Immaterialität der Software ergibt sich direkt, dass Software *abstrakt* und nicht konkret oder fassbar ist. Ebenso kann sie nur *schwer visualisiert* werden. Der Code selbst zeigt einem Betrachter lediglich die Details, ohne ein Verständnis für Zusammenhänge und Strukturen innerhalb der Software aufzuzeigen. Diese müssen ohne weitere Informationen mühsam aus dem häufig umfangreichen Code erarbeitet

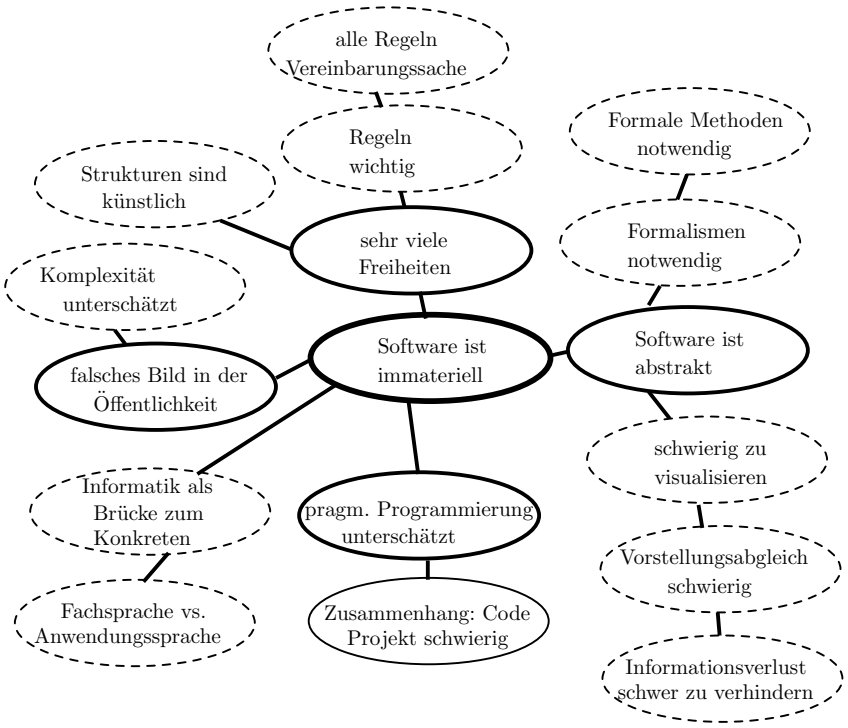

Abbildung 5: Eine zentrale Eigenschaft von Software ist ihre Immaterialität

werden. Systemübersichten wie beispielsweise in UML-Diagrammen zeigen wiederum nur einen bestimmten Ausschnitt der Gesamtzusammenhänge. Damit zeigt sich, dass für Software wie für andere abstrakte Gebilde ein *Abgleich der Vorstellungen* verschiedener beteiligter Personen schwierig ist. Insbesondere ist es schwierig, in der Kommunikation über Software den *Informationsverlust*, der in jeder Kommunikation auftritt (Informationsverlusttreppe), ohne zusätzliche Hilfsmittel zu verringern. Ein notwendiges Hilfsmittel hierfür stellen die *Formalismen* dar, die helfen, Beschreibungen und Vereinbarungen eindeutig zu formulieren. Dabei ist es weniger wichtig, welche Formalismen es sind, als vielmehr, dass es Formalismen gibt, durch die eine Kommunikation eindeutig und unmißverständlich gehalten werden kann. Für die Verwendung von Formalismen stehen Informatikern eine Reihe von *formalen Methoden* zur Modellierung und zur Komplexitätsanalyse wie z.B. endliche Automaten, Turingmaschinen, Petrinetze oder Grammatiken zur Verfügung.

Die beschriebenen Eigenschaften von Software verlangen von einem Informatiker, wie in Abb. 6 dargestellt, eine Reihe von Kompetenzen wie die Fähigkeit zur Abstraktion, eine Formalisierungskompetenz, die Fähigkeit, formale Methoden anwenden zu können, und die Visualisierungskompetenz, um eigentlich nicht Visualisierbares zu mindest in Ausschnitten doch sichtbar gestalten zu können. Andererseits ist es für einen Informatiker notwendig, die Idee der Sprache mit ihren Aspekten der Syntax und Semantik verinnerlicht zu haben. Erst durch ein derartiges Verständnis ist ein Informatiker in der Lage Formalismen und formale Methoden wirklich sinnvoll in

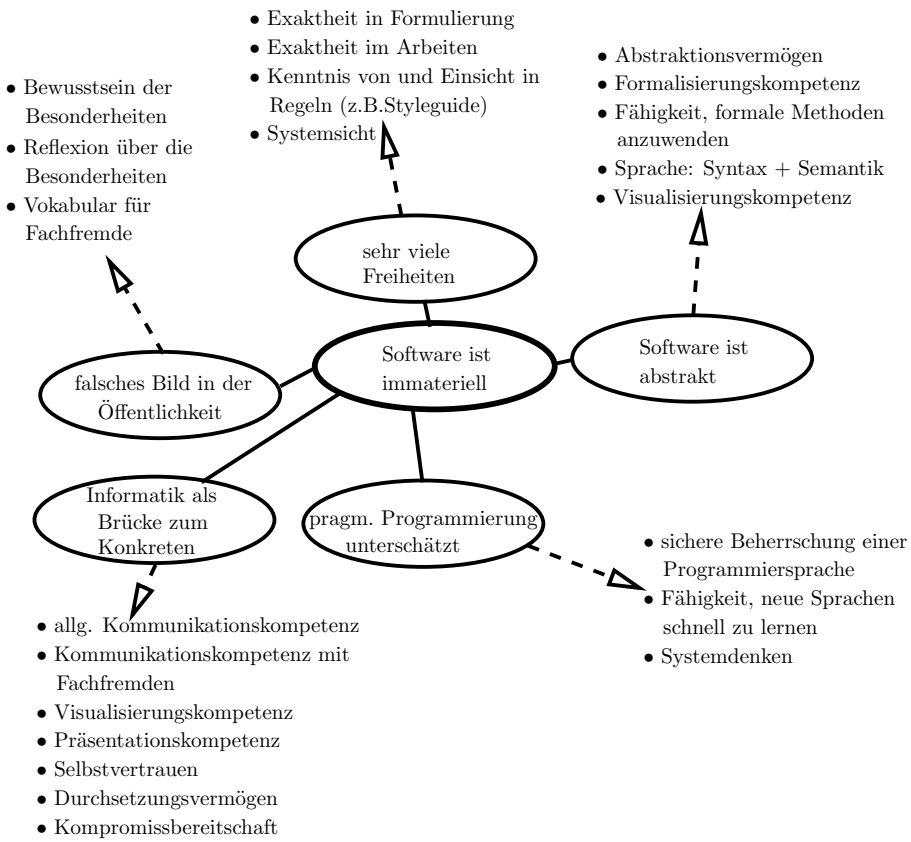

Abbildung 6: Aus der Immaterialität ergibt sich die Notwendigkeit für eine Reihe von fachlichen und überfachlichen Kompetenzen

neuen Kontexten umzusetzen.

2.1.2 Große Freiheiten bei der Erstellung von Software

Eine weitere Folgerung der Immaterialität von Software ist die Tatsache, dass Informatiker eine sehr große Freiheit bei der Erstellung von Software besitzen (vgl. Abb. 5). Software kann auf der Ebene der Maschinensprache (Assembler) ebenso entwickelt werden wie auf einer der modernen Sprachen wie Java, die bereits bestimmte Strukturen vorschreiben. Unstrukturierte Programmierung kann ebenso bzw. z.T. in noch weniger Code zum gleichen Ergebnis führen, wie strukturierte und gut kommentierte Programmierung. Um jedoch mit den entstandenen Softwareteilen umgehen zu können, sie verändern oder in andere Softwareumgebungen einpassen zu können, ist es notwendig, sich auf *Regeln* zu einigen. Dabei handelt es sich jedoch um Regeln, über die *Vereinbarungen* getroffen wurden, und nicht um Gesetze, die immer gültig wären. Eine andere Folge der Tatsache, dass Software keinen Naturgesetzen unterliegt, besteht darin, dass Software auch keine natürliche Lokalität besitzt. Softwareteile, die sich untereinander möglichst nicht beeinflussen sollen, sind durch *künstliche Strukturen* wie z.B. eine Modularisierung oder Hierarchisierung zu trennen.

Die Fähigkeiten, die einen Informatiker in die Lage versetzen, mit diesen Folgerungen der Immaterialität von Software umgehen zu können (vgl. Abb. 6), sind auf der einen Seite exaktes Formulieren und exaktes Arbeiten sowie die Kenntnis um die gängigsten Regel wie beispielsweise Styleguides oder Prozessmodelle des Software Engineering und die Einsicht in die Notwendigkeit diese Regelwerke einzuhalten. Auf der anderen Seite benötigt ein Informatiker die Fähigkeit des Systemdenkens. Damit ist die Eigenschaft gemeint, sowohl selbst bei der Arbeit am Detail die systemischen Zusammenhänge der eigenen Arbeit in einem größeren Projekt im Blick behalten zu können.

2.1.3 Bild der Informatik in der Öffentlichkeit

Die *Vorstellung in der Gesellschaft*, was Informatik ist, wird wesentlich von den Auswirkungen der Informatik auf den Alltag des Einzelnen geprägt (vgl. Abb. 5). Dabei spielen Handys, Computer, Internet, E-Commerce sowie gängige Textverarbeitungs- oder Kalkulationssysteme eine wesentliche Rolle. Die tatsächliche *Komplexität* der Software hinter diesen nach außen sichtbaren Leistungen der Informatik bleibt dem Nichtinformatiker dagegen verborgen. Gerade die Eigenschaft der Immaterialität der Software hat zur Folge, dass Informatik an der sichtbaren und oft simpel erscheinenden Benutzerschnittstelle gemessen wird. Die Auswirkungen dieser Fehleinschätzung sind, am deutlichsten in der überdurchschnittlichen Anzahl an Studienanfängern in der Informatik zu sehen, die innerhalb weniger Semester ihr Studium abbrechen. Doch auch in Betrieben, in denen die Informatik als Dienstleistung benötigt wird, herrscht häufig Unverständnis für die Probleme, denen sich ein Informatiker zu stellen hat. Ein deutliches Kennzeichen hierfür zeigt sich in der oft sehr geringen Berücksichtigung für wichtige Phasen des Softwareentwicklungsprozesses wie z.B. der Anforderungsspezifikations bzw. der Testphase.

Wenn sich ein Informatiker der Besonderheiten der Informatik bewusst ist, kann er seine Bedeutung und den Wert seiner Arbeit anders einschätzen und einordnen (vgl. Abb. 6). Doch erst durch eine tiefere Reflexion über diese Besonderheiten und einem geeigneten Vokabular zur Vermittlung der speziellen Probleme der Informatik auch an einen Nichtinformatiker besteht eine Chance, in Betrieben mehr Geld und Zeit für eine gute Entwicklung und Wartung von Software durchzusetzen.

2.1.4 Informatik als Brücke zum Konkreten

Obwohl Software immateriell ist, wird sie zur Lösung von konkreten Aufgaben verwendet (vgl. Abb. 5). Das Spannungsfeld des Informatikers besteht darin, eine *Brücke zwischen konkreten Anwendungen und dem abstrakten Konstrukt einer Software* zu schlagen. Dabei ist seine Aufgabe, reale Situationen geeignet zu analysieren und in Modellen abzubilden, so dass die Anforderungen des Kundens und zusätzlich die der späteren Anwender der Software in den Modellen geeignet berücksichtigt werden. Ein wesentliches Problem, dem sich der Informatiker in dieser Phase seiner Arbeit zu stellen hat, liegt in der Unterschiedlichkeit der jeweiligen *Fachsprachen*. Informatik hat

ebenso wie jede andere wissenschaftliche Disziplin seine eigene Fachsprache, wobei damit weniger die Programmiersprachen sondern die informatikspezifische Ausdrucksweise in der Kommunikation über Software gemeint ist. Kunde und Anwender haben in aller Regel eine andere Fachsprache, so dass gerade die Anforderungsspezifikation sehr schwierig werden kann.

Der Brückenschlag der Informatik vom Konkreten zum Abstrakten verlangt vom Informatiker neben einer allgemeinen Kommunikationskompetenz die Fähigkeit, sich auch mit Fachfremden über seine Arbeit auszutauschen (vgl. Abb. 6). Für den Informatiker gilt es sich immer wieder in sein Gegenüber einzudenken (Empathie) und entstehenden Mißverständnissen möglichst schnell entgegen zu wirken. Zusätzlich sollte der Informatiker über geeignete Visualisierungs- und Präsentationskompetenzen sowie über ausreichend Selbstvertrauen, Durchsetzungsvermögen und Kompromissbereitschaft verfügen, um seine Lösungsansätze Kunden und Anwendern verdeutlichen zu können.

2.1.5 Wert der pragmatischen Programmierung

Auch für den angehenden Informatiker stellt die Immaterialität der Software ein Problem dar, da es häufig zu genügen scheint, wenn Zusammenhänge begriffen oder Lösungsansätze im Modell gefunden werden (vgl. Abb. 5). Vor die Aufgabe gestellt, einen Tisch anzufertigen, ist jedem offensichtlich, das es nicht genügt, verstanden zu haben, wie der Tisch glatt gehobelt wird. Es ist notwendig, selbst die Technik des Hobelns pragmatisch zu erlernen. Bei der Erstellung von Software dagegen wird gerade der *pragmatische Aspekt der Programmierung* häufig unter- und die eigenen Fähigkeiten in dieser Hinsicht werden entsprechend überschätzt. Die *Zusammenhänge* zwischen der Planung eines Projekts (Analyse, Spezifikation und Entwurf) und der tatsächlichen Umsetzung sind nicht einfach zu überblicken. So sind einerseits aus der Sicht der Planung die Probleme, die sich bei der Realisierung ergeben, häufig kaum abzuschätzen. Andererseits haben die Codierer Mühe, Vorstellungen und Nebenbedingungen, die nicht exakt festgehalten sind, umzusetzen, weil ihnen das Gesamtbild und die Bedeutung der Software im Projekt fehlen.

Konkret sollte im Unterricht der Informatik von Anfang an die Bedeutung der pragmatischen Programmierfähigkeit als ein wichtiger Aspekt der Informatik deutlich in den Vordergrund gestellt werden (vgl. Abb. 6), ohne allerdings sich auf diesen wichtigen Bestandteil der Informatikkenntnisse zu beschränken. Ein Informatiker sollte in der Lage sein, eine Programmiersprache sicher zu beherrschen. Daneben ist es notwendig, dass die entsprechenden Grundkonzepte der Programmierung verstanden sind, so dass der Transfer von dieser Sprache in eine andere leicht und schnell möglich ist. Eine weitere, bereits angesprochene Kompetenz in diesem Zusammenhang, über die ein Informatiker verfügen sollte und die während auch schon in der Schule geschult werden kann, ist die Fähigkeit des Systemdenkens.

2.1.6 Zusammenfassung der Kompetenzen bzgl. der Immaterialität

Die Qualifikationen, die sich durch die Immaterialität von Software ergeben, lassen sich untergliedern in fachliche, methodische, soziale und Selbstkompetenzen.

Die fachlichen Kompetenzen, die einen Informatiker befähigen, mit der Immaterialität von Software umzugehen, sind u.a. das Abstraktionsvermögen, die Formalisierungskompetenz, das Verständnis für das Konzept der Sprache mit der Unterscheidung in Syntax und Semantik sowie die Kenntnis von und Einsicht in Vereinbarungen und Regeln, wobei die beiden ersteren Punkte hauptsächlich in der Theorie der Informatik geschult werden, während letzteres in Veranstaltungen zum Software Engineering vermittelt wird. Die Sprache als fundamentale Idee der Informatik Schwill (1993) zieht sich durch jede Form des Informatikunterrichts.

Die wichtigsten Methoden zur Beherrschung der Immaterialität von Software bilden die Exaktheit in Formulierung und Arbeit sowie die Fähigkeit, formale Methoden anwenden zu können. Ebenso wichtig ist die sichere Beherrschung einer Programmiersprache und die Kenntnis der programmiertechnischen Grundlagen und Konzepte, so dass weitere Programmiersprachen schnell erlernt werden können. Darüberhinaus ist die Fähigkeit, abstrakte Zusammenhänge visuell darstellen zu können, eine weitere wichtige Methode für den Umgang mit Immaterialität. Für diese Methodenkompetenzen kann im Informatikunterricht bereits eine Grundlage gelegt werden. Wesentlich ist dabei die Beschäfitigung mit den Grundlagen der Informatik sowie dem Erlernen einer Programmiersprache. Die Visualsierungskompetenz kann im Rahmen von Referaten gefördert werden.

Soziale Qualifikationen sind vor allem die Kommunikationskompetenz sowohl allgemein als auch speziell im Umgang mit Fachfremden und die scheinbar widersprüchlichen Aspekte Kompromissbereitschaft und Durchsetzungsvermögen. Bis auf die fachliche Kommunikation mit Nichtinformatikern können diese Kompetenzen durch Projektarbeit gefördert werden. Für eine gezielte Förderung des Umgangs mit Fachfremden bieten sich Industriepraktika oder disziplinübergreifende Arbeiten an.

Die wesentlichen Aspekte der Selbstkompetenz, die ein Informatiker im Umgang mit der Immaterialität von Software erwerben sollte, ist ein Bewusstsein für und Reflexion über die Besonderheiten der Informatik sowie Selbstvertrauen und Präsentationskompetenz. Zusätzlich sind auch die Punkte der Systemsicht und des Systemdenkens der Selbstkompetenz zu zu ordnen. Die Reflexionsebene über die Informatik kann ebenso wie die Systemsicht und -denken während des Studiums gefördert werden, in dem zu verschiedenen Zeitabschnitten im Verlauf des Studiums immer wieder Denk- und Diskussionsaufgaben zu den Charakteristika der Informatik angeboten werden. Für die Ausbildung eines systemischen Denkansatzes kann es hilfreich sein, häufig wechselnde Positionen in Projekten vertreten zu müssen. Selbstvertrauen und Präsentationskompetenz können im Rahmen von Projekten und Seminaren geschult werden.

Literatur

Rüdiger Baumann. *Didaktik der Informatik*. Klett, Stuttgart, 1996.

Benjamin S. Bloom, Max D. Engelhart, Edward J. Furst, Walker H. Hill, and David R. Krathwohl. *Taxonomie von Lernzielen im kognitiven Bereich*. Beltz, Weinheim, 1974.

Wilfried Brauer and Siegfried Münch. *Studien- und Forschungsführer Informatik*. Springer Verlag, Berlin, 3 edition, 1996.

Volker Claus and Andreas Schwill. *Duden Informatik*. Dudenverlag, Mannheim, 3 edition, 2001.

Franz Eberle. *Didaktik der Informatik bzw. einer informationstechnologischen und kommunikationstechnologischen Bildung auf der Sekunda*. Sauerländer GmbH Verlag, Aarau, 1996.

Peter Hubwieser. *Didaktik der Informatik: Grundlagen, Konzepte, Beispiele*. Springer, Berlin, 2000.

Helen Orth. *Schlüsselqualifikationen an deutschen Hochschulen*. Luchterhand, Neuwied, Kriftel, Berlin, 1999. Konzepte, Standpunkte und Perspektiven.

Sigrid Schubert and Andreas Schwill. *Didaktik der Informatik*. Spektrum Akademischer Verlag, Heidelberg, Berlin, 2004.

Andreas Schwill. Fundamentale Ideen der Informatik. *Zentralblatt für Didaktik der Mathematik*, 1:20–31, 1993.