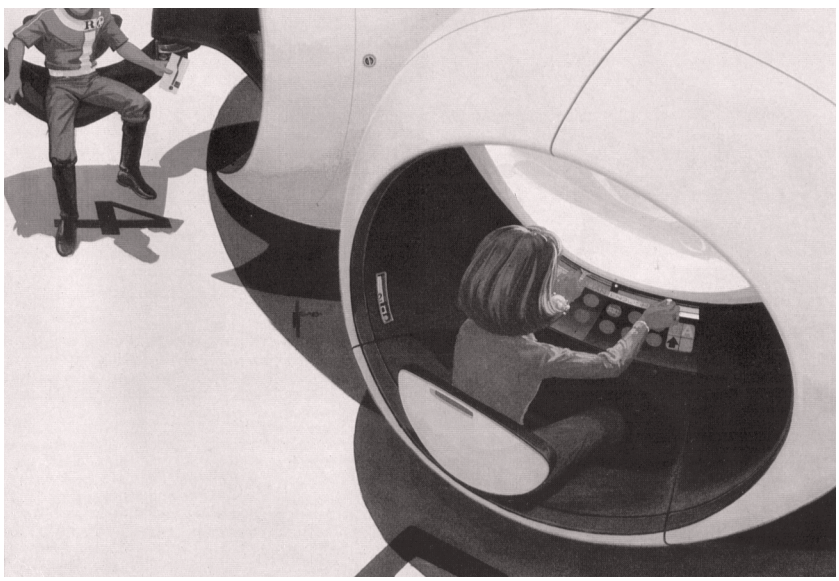
Eine Mustersprache für das Design von Autorensystemen

Eine Anwendung des Open-Source–Entwicklungsmodells auf Entwurf und Herstellung von Lernsoftware

Dissertation
zur Erlangung des akademischen
Grades eines Doktors der Erziehungswissenschaft (Dr. phil.)
im Fachbereich Erziehungswissenschaft/Humanwissenschaften
der Universität Gesamthochschule Kassel
vorgelegt von Thomas Fischer
Kassel, den 18. Juni 2001



Für Candy,
meine Eltern und Großeltern

◊ ◊ ◊

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Vorwort

Zu dieser Arbeit

Diese Arbeit diskutiert Probleme und Lösungen, die bei der Herstellung von Autorensystemen für den Unterricht anfallen, also bei der Entwicklung von Werkzeugen, mit denen Lehrer wiederum digitale Lernmittel herstellen können. Ich könnte diesem Gegenstand kaum gerecht werden, wenn ich mich bei seiner Untersuchung auf die Perspektive und die Möglichkeiten einer einzigen Disziplin beschränken würde, denn Autorensystem–Design für unterrichtliche Zwecke ist eine interdisziplinäre Herausforderung. Es erfordert die verantwortungsvolle Anwendung von Mitteln didaktischer, planerischer und technischer Natur, um nur einige wenige Beispiele zu nennen. Dennoch verfolgt diese Arbeit das ganz und gar erziehungswissenschaftliche Ziel der Sicherung und Steigerung von *Unterrichtsqualität* im Kontext zunehmender Vernetzung und Digitalisierung unserer Umwelten.

Es ist nicht das Ziel dieser Arbeit, medienpädagogische Fragen zu beantworten, etwa *ob* und *was* wir überhaupt mit elektronischen Medien unterrichten sollten, oder welches Medium effizienter ist als andere. Zu diesem Themengebiet existiert bereits ein umfassender Apparat an Studien, Evaluationen und Empfehlungen[1]. Angesichts der fortgeschrittenen Verbreitung von Informationstechnologien in den uns umgebenden Lebens– und Arbeitsräumen avanciert die Anwendung von Computer und Internet zu einer grundlegenden Kultur-

[1] Eine jüngere deutschsprachige Publikation dieser Art mit vielen weiterführenden Verweisen auf Studien etwa der vergleichenden Medienwirkungsforschung sowie auf gedruckte und digitale Unterrichtsmaterialien ist: Astleitner, Hermann et al.: [2]

technik. Die Notwendigkeit von computervermitteltem Unterricht heute und in Zukunft ist eine schlichte Tatsache. Der zum gegenwärtigen Zeitpunkt diskutierte Bedarf an Qualifikation auf diesem Gebiet belegt dies.

Hieraus erwächst eine große Verantwortung für unsere Bildungssysteme, die uns umfassend und rechtzeitig mit den erforderlichen Qualifikationen versorgen müssen. Da diese Versorgung den didaktischen Einsatz von Informationstechnologien selbstverständlich einschließt, stellt sich aus praktischer Sicht viel dringlicher die Frage, *wie* wir mit dem Computer unterrichten sollen: *Wie* sollen Lehrende ihre Unterrichtsentwürfe in digitale Lehrmittel und Interaktionsprosesse umsetzen? *Wie* sollen sie die Organisation von Unterrichtsvorbereitung, Lernergruppen, Materialien, neuen Interaktionsformen und Leistungsbewertungen bewerkstelligen? *Wie* sollen sie netzwerkvermittelten und traditionellen Unterricht integrieren? Zwar sprechen wir seit der Mitte des 20. Jahrhunderts voller positiver Erwartungen von umfassender Computer–Anwendung im Unterricht, die Praxis der Lehre ist von diesen Ambitionen bis heute jedoch nahezu unberührt geblieben.

Zur Erlangung des Staatsexamens für das Lehramt an Gymnasien habe ich 1996 eine Arbeit mit dem Titel „Hypertext und Hypertextmedien — Strukturen und didaktisches Potential“ verfaßt[2]. Darin habe ich unter anderem verschiedene verfügbare Hypertextsysteme und ihre Autoren–Schnittstellen unter didaktischen Gesichtspunkten analysiert und bin zu dem Schluß gekommen, daß diese Systeme nicht oder nur in geringem Maß von ihren Nutzern an konkrete Erfordernisse ihres Unterrichts angepaßt werden können. Dieses grundsätzliche Problem wurde seither nicht gelöst und bis heute sind sowohl der alltägliche didaktische Einsatz von Autorensystemen als auch die Bereitstellung digitaler Lernmittel weltweit seltene Ausnahmen. Vor diesem Hintergund versteht sich dieser Text als konkreter, konstruktiver und praxisorientierter Entwurfsansatz für eine Strategie, die über die Grenzen einzelner Unterrichtssituationen oder Fachdidaktiken hinaus nach anwendbaren Lösungen sucht.

Die vorliegende Arbeit hat drei aufeinander folgende Schwerpunkte: Zunächst

[2] Fischer, Thomas: [13]

fasse ich den gegenwärtigen Stand der Autorensystemtheorie zusammen und diskutiere aus einem interdisziplinären Blickwinkel die bislang unbefriedigende Situation der Versorgung Lehrender mit Werkzeugen für die Herstellung und Organisation digitaler Lernmittel. In diesem Überblick demonstriere ich unter anderem mit designtheoretischen Mitteln die zutiefst widersprüchlichen Charaktere des Problems Unterricht und des Werkzeugs Computer.

Anschließend entwerfe ich ein Modell für die Entwicklung von Autorensystemen, wobei die Analyse bestehender Probleme und die diskutierten Konzepte und Ideen eine wichtigere Rolle spielen als technische Details. Zentrale Aspekte des präsentierten Verfahrens sind die Anwendung des *Open Source*–Entwicklungsmodells[3] und des *Pattern Language*–Ansatzes[4] auf didaktische Probleme. Für die Herstellung frei verfügbarer Werkzeuge sowie von Online–Unterricht bieten *Offene Quelltexte*, *peer review*, deren direkte Verwurzelung in der praktischen Anwendung und auch die Kommunikation über das Internet große Chancen.

Die Entwurfswissenschaft hat bereits in den 70er Jahren mit sogenannten *Mustersprachen* ein Entwicklungsmodell vorgeschlagen, das mit dem Open Source–Modell gut vereinbar erscheint. Mit diesem Verfahren können erfolgreiche (und erfolglose) Software–Lösungen als solche dokumentiert, kommuniziert, ggf. kommentiert, wiederverwendet und weiterentwickelt werden, wodurch die ständige Neu–Entwicklung von Entwurfselementen vermieden werden kann. Die gemeinsame Anwendung dieser beiden Ansätze stellt einen Versuch dar, ein soziales und kulturelles Problem der Didaktik, das bislang mit einer Vielzahl unterschiedlicher technischer Mittel nicht gelöst werden konnte, seinerseits mit sozialen und kulturellen Mitteln anzugehen.

Im letzten Teil führe ich technische Beispiele als Startvokabular einer Mustersprache für die Entwicklung von Autorensystemen auf. Diese sollen trotz ihrer vollen Funktionsfähigkeit hier vor allem als prototypisches Demonstrationsmaterial dienen. Sie basieren zum größten Teil auf Konzepten des Online–Autorensystems SeSAMe, das ich während der Forschung für diese Arbeit an

[3] O'Reilly: [42]

[4] Christoper Alexander: [1]

der Universität Gesamthochschule Kassel entwickelt habe sowie auf der Programmiersprache SeL, die ebenfalls im Rahmen dieser Entwicklung entstand.

Ziel dieses Textes ist seine Diskussion und seine praktische *Anwendung*. Aus diesem Grund richtet er sich, durch viele praxisbezogene Anlehnungen und Beispiele illustriert, vornehmlich an programmierende und nicht–programmierende Lehrpraktiker, Lehrplaner, Lehramts–Studenten, Entwickler freier Software und andere Entwerfer. Das Format des Startvokabulars im hinteren Teil gestattet freie, nicht–lineare Lesestrategien und kann seine Intention und Struktur daher sehr schnell vermitteln. Um auch zuvor im theoretischen Teil das Querlesen und die rasche Aufnahme zentraler Aussagen zu erleichtern, finden sich wichtige Gedanken und Ideen in Textboxen, die dem eiligen Leser, auch ohne den jeweiligen Kontext rezipiert, das Anliegen dieser Arbeit näher bringen und ihre Diskussion ermöglichen. Um das interdisziplinäre Verständnis der verwendeten Fach–Terminologien zu erleichtern, habe ich im Anhang ein Glossar mit den wichtigsten verwendeten Fachbegriffen angelegt.

Ich nenne Berufs– und Tätigkeitsbezeichnungen an vielen Stellen nur in der männlichen Form mit dem Ziel, knapper und damit leichter verständlich zu schreiben. Selbstverständlich schließen diese Stellen immer auch die jeweilige weibliche Form ein.

Sämtliche aufgeführten Online–Referenzen sind — sofern nicht anders vermerkt) gültig am Abgabe–Datum (5. Januar 2001).

Dank

Von den vielen Menschen, die mich unterrichtet, mir geholfen und bei der Arbeit an diesem Buch unterstützt haben, nenne ich hier nur einen kleinen Teil namentlich. Ich möchte mich bei meiner Familie und allen, mit denen ich in den vergangenen Jahren gearbeitet habe, herzlich bedanken.

- **Prof. Wilhelm Sanke**, FB01/GhK
- **Prof. Hans Dehlinger**, FB12/GhK
- **Dr. Herbert A. Meyer**, FB03/GhK
- **Michael Hildebrandt, BSc**, FB03/GhK und FB Psychologie/Uni Göttingen
- **Axel Krägelius**, Nordhessen.net
- **Dr. Bernt Armbruster**, Zentralverwaltung/GhK
- **Jens Brömer**, Zentralverwaltung/GhK
- **Dr. Wolfgang Adamzcak**, Zentralverwaltung/GhK
- **Andreas Matthias**, Hochschulrechenzentrum/GhK
- **Christa Roth**, Studienberatung/GhK
- **Prof. Ellis Horowitz**, University of Southern California USA
- **Prof. Mark Burry**, School of Architecture and Building/Deakin University Australien
- **Prof. John Frazer**, School of Design/Polytechnic University Hong Kong
- **Assistant Prof. Cristiano Ceccato**, School of Design/Synapse Lab, PolyU Hong Kong
- **Assistant Prof. Philip Wong**, School of Design / Synapse Lab, PolyU Hong Kong
- **Associate Prof. Lorne Falk**, School of Design, PolyU Hong Kong
- **Associate Prof. Robert Woodbury**, School of Architecture, Landscape Architecture, And Urban Design / Adelaide University Australien
- **Prof. Urs Hirschberg**, Eidgenössische Technische Hochschule Zürich / Harvard University

Zum Titelbild

Das Lernsystem für Kinder (*Child's Learning Unit With VDU by Syd Mead* auf der Titelseite) wurde von Syd Mead für eine Design–Ausstellung der Firma Philips C.I.D.C. auf der ICOGRADA-Konferenz 1973 in Wien gestaltet. Es ist eine Anordnung aus individuellen Terminals, in denen vorproduzierte Kassetten mit Lernmaterialien zum Abruf bereitstehen.

> The child's learning unit illustrated [on the front page] was designed to operate automatically. Mead describes the concept by noting that "the seat swings out for entrance and exit. When it closes, the programming starts and several options are offered. The unit can be used to review previous study material; the student can insert a display presentation cartridge, contact the central library for references or view animated sequences of scientific or historical events. In fact, the whole principle is an idea generator. Up to a certain point, visual information helps one to learn, but once the intellectual alertness is stimulated there is a preference to invent imagery. Then comes a gradual decline in the need for new references, if the invention is done well."[5]

Die Illustration ist die Vision eines Designers von technologievermittelter Lehre. Daher ist sie eine passende Metapher für den Inhalt dieser Arbeit.

[5] The title image was created by Syd Mead PHILIPS C.I.D.C. Mit freundlicher Genehmigung von Oblagon Inc. Reproduziert aus: Mead, Syd: [35], S. 127

2. Einführung (Problemanalyse)

Mein Denken über Unterricht und didaktisches Handeln ist stark geprägt von meiner eigenen Unterrichtspraxis im Fach Design. Diese Disziplin erlebt (vielleicht prototypisch für viele andere, die an Schulen und Universitaten weltweit unterrichtet werden) rasche Veränderungen unter dem Einfluß neuer Technologien. Da Produkte, Dienste und Informationen heute nicht nur verfügbarer, sondern darüber hinaus auch sehr leicht und billig reproduzierbar und kopierbar sind, verlieren sie aus Sicht des Designers mehr und mehr den Charakter einzigartiger und daher wertvoller Erzeugnisse. Die zur Herstellung des Produkts angewendete Methode tritt infolgedessen in den Vordergrund und stellt nunmehr die zentrale Ressource des Designers dar: *Die Methode erhält Produkt–Charakter.*

Diese Verschiebung des Fokus vom unmittelbar Gegenständlichen auf eine handlungs– und reflexionsintensivere Meta–Ebene im Fach Design steht stellvertretend für ähnliche Wandlungen, die wir auch in anderen Bereichen (wie etwa der Didaktik) spüren. Insbesondere vor dem Hintergrund der sogenannten *Informationsgesellschaft* und der *Wissens–Explosion* wird zum Beispiel die Wichtigkeit der Vermittlung konkreten Wissens oft der Vermittlung von Methoden zur selbstgeleiteten Wissens–Aneignung untergeordnet.

Die Wirkung, die die Anwendung dieser Idee auf Lernende besitzt, wurde bereits vor langer Zeit erkannt und in didaktischem Planen und Handeln berücksichtigt. Davon zeugen geflügelte Worte wie z.B. „Wissen heißt wissen, wo's steht!“ und Handlungsorientierte Pädagogik, Konstruktive Pädagogik und Reformpädagogik weisen deutliche Einflüsse dieses Denkens auf. Die sogenannte

Montessori–Methode wird zum Beispiel oft mit dem folgenden Satz charakterisiert:

> „Give a man a fish and you feed him for a day. Teach him how to fish, and you feed him for a lifetime.“[1]

Jenseits der Ebene „beplanter“ Lerner hat die dargestellte Verschiebung von Verantwortlichkeiten aber auch Konsequenzen für „Planer“ (Lehrer) und „Meta–Planer“ (traditionell Entwerfer von Curricula und Lehrmethoden). Diese Konsequenzen finden bislang wenig Beachtung, sind aber von essentieller Bedeutung für das Design von Autorensystemen, da (wie ich im Laufe dieser Arbeit darstellen werde) Autorensysteme Werkzeuge für den Unterricht sind und Lehrer sich in ihrer neuen Verantwortung als „Werkzeug–Macher“ wiederfinden.

Wir finden Kaskaden aus *Beplanten*, *Planern*, *Planern für Planung* in vielen Bereichen des Lebens — nicht zuletzt in unseren Bildungssystemen. Die oben dargestellte Verschiebung der jeweiligen Verantwortlichkeiten auf handlungs– und reflexionsintensivere Meta–Ebenen führt auf Ebene der *Beplanten* zu einer Rollenverschiebung von bloßen *Empfängern* oder *Konsumenten* zu eigenverantwortlichen, aktiv handelnden Gestaltern. Lernende sind gemäß dieses Rollenverständnisses aktiv handelnde und reflektierende Entwickler eigener Kompetenzen. *Planer*, in diesem Fall Lehrer, nehmen teilweise die bisherigen Rollen von Meta–Planern wahr und entwerfen durch inhaltliche und methodische Auswahl zunehmend *Umwelten*, die *Beplante* in ihrem Handeln unterstützen. Im Fall von Unterricht bedeutet dies, daß Lehrer weniger die direkte Interaktion mit Lernenden bestreiten (diese erfolgt zunehmend im Dialog zwischen Lerner und Computer sowie im computervermittelten Dialog unter Lernern) und dabei *Wissen vermitteln*, sondern zunehmend *Werkzeuge* und *Umgebungen* aus Materialien und Strukturen herstellen, die sie im Kontext der jeweiligen Unterrichtssitiationen als sinnvoll erachten. Am oberen Ende der Entwurfs–Kaskade entsteht durch diese Verschiebung ein Problem: Klassische Meta–Planer werden

[1] Dieses Zitat findet sich in zahllosen Diskussionen der Montessori–Pädagogik, z.B unter http://sfparenting.com/educationalalternatives/montessori.htm

auf eine völlig neue Ebene verschoben, deren Verantwortlichkeiten auf der Basis traditioneller Rollenverstandnisse nicht beschrieben werden können. Was sind die neuen Vorbilder für die neuen Rollen bisheriger Entwerfer von Curricula und Unterrichtsmethoden, nachdem ihre traditionellen Aufgaben nun teilweise von Lehrenden wahrgenommen werden?

Mein besonderes Interesse gilt hier der Herstellung von Unterrichtswerkzeugen im Kontext computervermittelten Unterrichts. Wie ich weiter unten diskutieren werde, nehmen sich unterrichtliche Meta–Planer bislang nicht der neuen Herausforderung der Unterstützung Lehrender bei der Entwicklung unterrichtlicher Werkzeuge und Umwelten an. Der Bedarf an Methoden für die Herstellung digitaler Unterrichtsmittel wird von traditionellen Software–Herstellern mit einem Angebot aus einer begrenzten Zahl von *one-fits-all*–Pauschallösungen in Form sogenannter *Autorensysteme* gedeckt.

Die Verantwortung des Lehrers als Werkzeugbauer wird aufgrund der mangelhaften Flexibilität dieser Systeme untergraben. Daher tendiert computervermittelter Unterricht heute zu wenig adaptierter Standardmethodik infolge nicht ausreichend abstrahierter und flexibler didaktische Meta–Planung. In der Tat ähneln sich heute insbesondere die pionierhaft anmutenden Online–Kurse höherer Bildungseinrichtungen weltweit mitunter so frappierend, daß es schwerfällt, in ihnen kontextspezifische didaktische Analysen oder individuelle Methodenwahl zu erkennen. Unterricht dieser Art ist nicht nur problematisch, weil er der besonderen Natur jeder einzelnen Unterrichtssituation nicht gerecht wird, sondern auch, weil er infolgedessen nur schwer legitimiert und verantwortet werden kann. Das Resultat ist eine sehr breite Ablehnung der zugrundeliegenden Software–Werkzeuge, die durch mitunter schwierige Benutzbarkeit weiterhin verstärkt wird.

Die in Autorensystemen vordefinierten Methoden sind ebensowenig „Allheilmittel“ für guten Unterricht, wie zuvor andere, auf der Ebene didaktischer Meta–Planung vordefinierter Verfahrensweisen, wie Schallplatte, Radio und Fernsehen. Das folgende, bereits in den 1920er Jahren von einem Lehrer verfaßte Gedicht illustriert anschaulich den beschriebenen Kontrollverlust des Lehrers und sein Interesse, sich nunmehr selbst auf einer didaktischen Meta–Ebene durch

direktes Beeinflussen des Unterrichtswerkzeugs zu betätigen:

> Mr. Edison says
> That the radio will supplant the teacher.
> Already one may learn languages by means
> of Victrola records.
> The moving picture will visualize
> What the radio fails to get across.
> Teachers will be relegated to the backwoods.
> With fire–horses,
> And long–haired women;
> Or, perhaps shown in museums.
> Education will become a matter
> Of pressing the button.
> Perhaps I can get a position at the switchboard.[2]

Wie können Lernumgebungen den neuen Rollen Lernender, und wie können Autorensysteme den neuen Rollen Lehrender gerecht werden? Wie können Lehrer heute nutzerfreundliche „Switchboards“ erhalten, mit denen die die Umgebungen, die sie herstellen, auf nutzerfreundliche Art individuell an ihre Unterrichtssituationen anpassen können? Ich hoffe, zur Beantwortung dieser dringenden aktuellen Fragen mit der in der vorliegenden Arbeit entworfenen Entwicklungs–Strategie beizutragen.

Zu diesem Zweck werde ich in diesem Kapitel zunächst die Strategien darstellen, durch die Autorensysteme primär *Nutzerfreundlichkeit* und damit sekundär *Verbreitung*, *Akzeptanz* und letztlich *Anwendung* finden wollen. Im Anschluß daran werde ich Erklärungen vorschlagen, mit denen begründet werden kann, warum Autorensysteme diese gesetzten Ziele praktisch nicht erreichen und eine *Revolution des Lernens bislang ausgeblieben ist.*

[2] Cuban, Larry: *Teachers and Machines: The Classroom Use of Technology Since 1920.* Teachers College Press, New York, S. 5 Nach Postman, Neil: [46], S. 41-42

2.1. Autorensysteme

Every good work of software starts by scratching a developer's personal itch.[3]

Mit dem Begriff „Autorensystem“ wird eine Reihe verschiedener Software–Werkzeuge bezeichnet, die die Entwicklung von Hypermedia–Materialien[4] unterstützen. Mit diesem zunächst recht allgemeinen Begriff wird nicht explizit unterschieden, ob den mit Autorensystemen hergestellten Produkten eine didaktische oder gar unterrichtliche Intention zugrundeliegt oder nicht. Die Anwendung mit unterrichtlichen Zielsetzungen wird jedoch meist implizit verstanden. Ich konzentriere mich in dieser Arbeit auf die Diskussion von Design und Anwendung von Autorensystemen in unterrichtlichen Kontexten.

Um das unrealistisch positive öffentliche Bild vom *Lernen mit dem Computer* nicht noch mehr zu nähren, weise ich bereits an dieser Stelle darauf hin, daß didaktisch orientierte Autorensysteme bis heute eine Ausnahme darstellen. Kommerzielle, also von Softwarehäusern angebotene Autorensysteme haben bislang meist die Strategie der Erschließung möglichst großer Marktanteile verfolgt. Dadurch haben sie echte didaktische Qualitäten zugunsten sehr allgemeiner Problemorientierungen eingebüßt. Wenn man daher heute mit dem Computer unterrichten möchte und für diesen Zweck im Computerladen an der Ecke nach einem Autorensystem fragt, dann ist es sehr wahrscheinlich, daß man dort das gleiche Werkzeug empfohlen bekommt, mit dem die unvermeidlichen Diashow–Präsentationen hergestellt wurden, die im Schaufenster dieses Ladens in Endlosschleifen neue Technologien und Sonderangebote anpreisen.

Es hat auch eine Reihe mehr oder weniger erfolgreicher Versuche gegeben, in universitären Kontexten, jenseits der strengen Zwänge des Marktes, Autorenwerkzeuge zu entwickeln. Diese Werkzeuge konnten sich hingegen in den meisten Fällen gerade durch ihre didaktische Problemorientierung nicht behaupten. Sie

[3] Raymond, Eric, S.: [50], S. 24

[4] Ich verwende den Begriff „Hypermedia“ als Kompositum aus den beiden Elementen „Hypertext“ (elektronisch verlinkte Struktur textueller Knoten) und „Multimedia“ (Zusammenwirken mehrerer Medientypen).

sind entweder unabhängig von konkreten Unterrichtsvorhaben aus technischen Motivationen heraus entwickelt worden, wodurch ihre unflexiblen, technokratischen und schwerfälligen Charaktere bei den anvisierten Nutzern (zumeist Lehrenden an den jeweiligen Hochschulen) zu Akzeptanzproblemen führten. Oder sie wurden im Kontext konkret formulierter Lehrsituationen, etwa für bestimmte Kurse oder Seminare hergestellt. Damit war ihnen zwar die Akzeptanz der jeweiligen Lehrenden meist sicher und die Programme waren ideal auf die konkreten Probleme zugeschnitten, jedoch erwiesen sie sich bei der Übertragung auf andere Lehrsituationen als wenig nützlich. Bevor ich aber im Detail auf praktische Probleme der Entwicklung und des Einsatzes von Autorensystemen eingehe, gebe ich zunächst einen kurzen Überblick über die Palette verschiedener didaktischer Entwicklungswerkzeuge, ihre Geschichte und ihre Funktionsweise.

Die Herstellung computerbasierter Lernmittel ist im Vergleich zur Entwicklung ihrer traditionellen Vorgänger tendenziell zeitaufwendiger und — was in der Praxis ein noch schwerwiegenderes Problem darstellt — sie setzt eine entsprechende technische Qualifikation voraus. Hier sollen Autorensysteme als Entwicklungswerkzeuge insbesondere Lehrern in der Praxis eine nutzerfreundliche Alternative zur klassischen Programmierung bieten. Es ist eine alte Programmierer–Weisheit, daß *Nutzerfreundlichkeit* ein anderes Wort für *Programmiererfeindlichkeit* ist[5].

Für die Entwicklung digitaler Unterrichtsmaterialien stehen drei alternative Strategie bzw. drei Autorenwerkzeug–Klassen zur Verfügung, die historisch nacheinander in Erscheinung getreten sind und jeweils Versuche darstellen, die Nachteile ihrer jeweiligen Vorgänger zu beseitigen: Die Programmierung in einer *Programmiersprache*, in einer *Autorensprache* oder mit einem *Autorensystem*. Die von Freimut Bodendorf stammende Abbildung 2.1 zeigt anhand der Kriterien Flexibilität, Produktivität und Benutzerfreundlichkeit, inwieweit sich diese unterschiedlichen Autorenwerkzeuge trotz ihrer gemeinsamen grundsätzlichen Eignung zur Lernmittel—Herstellung unterscheiden.

[5] Eric Raymond: [49], S. 463-464: „[..] to describe systems that hold the user's hand so obsessively that they make it painful for the more experienced and knowledgeable to get any work done."

[6] nach Bodendorf, Freimut.[4], S. 79

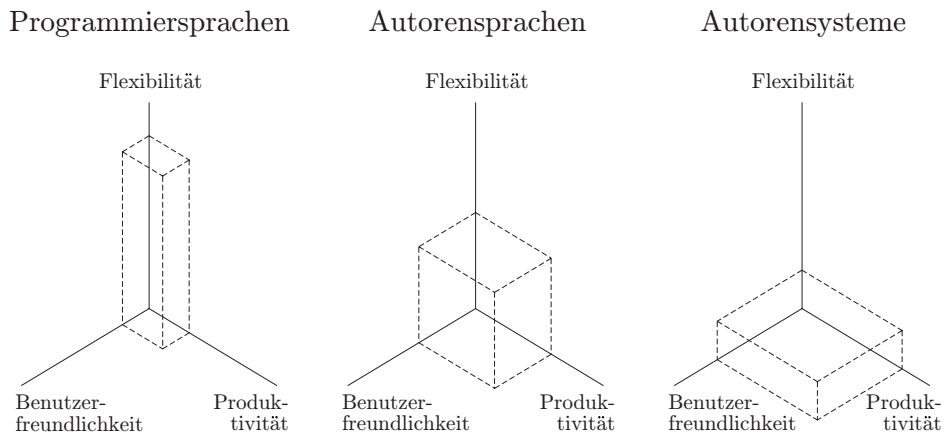


Abbildung 2.1.: Merkmale von Autorenwerkzeugen[6]

Die Verwendung von *Programmiersprachen* wie etwa C, C++, Pascal etc. ist das klassische Verfahren der Programmierung, wie sie seit den frühen Jahren der Computer–Geschichte praktiziert wird. Programmiersprachen haben sich währenddessen kontinuierlich entwickelt: Nachdem zunächst ausschließlich in reinen Maschinen–Instruktionen programmiert wurde, entstanden bald Übersetzungs–Programme (Assembler, später auch Compiler und Interpreter), die *mnemotechnischen* Programmcode verstanden — also Quelltexte, deren Syntax sich vergleichsweise näher an Konzepten der menschlichen Sprache orientiert. Das Erlernen von Programmiersprachen ist aufwendig und ihre Benutzung vergleichsweise unkomfortabel. Sie bieten dem Entwickler jedoch eine enorme Flexibilität und einen weitreichenden gestalterischen Einfluß bei der Erzeugung von Software, die lediglich durch die funktionalen Grenzen der jeweils verwendeten Sprache selbst limitiert werden. Diese Grenzen ergeben sich aus dem Umfang an Schnittstellen, die eine Programmiersprache zu ihrer Umgebung (etwa zum Betriebssystem oder zu Datenbanksystemen) unterstützt.

Autorensprachen sind (zumeist prozedurale) textuelle Entwicklungswerkzeuge. Im Vergleich zu Programmiersprachen zeichnen sie sich durch eine starke Problemorientierung für die Beschreibung von Lerndialogen aus. Diese wird dadurch erreicht, daß in den Architekturen der Sprachen didaktische Annahmen und Modelle vorgegeben werden. Diese können dann mit wenigen einfachen

Aufrufen in Lernumgebungen umgesetzt werden, ohne daß dazu weitreichende Programmierkenntnisse erforderlich sind. Daher weisen sie weniger Flexibilität, aber leichtere Erlernbarkeit und mehr Benutzerfreundlichkeit als echte Programmiersprachen auf. Autorensprachen wurden hauptsächlich auf Mainframe–Architekturen für die Präsentation von Inhalten an Text–Terminals eingesetzt, bevor der PC hohe und billige Rechenleistung an jedem Arbeitsplatz ermöglichte. Heute werden Autorensprachen praktisch kaum noch benutzt. Sie wurden von den benutzerfreundlicheren Autorensystemen abgelöst, die zunächst als (visuelle) Entwicklungsumgebungen eine Schnittstelle zwischen Entwickler und Autorensprache darstellten.

Autorensysteme haben sich weit von der klassischen Programmierung entfernt und werden über graphische Schnittstellen vorwiegend mit der Maus bedient. Das macht sie relativ leicht erlern– und benutzbar aber wenig flexibel. Somit bieten sie eine vergleichsweise höhere Produktivität und benutzerfreundlichen Komfort auf Kosten der individuellen Gestaltung ihrer Produkte. Die Benutzung eines Autorensystems erfordert zunächst nicht das Beherrschen einer Programmiersprache. Viele Produkte (z.B. Metacard oder Toolbook) integrieren jedoch Scriptspachen–Interpreter mit dem Ziel, die Flexibilität textueller Programmierung zumindest teilweise zu bewahren. Basierend auf der zugrundeliegenden *Autorensystem–Metapher* erfordert die Nutzung eines Autorensystems darüber hinaus in jedem Fall eine Form algorithmischen Denkens für die Strukturierung von Inhalten und Interaktionen. Autorensysteme fordern von ihren Anwendern folglich einen Lernprozeß, der durchaus lang und schwierig sein kann und mitunter nach wie vor das Erlernen einer Programmiersprache erfordert.

Zu den bekanntesten Autorensystemen gehören heute Toolbook, Macromedia Authorware und Macromedia Director sowie im WWW–Bereich Microsoft Frontpage und Macromedia Dreamweaver. Diese Produkte sind im weiteren Sinn Multimedia–Präsentationswerkzeuge, die sich durch keine explizite didaktische Ausrichtung auszeichnen. WebCT — ein WWW–basiertes Werkzeug für die Herstellung von Kursen höherer Bildungseinrichtungen — ist ein (seltenes) Beispiel für ein Autorensystem mit expliziter didaktischer Orientierung.

An Bildung und insbesondere an Werkzeuge für die Bildung, wird von Eltern,

Politikern und didaktischen Planern die Erwartung *möglichst effizienter Gestaltung* herangetragen. Das heißt, daß sparsamer Einsatz von Ressourcen zu einem Maximum an Unterrichtserfolg führen soll. Daher geben wir z.B. in der Regel Gruppenunterricht, obwohl Einzelunterricht in vielen Fällen sicherlich intensiveres, konzentrierteres Lernen ermöglichen könnte. Lehrenden stellt sich folglich mit Entwurf, Durchführung und Nachbearbeitung jeder Unterrichtseinheit eine Aufgabe, die letztlich immer als *ökonomische* Aufgabe interpretiert werden kann.

Ein Autorensystem ist ein Programm mit visueller Nutzerschnittstelle, das Lehrenden zur Herstellung digitaler Unterrichtsmaterialien übergeben wird. Aufgabe des Autorensystems ist dabei die Minimierung von Kostenfunktionen (etwa Entwicklungszeit, Materialreproduktion etc.) bei gleichzeitiger Sicherung oder Erhöhung der Lehreffektivität der hergestellten Materialien. Aus Sicht des Entwicklers wird das Autorensystem in einem unbekannten, komplexen und dynamischen Kontext angewendet. Das Urteil über die Qualität eines Autorensystems können nur diejenigen treffen, die die jeweiligen konkreten Lernziele des Unterrichts formulieren: die Lehrenden, die das Autorensystem in der Praxis anwenden.

Die Frage nach dem Grund für die Verwendung eines Autorensystems muß so wie jede andere didaktische Entscheidung auch in jeder Unterrichtssituation neu abgewogen werden. Sie kann bei rationaler Planung immer nach dem Muster „Die Eigenschaft x des Autorensystems gestattet die effizientere Erreichung des Unterrichtsziels y“ ökonomisch begründet werden. Die freie Entscheidung, ob ein Autorensystem eingesetzt wird oder nicht oder ob gelegentlich ein alternatives Produkt verwendet wird, ist jedoch — ebenfalls aus ökonomischen Gründen — nicht völlig frei. Sie wird in der Praxis dadurch erschwert, daß durch eine Investition in eine Technologie, also ihren Erwerb, ihre Installation, ihre fortlaufende Wartung und ihre Unterstützung auf der Client–Seite (der Seite der Lernenden) eine Bindung an diese Technologie erfolgt.

Die Gründe für und wider Autorensystem–Einsatz oder generell für und wider computervermittelten Unterricht werden typischerweise in unterschiedli-

chen Kontexten artikuliert. Gründe *für* die Verwendung von Autorensystemen werden z.B. in der Produktwerbung dieser Programme sowie im Handel und in Fachzeitschriften verbreitet und durch die generell positive gesellschaftliche bzw. wissenschaftlich–theoretische Einstellung gegenüber Informationstechnologien in der Bildung verstärkt. Diese Argumentation soll Werbezwecke erfüllen und impliziert sehr allgemeine Versprechungen über effektives Lehren. So heißt es z.B. auf den Webseiten der Firma Asymetrix[7] über deren Online–Autorensystem „Toolbook II Assistant“:

> „ToolBook II Assistant is the e-Learning solution for business professionals including trainers, sales and marketing managers, educators, and human resource managers who need to create and deliver training and educational courseware quickly, effectively, and economically. Using the intuitive drag–and–drop interface featured in Assistant, anyone can create high–impact, interactive e-Learning courses. Assistant makes it easy to distribute your courseware, whether over the Internet, an intranet or LAN, or on CD-ROM.“

Durch diese Form der öffentlichen Kommunikation über Autorensysteme erreichen die positive Bewertung dieser Software und die hohen Erwartungen an ihre Verwendung eine weite Verbreitung. Gründe *gegen* die Verwendung von Autorensystemen werden hingegen vornehmlich in der Praxis als Resultate konkreter Ernüchterungen artikuliert und finden typischerweise keine breite öffentliche Wahrnehmung. Vor diesem Hintergrund ist es schwierig, ein realistisches Bild von den Motiven zu zeichnen, die in der Bildungspraxis zu Verwendung oder Ablehnung von Autorensystemen führen.

Die fünf typischen Aspekte, die Autorensystem–Hersteller in ihren Argumentationen für die Verwendung ihrer Produkte regelmäßig heranziehen sind auch diejenigen Aspekte, die den Computer seit Jahrzehnten als vielversprechendes Medium für Unterricht erscheinen lassen. Sie sind Ausdruck desselben ökonomischen Motivs: Lehre zu *effektivieren*:

[7] http://www.asymetrix.com/products/assistant.html

wirtschaftliche Aspekte betreffen unmittelbar finanzielle Gesichtspunkte der Unterrichtsgestaltung. Autorensysteme werden bereitgestellt, um bei vermindertem Ressourcenaufwand eine mit alternativen Verfahren vergleichbare Unterrichts–Effektivität zu erreichen (oder sie zu übertreffen). Unter alternativen Verfahren können hier traditionelle (nicht computervermittelte) Unterrichtsverfahren oder die Verwendung von Programmier– und Autorensprachen verstanden werden. Zentrale Wirtschaftlichkeits–Faktoren computervermittelter Lehre sind die Lernerzahl, die Lernzeit, der Umfang der Lerninhalte, Arbeitsaufwand (bzw. –kosten) des/der Lehrenden, Arbeitsaufwand (bzw. –kosten) des/der Lernenden und Ausstattungskosten (hauptsächlich für Hard– und Software sowie deren Instandhaltung). Da computerbasierte Lehrmittel vergleichsweise entwicklungsintensive Softwareprodukte darstellen, resultiert aus der durch Autorensystem–Verwendung erzielten Ersparnis an Entwicklungszeit etwa im Vergleich zur Entwicklung auf Basis einer traditionellen Programmiersprache ein starker wirtschaftlicher Anreiz. Bodendorf zitiert „eine von vielen Faustregeln“, die besagt,

> „daß sich der Entwicklungsaufwand für einen computergestützten Unterricht durch den Einsatz eines Autorensystems auf ungefähr ein Drittel vermindert, z.B. also von 300 auf 100 Entwicklungsstunden für eine Unterrichtsstunde verkürzt.“[8]

Die „Multimedia Authoring Systems FAQ“[9] nennt in diesem Kontext sogar Faktor acht (die Differenz zwischen diesen beiden „Faustregeln“ deutet auf Schwierigkeiten bei Effekivitäts–Vorhersagen für Autorensystem–Einsatz). Aufgrund des erforderlichen Entwicklungsaufwandes professioneller Multimedia–Anwendungen ist es sinnvoll, frühe Entwicklungs–Versionen dieser Programme mit Autorensystemen zu implementieren, bevor sie später ggf. mit anderen Werkzeugen — etwa Compiler– oder Dokumentbeschreibungssprachen — hergestellt werden. Daher kommen

[8] Bodendorf, Freimut.[4], S. 86

[9] Olszewski, Pawel[41]

Autorensysteme auch in außerschulischen didaktischen Projekten für die schnelle Entwicklung von Prototypen zum Einsatz.

logistische Aspekte beziehen sich auf die Kommunikation zwischen Lehrenden und Lernenden. Computervermittelte Lehre erschließt zuvor nicht realisierbare didaktische Optionen und gestattet unterrichtliche Konstellationen, die traditionelle Unterrichtsformen nicht ermöglichen können. So macht die Möglichkeit, über Datennetze *geographische Grenzen* zu überschreiten, die computervermittelte Lehre z.B. zu einem geeigneten Vehikel für die Realisation interkultureller Unterrichtsprojekte. Angesichts unseres exponentiell wachsenden Wissens–Volumens und der resultierenden Tendenz zur inhaltlichen Spezialisierung gestatten Netzwerke auch die Lehre von Inhalten, für deren Realisierung in lokalen Rahmen kein ausreichender Lernbedarf besteht. Die Loslösung von traditionellen Schulklassen durch computervermittelte Lehre gestattet weiterhin die Versorgung von vergleichsweise großen Lernergruppen sowie *zeitlich asynchrone* individuelle Lernstrategien.

fachliche Aspekte betreffen die *Qualifikation Lehrender*, die digitale Lernmittel bereit– und/oder herstellen sollen: Die im Vergleich zu Programmiersprachen und Autorensprachen komfortablen Autorensysteme finden Verwendung, wenn computervermittelte Lehre angeboten werden soll, die Kompetenz Lehrender die Nutzung komplexerer Entwicklungsverfahren aber nicht einschließt.

didaktische Aspekte: Unter didaktischen Gesichtspunkten kann die Verwendung von Autorensystemen bzw. durch sie ermöglichte computervermittelte Lehre *effektiver* sein als andere Unterrichtsformen. Es liegt in der Verantwortung des jeweils Lehrenden, Unterrichtssituationen zu analysieren und die entsprechenden Kriterien zu identifizieren. Denkbare Gründe für didaktisch motivierte Autorensystem–Verwendung sind sehr zahlreich. Stichwortartig seien hier die Möglichkeiten genannt, Unterrichtssituationen durch Computereinsatz aus sozialen Kontexten zu lösen, individuelle Lerngeschwindigkeiten zu realisieren oder durch Codierung von Inhalten

etwa in alternativen Medienobjekten unterschiedliche Lernerpräferenzen zu versorgen.

medienpädagogische Aspekte: Die zunehmende Technisierung unserer Umwelt erfordert die Vermittlung technischer Kompetenzen sowie die Heranführung Lernender an neue Technologien. Die Verwendung von Autorensystemen führt zur Bereitstellung computer– oder netzwerkbasierter Unterrichtsmittel, die Lernende zwingt, Medien wie den Computer oder das Internet zu nutzen, ggf. Schwellenängste zu überwinden und als Nebeneffekt des inhaltlichen Lernens *zusätzlich* Medienkompetenz zu erreichen, die sich auch außerhalb des Unterrichts als nützlich erweisen kann.

Die folgenden Auszüge aus der Newsgroup `schule.software` sollen die Rolle, die Autorensysteme in der Bildungspraxis spielen sowie die vereinzelten Annäherungen Lehrender an Autorensysteme, entstehende Probleme und deren Diskussion exemplarisch dokumentieren. Ich bin auf diese Diskussion Anfang März 2000 gestoßen. Die Beiträge habe ich bis auf die Löschung von Personennamen unverändert übernommen.

```
Ich bin Deutschlehrer und suche nach einem Autorenprogramm, mit
dem ich fuer meine Schueler Uebungen erstellen kann. Calis (DOS)
gefaellt mir gut, aber es gibt keine
(Mausmarkierungsmoeglichkeiten). Clic 3.0 dagegen ist grafisch
und vom Bedienungskomfort sehr ansprechend, ist aber zu
beschraenkt in der Moeglichkeit, Fehler zu kommentieren. Und
3000 DM Programme kommen fuer eine Schule (Gymnasium) nicht in
Frage.
```

Dieser Beitrag wurde mit Hinweisen auf die Produkte *Matchware Mediator*, *Eckermann* und *Asymetrix Toolbook* beantwortet und von weiteren Fragen nach Softwareprodukten zur Schul– und Stundenplanverwaltung begleitet. Daraufhin stellte sich einem anderen Teilnehmer die folgende Frage:

```
Alle wollen sie Programme haben. Warum schreibt keiner die
Programme selber?
```

Mit der darauf folgenden Antwort wurden zentrale Punkte dieser Problemanalyse angesprochen:

```
Die Sache ist doch relativ einfach, IMO:
- Es fehlt das Geld dafuer,
- Es fehlt noetiges Fachwissen
- Keiner hat die Zeit dazu

Immerhin sitzt man einem Programm - je nach Umfang - von einigen
Wochen bis zu mehreren Monaten, manchmal sogar Jahren.
Und, um es mal auf Schulen zu projizieren: Welcher Lehrer hat
schon Ahnung von kompletter (professioneller)
Softwareentwicklung? Daran hapert es leider zu oft.
```

Der hier präsentierte Diskussions–Ausschnitt hat selbstverständlich weder einen Anspruch auf inhaltliche Vollständigkeit noch auf repräsentative Aussagekraft. Sie deckt sich jedoch mit meiner persönlichen Erfahrung im Bereich des computervermittelten Unterrichts an mehreren Universitäten und ist ein seltenes Beispiel öffentlicher Autorensystem–Kritik aus der Bildungspraxis. Diese kurze Zusammenfassung der Newsgroup–Diskussion enthält Aussagen über zentrale Probleme in der Praxis: Es ist nicht möglich, erwünschte Designmerkmale eines Programms auf ein anderes zu übertragen, es mangelt an finanziellen Mitteln, an technischer Qualifikation des Lehrpersonals und an dessen Arbeitszeit. Obgleich insbesondere die letzten der genannten Gründe wohl eher generelle Probleme unseres Bildungssystems sind, die nicht direkt das Design von Autorensystemen betreffen, bleibt anzumerken, daß es diesen Werkzeugen trotz diesbezüglicher Versprechungen nicht gelingt, breite Verwendung im Bildungsalltag zu erreichen.

2.2. Unterricht

Der Computer kann auf unterschiedliche Weise im Unterricht eingesetzt werden: zum Lernen *mit* dem Computer (Computer als Medium für Transport und Darstellung von Lernmaterial) und zum Lernen *über* den Computer (wie funktioniert er, wie wird er benutzt?). In dieser Arbeit möchte ich Fragen nachgehen, die sich bei der ersten der beiden Einsatzweisen stellen: Bei der Verwendung des Computers zur Darstellung von Lernmitteln und als Plattform zur Interaktion mit Unterrichtsmaterialien und anderen Lernern, kurz: bei der Verwendung von Computern und Software als Lehrwerkzeuge. Auf die Vermitt-

lung von Informatik– und EDV–Kenntnissen ist diese Arbeit nur für die Fälle anwendbar, in denen der Computer auch als Unterrichts-Werkzeug, (etwa zur Gestaltung und Präsentation von Lernmaterial) eingesetzt wird.

Unterricht ist eine institutionalisierte oder zu einem bestimmten Maß organisierte Situation, in der ein Individuum oder eine Gruppe einem anderen Individuum bzw. einer anderen Gruppe eine Umgebung bereitstellt, die Lernen fördert oder ermöglicht.

Dabei können Medien wie der Computer verwendet werden. Neue Medien sind potentiell neue Werkzeuge für den Unterricht und so weckte bereits die Entwicklung der Massenmedien Radio und Fernsehen Aufmerksamkeit und hoffnungsvolle Erwartungen in der Didaktik. Diese audiovisuellen Massenmedien können heute praktisch in jedem Haushalt empfangen werden, und die Herstellung von Inhalten für Radio und Fernsehen ist vergleichsweise einfach und billig geworden. Dennoch werden diese Technologien nicht in dem Maß im Unterricht eingesetzt, wie zunächst erwartet wurde. Grund hierfür ist vor allem die Anordnung der Kommunikations–Partner bei Radio und Fernsehen: Einer kommuniziert mit sehr vielen. In diesem Szenario ist es den Gestaltern des Unterrichts nicht möglich, Merkmalen und Bedürfnissen individueller Lerner oder konkreter Lerngruppen gerecht zu werden, wie es ein Lehrer vor Ort tun kann. Der Computer als individueller, interaktiver Dialogpartner ist Radio und Fernsehen daher didaktisch potentiell überlegen: Mit der erforderlichen Software ausgestattet kann der Computer Unterrichtsmaterialien inhaltlich und zeitlich personalisiert präsentieren. Daher wird die Geschichte des Computers begleitet von einer kaum überschaubaren Menge didaktischer Theorie, die das Unterrichtspotential des Werkzeugs Computer überwiegend positiv einschätzt.

1946 wird aus 17.468 Vakuumröhren der erste elektronische Universalrechner ENIAC an der Moore School of Engineering (University of Pennsylvania) in Betrieb genommen, um für die US–Streitkräfte dringend benötigte ballistische Tabellen herzustellen. Schon ein Jahrzehnt später beginnen Lehrende – auch außerhalb des Militärs – vom Einsatz des Computers im Unterricht zu träumen und Visionen von der Schule der Zukunft zu entwerfen. Bereits im Jahr 1957 fordert Simon Ramo ([48], S. 18) eine radikale Umgestaltung der Schule und die Nutzung von Rechenanlagen sowohl für die Durchführung als auch für die Verwaltung von Unterricht.

Das folgende Zitat aus Seymour Paperts „Revolution des Lernens“ aus dem Jahr 1994 hingegen beschreibt die ernüchternde Realität des heutigen Unterrichts. Die euphorischen Erwartungen sind im Laufe der vergangenen Jahrzehnte abgeklungen, und nachdem sehr viel Energie in die Erforschung des computerbasierten Unterrichts investiert wurde, läßt die digitale Revolution des Lernens noch immer auf sich warten:

> „Man stelle sich eine Gruppe Zeitreisender aus einem früheren Jahrhundert vor, unter denen sich ein Kreis von Chirurgen und ein weiterer Kreis von Lehrern befindet, beide ganz gespannt darauf zu erfahren, wieviel sich in ihrem Beruf seit hundert Jahren verändert hat. Man stelle sich weiter die Verwirrung der Chirurgen vor, wenn sie sich plötzlich im Operationssaal eines modernen Krankenhauses wiederfänden. Sie würden zwar erkennen, daß wohl irgendeine Art Operation durchgeführt wird, könnten sich vielleicht sogar denken, um welches Organ es geht; sie würden jedoch in den meisten Fällen nicht genau schlau daraus werden, was der Chirurg erreichen will oder welchen Zweck die vielen seltsamen Instrumente haben, die er und das Operationsteam verwenden. Die wie Rituale anmutenden Handlungen bei Sterilisierung und Anästhesie, das Piepsen der elektronischen Geräte, ja selbst die hellen OP–Lampen, alles, was heute jeder aus dem Fernsehen kennt, würde auf sie äußerst befremdlich wirken.
>
> Die zeitreisenden Lehrer würden auf das Klassenzimmer einer mo-

> dernen Grundschule ganz anders reagieren. Sie würden sich vielleicht über einige unbekannte Gegenstände wundern. Sie könnten vielleicht feststellen, daß sich einige Standardmethoden geändert haben — und hätten wahrscheinlich unterschiedliche Meinungen darüber, ob dies Veränderungen zum Guten oder zum Schlechten seien — sie würden jedoch den Sinn der meisten Vorgänge vollkommen verstehen und könnten den Unterricht ohne größere Schwierigkeiten selbst weiterführen."[10]

Anstelle von Chirurgen könnte Papert für diesen Vergleich ebenso die Tätigkeiten von Ingenieuren, Journalisten oder Reisekaufleuten nennen und deren Aufgabenbereiche jeweils mit dem Unterrichten vergleichen. Auch sind seine Beobachtungen von Problemen der Grundschule durchaus auf höhere Bildungseinrichtungen übertragbar, denn der technische Fortschritt hat auch in Gymnasien oder Hochschulen nicht den Effekt, den man sich bislang erhofft hat und weiterhin erhofft. Ganz offensichtlich fällt es in unterschiedlichen Aufgabenbereichen unterschiedlich leicht, den Computer als Werkzeug zu nutzen, und paradoxerweise kann ausgerechnet die Lehre, die seit langem große Hoffnungen in die Informationstechnologie setzt, ihre Visionen nur sehr schwer realisieren.

Unterricht befindet sich in einer paradoxen Situation: Zum einen soll er den Menschen in die Lage versetzen, sich in sozialer, politischer, kultureller, technischer, wissenschaftlicher etc. Hinsicht mit der Umwelt auseinanderzusetzen, in dieser Umwelt eine eigene Position einzunehmen, um hieraus das eigene Dasein gestalten zu können. Dies trifft in zunehmendem Maß neben jungen Menschen auch auf Erwachsene unterschiedlicher Altersstufen zu. Der Unterricht ist somit der Schüssel zum Erreichen zentraler gesellschaftlicher Ziele und zur Wahrung hoher sozialer Werte. In dieser Hinsicht kennen wir keine Alternative zum Unterricht, und so liegt es theoretisch auf der Hand, ihn mit hoher Priorität und unter hohem Kraftaufwand zu betreiben. Andererseits ist Unterricht jedoch ein wirtschaftliches Unterfangen, das ebenso ökonomischen Gesetzen unterliegt wie andere, ideell weniger zentrale Bereiche der Gesellschaft.

[10] Papert, Seymour: [45], S. 27

Bildungssysteme verfügen nur über begrenzte Geldmittel, mit denen sie ebenso haushalten müssen wie der einzelne Lehrer, der in seinem Unterricht unter Einsatz weniger Mittel ein Maximum an Effizienz erreichen soll. Wir haben abwechselnd nicht genug Lehrer, um den Bedarf an Unterricht zu decken, nicht genug Geld, um den Bedarf an Lehrern zu decken oder nicht genug Lerner um den Bedarf an qualifizierter Arbeitskraft in der Gesellschaft zu decken. Gleichzeitig tendieren unsere Bildungseinrichtungen zu verstärktem Wettbewerb untereinander, Unterricht verliert unter dem ökonomischen Druck mehr und mehr seine ideelle Bedeutung und wird zur alltäglichen Dienstleistung. Rationalisierung ist eine zentrale Strategie, mit der die entstehenden ökonomische Probleme im Unterricht gelöst werden sollen. Auf der Suche nach entsprechenden Strategien haben wir im Laufe der Zeit immer wieder versucht, Technologie im Unterricht nach den Vorbildern der Arbeitswelt einzusetzen. So haben wir im Industriezeitalter (letztlich ohne nachhaltige Folgen für unsere Unterrichtsmethoden) Lern*maschinen* eingesetzt und setzen heute — im Informationszeitalter — Lernmittel–Datenbanken und Kommunikationsnetzwerke ein. Veränderungen, die durch Informationstechnologie und computerbasierte Kommunikation unser Leben prägen, erzeugen weiterhin neue Herausforderungen an Unterricht und Bildung.

Die Geschwindigkeit technologischer Innovation beispielsweise ist bereits heute zu hoch als daß das traditionelle Modell der Lehrer–Ausbildung die Abdeckung des resultierenden Bedarfs nach Bildung sichern könnte. Die sogenannte Globalisierung[11] stellt Unterricht ferner vor die Aufgabe, inhaltliche und formale internationale Anpassungen und Kooperationen durchzuführen und z.B. Bildungsabschlüsse international kompatibel zu gestalten. Vor dem Hintergrund dieser durch Technologie angetriebenen Veränderungen bleibt es abzuwarten, ob es den Bildungssystemen gelingt, sich neuer Technologien anzunehmen, sie für sich zu nutzen und sowohl alte als auch neu anfallende Probleme zu lösen.

[11] Dieser Begriff erscheint mir zumindest in erziehungswissenschaftlicher Hinsicht als problematisch, da er bislang eher ein wirtschaftliches Zweckbündnis reicher Industrienationen als eine tatsächliche Intensivierung weltweiten Austauschs bezeichnet.

2.3. Computer

Der Computer, so wie wir ihn heute benutzen, wurde unter dem Druck des Zweiten Weltkriegs und des kalten Krieges entwickelt, um umfangreiche Berechnungen in der Ballistik zu lösen. Er ist die elektronische Fortsetzung der Entwicklung mechanischer Rechenmaschinen, die bereits in den Jahrhunderten zuvor zur Bewältigung umfangreicher Kalkulationen (zumeist in Handel und in der Waffentechnik) entwickelt wurden. Die Architektur des modernen Computers (der sogenannten von–Neumannschen Maschine) hat ihren Ursprung in dem Bedarf nach programmgesteuerter Verarbeitung gleichartiger Daten, die auf diese Weise schneller und zuverlässiger bewältigt werden kann als durch menschliches Rechnen. Grundlegende Designentscheidungen wurden in der Entwicklung des Computers nicht mit dem Ziel getroffen, eine Maschine zum individuellen Lösen von alltäglichen Aufgaben zu schaffen, sondern einen Apparat zum effizienten Abarbeiten umfangreicher Rechenaufgaben.

Durch die Verfügbarkeit der Vakuumröhre und später des Transistors wurde es möglich, mathematische und mechanische Konzepte von Rechenmaschinen vergangener Jahrhunderte elektronisch nachzubilden und fortzuentwickeln. Diese Entwicklung wurde seither hauptsächlich durch den Stand des technologischen Fortschritts, durch militärische Anwendungen und — zumindest was die Entwicklung der Mensch–Maschine–Schnittstelle angeht — den Erfolg des Computers auf dem Spielemarkt geprägt. Uns heute selbstverständliche Konzepte wie die digitale Repräsentation von Information unter Verwendung des binären Zahlensystems und die von-Neumann-Architektur haben technische, ökonomische und letztlich historische Entstehungsgründe. Wir müssen uns darüber im klaren sein, daß der Computer selbst ein Designproblem[12] darstellt.

Moderne Anwendungen sollen nicht primär umfangreiche Rechenaufgaben lösen, sondern Nutzer dabei unterstützen, sozial zu interagieren oder Medien–Inhalte herzustellen. Bei der Nutzung von Videokonferenz–Systemen oder Online–Whiteboards tritt die planvolle Rechenarbeit des Computers in den Hintergrund und spontanes Interagieren in den Vordergrund. Die letztlichen

[12] Eine Definition hierfür liefere ich in Abschnitt 2.4.

Programmabläufe existieren in diesem Kontext nicht mehr ausschließlich als arithmetisch–logische Algorithmen, implementiert in Software. Sie müssen als höchst individuelle Prozesse verstanden werden, deren zugrundeliegende Regeln Manifestationen sowohl von programmierten arithmetisch–logischen Algorithmen als auch von Psyche, Geschmack und Aufgaben von Nutzern sind. Designentscheidungen, die in der frühen Entwicklung des Computers getroffen wurden, dominieren nun als problemfremde Artefakte die Kommunikation zwischen Mensch und Maschine und somit indirekt die Kommunikation zwischen Menschen. Darüber hinaus sind moderne Anwendungsgebiete so vielfältig, daß das Werkzeug Computer nur die Qualität eines kleinsten gemeinsamen Nenners hat. Er ist eine Kompromiß–Lösung und daher a priori nicht optimal geeignet, ein konkretes Problem zu lösen. Wir stehen folglich vor der Aufgabe, mit einem Werkzeug zu unterrichten, das auch als Steuerung für Raketen, als Werkzeug zur Buchung von Flugtickets, als Supermarkt–Kasse und als Zeichenbrett benutzt wird. Im Gegensatz zu diesen anderen Einsatzfeldern ist der Computer als Unterrichtswerkzeug noch nicht sehr erfolgreich. Die Vielseitigkeit des Computers als Problemlöser hat offenbar Grenzen.

Werkzeuge, von denen eine hohe Vielseitigkeit erwartet wird, werden oft mit Schweizer Taschenmessern verglichen. Dieser Vergleich verdeutlicht anschaulich den problematischen Charakter des Computers: Alle Funktionen, die ein Schweizer Taschenmesser vereint, können damit ausgeführt werden – aber nicht besonders gut. Die Verwendung eines solchen Messers ist meistens eine Kompromiß–Lösung und für alle seine Funktionen gibt es effizientere, spezielle Werkzeuge, mit denen man besser arbeiten kann. Jemand, der die Auswahl zwischen einem ergonomisch gestalteten Dosenöffner und einem Schweizer Taschenmesser hat, wird eine Dose mit hoher Wahrscheinlichkeit mit dem Dosenöffner öffnen, weil er mit dem spezialisierten Werkzeug bequemer, schneller, sicherer, sauberer usw. sein Ziel erreichen kann. So verwundert es auch nicht, wenn Lehrer, die für ihren Unterricht zwischen Computer und Klassenzimmer wählen können, sich gegen den Computer entscheiden.

Für den Online–Unterricht finden wir neben der Rechenmaschine Computer

keine problemorientiertere, effizientere Alternative. Jeder verfügbare moderne Computer ist primär ein Rechner und kein Lernmedium. Potentiell kann der Computer ein Lernmedium sein, denn mathematisch betrachtet ist der Computer (der formal den Kriterien der „Turing–Maschine“ entspricht) eine Universalmaschine, die das Verhalten jeder anderen Maschine simulieren kann. Wie sieht nun aber die Maschine aus, die der Computer simulieren soll, um ein effektives Lernmedium zu werden? Was könnte ihn dazu machen? Was für ein *Design* brauchen wir?

Was einen Computer problemorientert agieren läßt, ist seine Software. Software ist jedoch letztlich nichts anderes als Bibliotheken von Symbolen für konkrete oder abstrakte Ideen, die über mathematische Regeln und aristotelische Logik miteinander in Beziehung stehen. Mit diesem Werkzeug Probleme der realen Welt nachbilden zu wollen bereitet mitunter außerordentliche Schwierigkeiten. Autorenwerkzeuge sind die Schnittstelle zwischen Rechner und didaktischem Alltag, die die Universalmaschine Computer in ein problemorientiertes Werkzeug für computervermittelten Unterricht verwandeln soll.

Seit der Computer Mitte der 1960er Jahre auch außerhalb von Forschungsstätten und Militär breite Anwendung fand, können wir ein Phänomen beobachten, das als Softwarekrise bezeichnet wird: Die Leistungsfähigkeit von Hardware–Technologien hat sich (eher quantitativ in bezug auf Verarbeitungskapazitäten und Geschwindigkeiten als qualitativ in bezug auf Bauweisen) exponentiell fortentwickelt während Software–Technologien mit dieser Entwicklung nicht mithalten konnten. Software ist in den vergangenen Jahrzehnten im Umfang mit den jeweils verfügbaren Speichern gewachsen — jedoch ist das, was Software heute zu leisten vermag angesichts früher Erwartungen, jahrzehntelanger Erforschung Künstlicher Intelligenz und riesigen Rechenkapazitäten eher enttäuschend. Seit der Verfügbarkeit des Internet und schulischer Netzwerke in den 90er Jahren können wir eine Wiederholung dieses Phänomens beobachten: Die fortschreitende hardwareseitige Verbreitung und die Weiterentwicklung von Netzwerken wird nicht begleitet von entsprechenden Innovationsraten in den Bereichen Lernsoftware und Autorensysteme bzw. deren Akzeptanz.

2.4. Das bösartige Problem Unterrichtsentwurf

Die Aufgabe des Lehrers ist — ich habe es bereits angesprochen — die Planung und Durchführung von Interaktionsprozessen, die Lernern das Lernen ermöglicht bzw. erleichtert.

Der Computer und die verwendete Software sollen dem Lehrer dabei als Werkzeug dienen. Diese Aufgabe besteht aus einer Anordnung von einem verantwortlichen Problemlöser (Lehrer), einem Problemraum (zu unterrichtende Lerner) und einem Lösungsraum (Unterricht, der den Vorstellungen des Lehrers und/oder der Lerner entspricht)[13]. Diese Anordnung entspricht denen vieler anderer Aufgaben in unterschiedlichen Dienstleistungsbereichen — wie z.B. der Buchung von Flugtickets: Ein Problemlöser (Reisekaufmann) wird mit einem Problemraum (reisewillige Kunden) konfrontiert, und arrangiert eine Lösung (Buchung, die seinen Vorstellungen und/oder den Vorstellungen seiner Kunden entspricht). Aus diesem Blickwinkel betrachtet wirken beide Aufgaben, die Online–Lehre und das Verkaufen von Reisetickets, durchaus vergleichbar und so liegt es nahe, von dem Werkzeug Computer in beiden Kontexten sowohl eine ähnliche Nützlichkeit als auch eine vergleichbare Akzeptanz seitens der jeweils Verantwortlichen zu erwarten. Für beide Aufgaben gibt es umfangreiche Computeranwendungen, deren Verbreitung, Nützlichkeit und Akzeptanz unverkennbar auf grundsätzliche Unterschiede beider Aufgabenbereiche hindeuten.

SABRE ist ein Programm, mit dem Reisekaufleute mit dem Computer über ein Netzwerk Flugtickets buchen können. Es ist das erste zivile Großprojekt in der Geschichte der Software–Entwicklung, das alle in einem Problem relevanten Elemente in Software modelliert und für netzwerkbasierte Echtzeit–Transaktionen bereitstellt. Die erste Version wurde bei IBM in der Zeit zwischen 1960 und 1963 von etwa 200 Technikern implementiert und war mit einer Million Zeilen Programmcode das größte Software–Projekt seiner Zeit. In der Datenbank–

[13] Das Abstraktionsniveau dieser (entwerferischen) Terminologie erscheint mir hier angemessen, da sie Entwurfs–Situationen nicht nur in der Software–Entwicklung sondern auch im Unterrichtsentwurf fassen kann und gleichzeitig leicht und ohne besondere Vorkenntnisse verstanden werden kann. Die Begriffe „Problem Space“ und „Solution Space“ werden in diesem Sinn beispielsweise von Bruce Eckel [11] S. 29 ff. verwendet.

Zentrale in Briarcliff Manor, etwa 50 Kilometer nördlich von New York City, bedienten zwei IBM 7090 Mainframe–Computer mit der damals gigantischen Speicherkapazität von 800 Megabyte Tausende Terminals in 50 Städten, die an über 10.000 Meilen geleaste Telefonleitungen angeschlossen waren. Zum Beispiel die Lösung des Problems der Echtzeit–Platzreservierung, bei der unabhängige gleichzeitige Datenbank–Zugriffe etwa auf identische Sitzplätze nicht kollidieren dürfen, ist technisch alles andere als trivial. Dennoch wurde es in SABRE zuverlässig gelöst und so war und ist SABRE ein überaus erfolgreiches Werkzeug. Es prägte die Verarbeitung von Transaktionsgeschäften von Banken, Eisenbahngesellschaften und Versicherungen und seine nachfolgenden Versionen und ähnliche Konkurrenz–Produkte sind bis heute eine selbstverständliche Voraussetzung für Online–Buchungen von Flügen, Hotels, Mietwagen usw.

Abbildung 2.2.: Zentrale des SABRE Reservierungs–Systems in Briarcliff Manor in den 60er Jahren

WebCT („World Wide Web Course Tools“) ist ein Autorensystem, mit dem Lehrende Online–Unterricht planen und durchführen können. Es wurde (über 30 Jahre nach SABRE) zunächst an der Universität von British Columbia in Vancouver entwickelt und seit 1995 von einer unter gleichem Namen gegründeten Firma vermarktet und weiterentwickelt.

WebCT ist insofern mit SABRE vergleichbar, als es als erstes Softwareprodukt seiner Art versucht, seinen „Problem–Weltausschnitt“ umfassend nachzumodel-

lieren und die Realität des Unterrichts weit über die bloße Bereitstellung von Dokumenten hinaus institutionsorientiert abzubilden und Lehrende, Lernende, Tests und Noten zu verwalten. Da diese Eigenschaft praktisch kein anderes Online–Autorensystem teilt, führt WebCT heute auf dem Markt dieser Werkzeuge. Obwohl es ein kommerzielles Produkt ist, verfügt heute eine große Anzahl von Universitäten über lizensierte Kopien. Lehrenden wird somit nahezu weltweit ein Werkzeug für ihren Online–Unterricht angeboten. Trotz seiner Quasi–Monopolstellung und dem globalen Bedarf nach Online–Lehre vermag WebCT aber nicht, einen Erfolg zu verbuchen, der nur annähernd mit dem SABREs zu vergleichen wäre. Lehrende (und Lernende) kritisieren das Programm unter anderem weil es zu rigide und nicht ausreichend an individuelle Anforderungen anpaßbar ist, weil es umständlich zu bedienen ist und das Nutzer–Interface schlecht gestaltet und unangemessen anachronistisch wirkt.

Abbildung 2.3.: Startseite eines WebCT–Kurses

Wären solche Kritikpunkte bei einem System für Ticketbuchung denkbar gewesen? Warum war SABRE bereits vor über 30 Jahren so problemlos zu bedienen während WebCT heute Nutzer abschreckt? Warum wünschen sich Lehrende individuelle Anpaßbarkeit, während Reisekaufleute auch ohne diese Eigenschaft

arbeiten können?

Das Wort „Unterrichtsgestaltung“ selbst weist auf einen grundsätzlichen Unterschied zwischen dem Unterrichten und anderen, typischerweise computerisierten Aufgaben hin: **Unterrichten ist eine Design-Aufgabe und Lehrer sind Designer**.

Diese These läßt sich mit Hilfe der Design–Theorie illustrieren, indem man die vielen für den Begriff *Design* existierenden Definitionen auf *Entwurf und Durchführung von Unterricht* anwendet — dies führt praktisch ausnahmslos zu zutreffenden und einleuchtenden Einsichten über das Problem Unterrichtsgestaltung. Ich möchte dies belegen, indem ich hier exemplarisch eine Definition von Horst Rittel und Melvin Webber zitiere. Sie bezeichnen Designprobleme als *bösartig* und unterscheiden sie von anderen, *zahmen Problemen*, indem sie die folgenden zehn charakteristischen Eigenschaften von Designproblemen[14] aufzählen:

1. Designprobleme sind nicht eindeutig und endgültig formulierbar

2. Designprobleme haben keine Stoppregel: Im Gegensatz zu zahmen Problemen können sie ihre erfolgreiche Bearbeitung bzw. ihren Abschluß nicht anzeigen.

3. Lösungen von bösartigen Problemen sind nicht richtig oder falsch, sondern gut oder schlecht

4. Es gibt für die Qualität von Lösungen keinen unmittelbaren Test bzw. Kontrolle

5. Jede Lösung eines bösartigen Problems ist eine „one–shot–operation“: Es gibt keine Möglichkeit, durch trial–and–error zu lernen, jeder Versuch zählt.

[14] Rittel, Horst W. J. und Webber, Melvin M.: [51], S.160ff.

6. bösartige Probleme besitzen keine vollständige Liste von Lösungen, ebensowenig läßt sich eine endliche Auflistung von Lösungsoperationen formulieren

7. Es gibt keine zwei gleichartigen Designprobleme

8. Jedes Designproblem kann als Symptom eines anderen Problems aufgefaßt werden

9. Für Designprobleme gibt es eine Vielzahl „richtiger“ Lösungen

10. Der Entwerfer hat kein Recht auf Versagen

Zahme Probleme hingegen seien dadurch charakterisiert, daß sie in rationalen Einzelschritten zu lösen sind, wie das Lösen einer mathematischen Gleichung, eine chemische Analyse, das Schachspiel oder das Buchen eines Tickets: Für sie gibt es allgemeine Regeln, und aufgrund des Vorhandenseins dieser formalen Kriterien kann sich eine erfolgreiche Lösung eines zahmen Problems auch als solche zu erkennen geben: Wir wissen, wann eine mathematische Gleichung gelöst, eine Partie Schach beendet oder ein Ticket gebucht ist. Die erfolgreiche oder schlechte Lösung eines bösartigen Problems hingegen kann sich als solche ohne individuelles menschliches Urteilen weder zu erkennen geben noch gibt es für sie überhaupt definitiv richtige oder falsche Lösungen. Da sie nur in der praktischen Anwendung ihre Qualität beweisen können, haben sie (ob gut oder schlecht) immer einen direkten Effekt auf die Gesundheit, die Arbeit, die Freiheit usw. — letztlich das Leben von Menschen.

Obgleich Rittel und Webber ihre Charakterisierung aus Sicht der Designtheorie vornehmen, betonen sie, daß über die Grenzen klassicher designerischer Arbeitsfelder hinaus auch andere Probleme in sozialen Handlungsfeldern bösartig sind. Sie nennen — eher beiläufig — auch die didaktische Planung als Beispiel:

> „Wicked probelms [..] have neither of these clarifying traits; and they include nearly all public policy issues — whether the question concerns the location of a freeway, the adjustment of a tax rate, the

modification of school curricula, or the confrontation of crime."[15]

Der Benutzung des Begriffs des „bösartigen Problems" liegen im wissenschaftlichen Diskurs mitunter abweichende Definitionen zugrunde[16]. So ist mir an mehreren Stellen die Interpretation „Ein Problem ist bösartig, wenn der Versuch einer Lösung das Problem selbst verändert" begegnet. Diese Charakterisierung trifft jedoch nicht ausschließlich auf bösartige Probleme im oben dargestellten Sinn zu. Dies kann z.B. auch vom Schachspiel behauptet werden, das nach Rittels und Webbers Liste ohne Zweifel zahm ist. Diese 10–Punkte–Checkliste ist im strengeren Sinn weder eine Definition noch drückt sie den eigentlichen Kern ihres Inhaltes explizit aus. Ihre Essenz kann in dem folgenden Satz zusammengefaßt werden:

Ein „bösartiges Problem" (und in diesem Sinne ein Entwurfsproblem) ist eine Planungsaufgabe, deren erfolgreiche Bewältigung nicht anhand eines formalen Bezugsrahmens festgestellt werden kann.

Vor dem Hintergrund einer derartigen Komprimierbarkeit kann die Frage nach der Relevanz der Checkliste gestellt werden. Ihren eigentlichen Wert hat Rittels und Webbers Liste aus meiner Sicht in ihrer pragmatischen und letztlich didaktischen Dimension. Sie ist ein sehr mächtiges Werkzeug, wenn der Mangel an *Operationalisierbarkeit* von Design–Aufgaben und resultierende Probleme bei deren Bearbeitung mit Computern vermittelt werden sollen. Die anschauliche Bezeichnung als „bösartig" und „zahm" sowie zehn Charakteristika, mit denen jeder Entwerfer nur allzu vertraut ist, sind bei der Einführung in diese Problematik wesentlich verständlicher, hilfreicher und effektiver als eine kurze, abstrakte Definition. In diesem Sinne werde ich mich im weiteren Verlauf dieser Arbeit an Rittels und Webbers Terminologie halten.

Den Kriterien der obigen 10–Punkte–Checkliste (und daher auch der knapperen Definition) entspricht auch die Aufgabe des Lehrens. Der Entwurf von Unter-

[15] Rittel, Horst und Webber, Melvin; ebd., S. 160

[16] Dies habe ich insbesondere auf Konferenz–Besuchen im Bereich der Entwurfswissenschaften festgestellt.

richt kann beispielsweise seinen erfolgreichen Abschluß weder garantieren noch ohne praktische Umsetzung erkennbar machen. Bereits die Bestimmung relevanter Restriktionen und Handlungsspielräume für eine Unterrichtseinheit ist eine potentiell endlose Aufgabe, deren Abschluß von Lehrenden auf der Basis von Erfahrung und individuellem Urteilsvermögen erkannt werden muß — eine allgemein gültige Stoppregel existiert nicht. Unterricht kann auch nicht simuliert, nicht wiederholt und nur schwer verglichen werden. Auch läßt sich der Erfolg von Unterricht nicht an formal beschreibbaren Indikatoren festmachen[17]

Probleme haben in ihrem Ursprung zunächst immer einen bösartigen Charakter. Wir haben jedoch eine Reihe von Strategien (Regelwerke und Prozeduren) entwickelt, mit denen wir einen Teil der bösartigen Probleme zähmen können. Hierzu gehören die Mathematik, Spielregeln, Gesetze, Bauanleitungen und weitere auf Konventionen beruhende Regelwerke. Diese bilden Maßstäbe, durch die Bedingungen, Eigenschaften und Resultate von Problemen und ihren Lösungen prüfbar und formal vergleichbar werden. Genaugenommen ist ein gezähmtes Problem nicht mehr mit dem ursprünglichen bösartigen Problem identisch. Der Prozeß der Zähmung bzw. der Operationalisierung geht mit einer Abstraktion einher, durch die das zahme Resultat *generische* Eigenschaften erhält. Das heißt, es beschreibt nicht mehr ein konkretes Problem sondern generell gültige Kriterien für das Lösen einer kompletten Klasse von Problemen.

Das Problem der Entscheidung, ob die durch ein studentisches Referat erbrachte Leistung als „ausreichend“ bewertet werden kann, ist z.B. überaus bösartig, denn praktisch sind keine zwei Referate wirklich gleich. Dieses Problem kann gezähmt werden, indem formal überprüfbare Kriterien eingeführt werden (z.B. „wurde den Zuhörern die Möglichkeit gegeben, Fragen zu stellen?“, „wurde vorgelesen oder frei vorgetragen?“). Dieses Bewertungssystem ist nun nicht mehr nur auf ein besonderes Referat sondern (zumindest innerhalb desselben Kontextes) auf jedes Referat (also die gesamte Problem–Klasse) anwendbar.

[17] Es sei denn, das jeweilige Unterrichtsziel wurde durch eine explizite Problemzähmung operationalisiert oder das jeweilige Lernziel ist sehr eng an Lehrinhalt gekoppelt, die an sich eine zahme Natur haben, so daß es leicht möglich ist, Verhaltensweisen zu definieren, die einen erfolgreichen Lernprozeß belegen.

Aufgaben, die Rittel und Webber als *gezähmte Probleme* klassifizieren, werden von Didaktikern als *operationalisierte Lernziele* bezeichnet. Rittels und Webbers *bösartige Probleme* bezeichnet der Instruktions–Theoretiker Robert F. Mager als *Fuzzies*, was in seinem Sinn etwa soviel wie „unscharf definierte Probleme“ bedeutet.

Insbesondere Lernzielorientierte Ansätze wie der Programmierte Unterricht haben einen starken Bedarf nach Überprüfbarkeit von Lehrwirksamkeit und damit eine Diskussion über die Formulierung entsprechender Bewertungsmaßstäbe nach sich gezogen. Mager schlägt hier eine Strategie zur Lernzielanalyse vor, die es Lehrern gestatten soll, „Fuzzies“ zu erkennen und in operationalisierte Ziele umzusetzen, deren Erreichen formal prüfbar ist. Die Schwierigkeiten, die Mager im Umgang mit Problemen im (Bildungs–)Alltag identifiziert, entsprechen weitgehend den Beobachtungen Rittels und Webbers.

> „It seems pretty obvious that if your goal is to improve students' understanding of history, you don't proceed to make them expert buggy whip makers. Nor would you instruct them in welding or weaving. But what *would* you do? Maybe instructing isn't even the right approach. Maybe some other action is indicated. Or maybe *no* action at all. How can you decide how to proceed *until* you know what 'understanding history' means?
>
> To take other examples, how should you proceed if the goal is to make 'better citizens'? What should you do if the goal is to achieve 'good judgement,' 'perceptive listening,' 'motivated workers,' or 'effective therapists'? Though these states may be among the most important to achieve — and all goals *sound* important — *the act of stating them does little to suggest the means of their achievement.*“[18]

Mager beschreibt Lernziel–Operationalisierung als generelle Antwort auf „Fuzzies“, was einer generellen Empfehlung zur Zähmung aller bösartigen Probleme im Unterricht gleichkommt. Seiner Anleitung zufolge sollen operationalisierte

[18] Mager, Robert F.: [32], S. 6 ff.

Ziele als formale Bezugsrahmen und Verhaltensweisen als Kriterien für das Erreichen dieser Ziele definiert werden. Wenn diese Verhaltensweisen schließlich beobachtet werden können, sei das jeweilige Ziel erreicht.

Aus didaktischer Sicht haben Operationalisierungen von Lernzielen Vorteile und Nachteile. Die ntstehende *objektive Vergleichbarkeit* trägt sicherlich zu Fairness und Gleichbehandlung bei und erleichtert die Beurteilung von Lernerleistungen. Andererseits können sie ihr Versprechen allgemeiner Anwendbarkeit nicht halten, weil es prozessorientierte Unterrichtsmodelle gibt, die auf der bewußten Ausblendung von Zieldefinitionen basieren (z.B. Gesprächsrunden oder Workshops). Da zahme Kriterien zur Leistungsbewertung immer auf Abstraktionen beruhen, sind sie auch wenig spezifisch, wodurch nicht erwartete und individuell herausragende Leistungen ggf. nicht ausreichend berücksichtigt werden. Ferner sind Lernziel–Operationalisierungen für Lehrende beim Unterrichtsentwurf wenig hilfreich, da die Definition eines Ziels noch nichts über die Handlungen aussagt, die zur Erreichung des Ziels führen. Angesichts der Dynamik von Unterricht, seines Verhaltens entlang der Zeitachse und der ständigen Überwachung von Zielen und Strategien durch seine Entwerfer (Lehrer) erscheint Magers Ansatz auch als zu statisch. Die Beobachtung beliebiger realer Unterrichtsverläufe zeigt allzu oft, daß zunächst definierte Ziele auf halbem Weg revidiert und modifiziert werden und daß der Weg zwischen einem didaktischen Ausgangspunkt und einer didaktischen Zielsetzung selten eine gerade Linie ist, sondern ein verschlungener Kurs, dessen Auffindung und Einhaltung einen erfahrenen Navigator braucht.

Besonders interessant erscheint mir hier Magers Hinweis darauf, wie einfach es oft ist, Strategien zu nennen, die zur Erreichung von Zielen *nicht* angemessen sind, während es sehr schwer ist, angemessene Strategien *für* die Erreichung von Zielen zu identifizieren. Auf die Natur negativer und positiver Sätze gehe ich später genauer ein. Ich nehme diese Beobachtung bereits hier zum Anlaß, um kurz auf die Schwierigkeiten einzugehen, die bösartige Probleme dem Computer bereiten.

Der Computer kann a priori nur zahme Probleme lösen, weil die Symbole, die Mathematik und die Regeln der Logik, die er verarbeiten kann, künstliche Pro-

dukte menschlicher Kulturen und letztlich die essentiellen Werkzeuge der Problemzähmung sind. So können wir gegen den Computer Schach spielen, weil wir Wege gefunden haben, bestimmte Weisen des Schachspielens in algorithmischen Prozeduren zu beschreiben, die Computer befolgen können. Es macht aber wenig Sinn, ihn mit der Überarbeitung eines Curriculums für den Politik–Unterricht zu beauftragen, da wir nicht jedes Problem zähmen können. Dies hat zwei Gründe:

Zum einen fehlt dem Computer bislang ausreichendes Wissen über die natürliche Welt, um sinnvoll komplexere Probleme (etwa sozialer Natur) zu bearbeiten. Wenn es uns gelungen ist, ein Problem zu zähmen, es also in logische Operationen und mathematische Prozesse zu zerlegen und als solche zu beschreiben, und, sofern dies notwendig ist, seine Variablen zu erfassen und zu digitalisieren, dann ist es gezähmt, und es ist durchaus wahrscheinlich, daß es vom Computer bearbeitet werden kann. Dies gilt aber nur soweit, wie der Computer den jeweiligen Weltausschnitt kennt: Um rational zu agieren (oder zu entwerfen) muß der Computer alle für das jeweilige Problem relevanten Variablen kennen.

Zum anderen kann eine Aufgabe wie die Überarbeitung eines Curriculums nur dann als erfolgreich gelöst bezeichnet werden, wenn diese sich praktisch bewährt hat. Es ist also ein Ausprobieren der Lösung erforderlich. Genau dieses Modell wird vom Computer angewendet, wenn er Schach spielt: Für jeden Zug generiert er auf der Basis der gegenwärtigen Konstellation alle Kombinationen weiterer möglicher Spielverläufe und bewertet diese kummulativ. Mit jedem weiteren simulierten Zug wächst dabei das hierarchische Modell in Erwägung gezogener zukünftiger Konstellationen, der sogenannte „Spielbaum“. Mit jeder Ebene des Spielbaums wächst der erforderliche Rechenaufwand exponentiell. Die Qualität getesteter Züge läßt sich im Fall des Schachspiels leicht bewerten, da alle möglichen Verläufe des Spiels der definierten Problemzähmung — den Schachregeln — entspechen müssen. Züge und deren Kombinationen können in Erwägung gezogen und ausgewertet werden, ohne tatsächlich durchgeführt zu werden. Bösartige Probleme unterscheiden sich hiervon insofern, daß sie praktisches Ausprobieren, also die reale Anwendung erfordern.

Würde einem Computer das zum Curriculum–Entwurf erforderliche riesige

Weltmodell zur Verfügung stehen, würde er weiterhin auch die nicht gegebene Möglichkeit zum theoretischen Ausprobieren verschiedener Strategien benötigen, um bessere von schlechteren Lösungen unterscheiden zu können. Doch der Computer verfügt weder über das erforderliche Weltmodell, noch über Mittel, mit denen darin Sets varianter, zu testender Hypothesen erzeugt werden könnten.

Menschen ziehen ebenfalls unterschiedliche Strategien in Betracht, bevor sie sich für eine von ihnen entscheiden. Doch sie verfügen über umfassende Weltmodelle und intuitive Fähigkeiten, mit denen sie variante Lösungs–Sets erzeugen und aus ihnen auswählen können.

Interessanterweise „sehen“ wir Menschen im Gegensatz zum Computer nur eine kleine Menge tendenziell sinnvoller Lösungen. So schließen wir aufgrund unseres Wissens (Weltmodelle) wenig erfolgversprechende Strategien bewußt aus und ziehen viele tendenziell schlechtere Lösungen gar nicht in Erwägung, ohne dies überhaupt zu wahrzunehmen. Dies ist beim Schach spielen und beim Entwerfen eines Cucciculum oft der Fall. Hierbei gelingt es uns, riesige Mengen uninteressanter Strategien einfach auszublenden, während der Computer ohne einen solchen Filter jede einzelne Strategie abwägen muß[19]. Wir verdanken intelligentes und effizientes Denken und Handeln folglich oftmals dem, was wir *nicht* sehen (bzw. was wir unbewußt übersehen) als dem, was wir sehen. Strategien dieser Art widersprechen denen, die wir als Verhalten von Computern in Software modellieren. Unsere Programme (z.B. Computerschach) evaluieren typischerweise die Qualität systematisch generierter positiver Sätze. Die zu diesem Zweck erforderlichen Bewertungsmaßstäbe liegen uns aber nur im Fall zahmer Probleme in Form von Kriterien vor, die trivial genug sind, daß wir sie in Software abbilden können. Unsere Computer und Software versagen bislang vor bösartigen Problemen. Sie sind nicht einmal in der Lage, die einfacheren (aber weitaus zahlreicheren) negativen Sätze vom Typ “Strategie x erscheint wenig geeignet, um zur Verbesserung der Qualität eines Curriculums beizutragen“ zu formulieren.

[19] Optimierte Algorithmen versuchen ebenfalls, unnötige Berechnungen zu vermeiden. Diese Ausnahmebehandlungen müssen jedoch rechteitig explizit programmiert werden.

2.5. Rollenverteilungen und Paradigmenwechsel

Einführungen neuer Technologien werden typischerweise von hohen Erwartungen begleitet, die sowohl als Hoffnungen der Nutzer als auch als Versprechungen der Entwickler daherkommen können. Die Veränderungen, die diese Technologien hervorrufen, sind dann nicht selten weniger spektakulär als erwartet: So ersetzen neue Produkte ihre Vorgänger oft nicht wie erwartet, sondern ergänzen diese. Dies kann in positiven Fällen zur Erweiterung von Medienangeboten und sogar zu Demokratisierungsprozessen führen, in negativen Fällen führt es zu inkompatiblen technischen Standards, technologischen Sackgassen oder sogar zu Monopolbildungen. In jedem Fall prägt das Erscheinen neuer Technologien unser Handeln, unser Denken und sogar unser Selbstverständnis und bewirkt mitunter Veränderungen, die weitaus subtiler sind als zunächst erwartet wurde. Der Einzug von Informationtechnologie in den Unterricht ist in dieser Hinsicht keine Ausnahme. Da die *Revolution des Lernens* noch nicht stattgefunden hat, wird sich erst in der Zukunft zeigen, welche Effekte computervermittelter Unterricht auf unser Handeln, Denken, unsere Technologien und auf unsere tägliche didaktische Arbeit haben wird. In diesem Abschnitt diskutiere ich wahrscheinliche bzw. mögliche Effekte von computervermitteltem Unterricht im allgemeinen und vom Einsatz von Autorensystemen im besonderen, die zum Teil bereits eingetreten sind und erst bei genauerem Hinsehen erkennbar werden oder in Zukunft möglicherweise noch eintreten werden.

„Hypertexte und –medien können den Leser von der Diktatur des Leseflusses befreien, die Autoren in linearen Texten ausüben.“ Mit dieser und ähnlichen Erwartungen begegnete man in den späten 80er und frühen 90er Jahren dem neuen Massenmedium Hypertext. Zunächst durch die Verfügbarkeit von Autorensystemen wie HyperCard, das ab 1987 gratis mit jedem Apple Macintosh ausgeliefert wurde, und später durch die Entwicklung von HTTP und WWW bestärkt, blühte die Hypertextforschung auf. 1990 schrieb Jakob Nielsen:

> „Hypertext basically destroys the authority of the author to determine how readers should be introduced to a topic. From the reader's perspective, this is one of the great advantages of hypertext since

> it means that they are free to explore the information as they see it.“[20]

Eines der häufigsten Argumente für den computervermittelten Unterricht, insbesondere für das Lernen mit Hypertext, ist der erhöhte Aktivitätsgrad des Lerners, der bei der aktiven und selbstbestimmten Navigation durch seine Lerninhalte eine höhrere intrinsische Motivation aufweist und somit potentiell besser lernt. Derartige lernpsychologische Aussagen über Lehrwirksamkeit können nicht wissenschaftlich bewiesen werden. Daher werden sie zunehmend als „pädagogische Mythen“[21] ohne verläßlichen Aussagewert bezeichnet und behandelt. Während Nielsen den von der Autorität des Autoren befreienden Effekt von Hypermedia begrüßt, muß die Wirkung dieses Effekts in *Unterrichtssituationen* daher etwas differenzierter betrachtet werden. Leider ist jedoch nicht alles Lernen primär intrinsisch motiviert. Die Definition von Unterricht in Abschnitt 2.2 umfaßt *zwei* am institutionalisierten Unterricht beteiligte Parteien: Lehrende und Lernende. Unterricht erfolgt nämlich aus gutem Grund nicht in Isolation. Er wird inhaltlich und methodisch kontinuierlich von Lehrenden (bzw. „Autoren“) begleitet und angepaßt. Lehrende motivieren Lerner und leiten sie an, um sie dort zu unterstützen, wo sie aus eigener Motivation und durch eigene Disziplin und Anleitung nicht oder nicht effektiv genug lernen würden. Wir setzen Lehrer ein, um Unterricht mit professionell ausgewählten und aufbereiteten Inhalten und Methoden anzubieten, und dabei an sich dynamisch verändernde Unterrichtssituationen anzupassen, denn wir gehen davon aus, daß die in unserer Gesellschaft angestrebten Bildungsziele durch ausschließlich selbstgesteuertes Lernen nicht umfassend erreicht werden können. Wäre dies der Fall, könnten wir schließlich auf Schulen und Lehrer verzichten.

Die These, daß die *Autorität des Autoren* von Hypermedien untergraben wird, stützt sich auf die Annahme, daß traditioneller (gedruckter) Text linear und Hypertext nicht– oder besser: multilinear ist und die jeweilige technische Präsentationsform bestimmte Rezeptionsmuster erzwingt, erschließt oder verbietet. Diese Annahmen sind letztlich nicht haltbar, da Hypertext selbstverständlich auch

[20] Nielsen, Jakob: [39], S. 179

[21] Schulmeister, Rolf: [55], S. 245

linear aufgebaut werden kann, weil gedruckter Text durchaus nichtlinear strukturiert werden kann und in beiden Fällen letztlich immer der Leser entscheidet, wie er den unmittelbaren Lesefluß sequenziert. Lehrende „Autoren“ können bei der Herstellung ihrer Produkte bestimmte Rezeptionsformen wahrscheinlicher und andere unwahrscheinlicher machen. Dabei sind sie aber auf eine hohe Flexibilität ihres jeweiligen Werkzeugs, also des Autorensystems, angewiesen. Da Autorensysteme sich jedoch meist nicht sehr kooperativ gegenüber formalen Eingriffen durch ihre Anwender verhalten, weicht die *Autorität des Autoren* praktisch einer *Autorität des Autorensystementwicklers*. Dieser besitzt vor dem Hintergrund des heute verbreiteten Verständnisses von Autorensystemdesign jedoch keine Kenntnis der Unterrichtlichen Situationen, für die er Werkzeuge herstellt (vgl. Ablaufschema in Abb. 2.4).

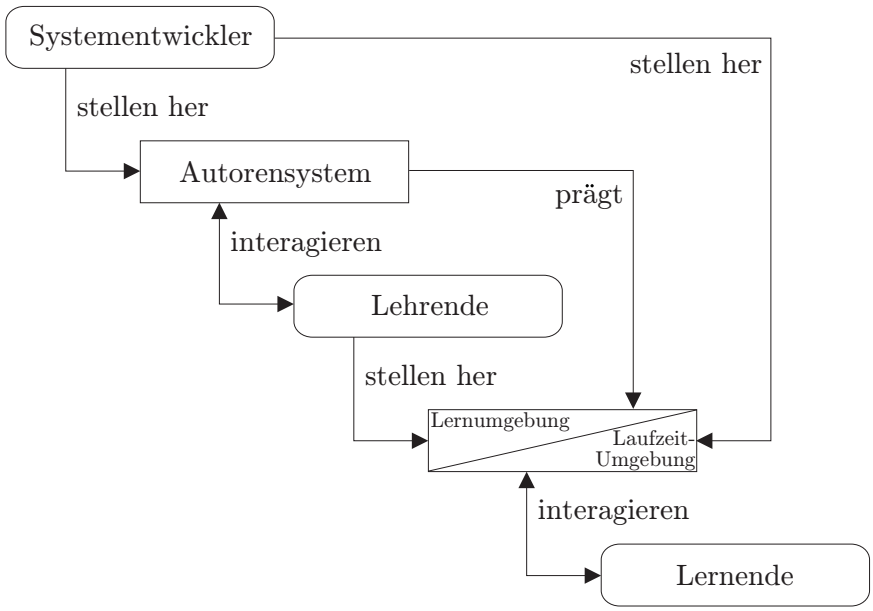


Abbildung 2.4.: Verbreitetes Verständnis des Ablaufs vom Autorensystemdesign bis zum Unterricht

Tatsächlich zeichnen sich heutige Autorensysteme durch inhaltliche Offenheit und gleichzeitige formale Abgeschlossenheit aus. Didaktische Entscheidungen werden in Form von Wizards, Formatvorlagen etc. von Softwareherstellern außerhalb konkreter Unterrichtssituationen getroffen. Während didaktisch–

methodische Entscheidungen also im traditionellen Klassenunterricht von Lehrern vor Ort (dezentral) getroffen werden, erfolgt im Fall von Autorensystemen eine zentralisierte methodische Vorauswahl und letztendlich eine Kanonisierung. Wenn durch Autorensysteme die „Autorität des Autoren“ in Frage gestellt wird, dann also allenfalls zugunsten örtlich, zeitlich und fachlich entfernter Entwickler und nicht zugunsten des Lerners. Neil Postman erinnert daran, daß wir bereits wiederholt dem Trugschluß unterlegen sind, daß auf zentralisierten didaktischen Planungsmodellen basierende und über Massenmedien verbreitete Bildung die Lehre grundsätzlich revolutionieren, effektivieren und schließlich „verbessern“ könnten.

> „I do not go as far back as the introduction of the radio and the Victrola, but I am old enough to remember when 16–millimeter film was to be the sure cure, then closed–circuit television, then 8–millimeter film, then teacherproof textbooks. Now computers.“[22]

Während Lehrer Unterricht traditionellerweise sowohl vorbereiten als auch die Interaktion mit Lernern durchführen, verschiebt sich der Schwerpunkt ihrer Tätigkeit und Verantwortung im (noch immer seltenen) computervermittelten Unterricht in Richting vorbereitender (eher inhaltlich als formal orientierter) Tätigkeiten[23]. Die Interaktion Lernender mit ihren Inhalten und die Bearbeitung ihrer Aufgaben erfolgt — insbesondere im Fern– oder netzwerkbasierten Unterricht — in Abwesenheit Lehrender. Dies wird von Lehrenden in aller Regel nicht als negativer Nebeneffekt des Mediums verstanden sondern insbesondere in der universitären Lehre genutzt um die Fähigkeiten der Lerner zu selbständigem Arbeiten zu fördern. Im Rahmen computervermittelten Unterrichts dieser Art interagieren Lehrende in vergleichsweise geringerem Umfang direkt mit Lernern und nehmen verstärkt die Rolle von Planern bzw. Entwerfern von Interaktion ein. In dem Maß, in dem der Computer die Interaktion mit Lernenden

[22] Postman, Neil: [46], S. 50

[23] Es ist durchaus denkbar, daß *Virtuelle Realität* in Zukunft computervermittelten Unterricht mit Echtzeit–interaktion ermöglichen wird, in der das Verhältnis von vorbereitenden Tätigkeiten zur Unterrichtsdurchführung mit dem des traditionellen Unterrichts vergleichbar sein kann. Dies ist zur Zeit allerdings nur eine Zukunftsvision.

übernimmt, verschieben sich die Aufgaben Lehrender in Richtung Curriculum–Entwurf: Sie legen fest, was unterrichtet wird und wie, führen diesen Unterricht jedoch nicht mehr wie im traditionellen Unterricht direkt oder zumindest nicht mehr ausschließlich direkt durch. Diese Verschiebung hat mehrere Konsequenzen: Zum einen verkürzen sich tendenziell die Zeiteinheiten, für die Curricula entworfen werden: Statt den Lehrplan halbjahresweise zu entwerfen, wie es für traditionellen Unterricht üblich ist, liegt der Fokus vorbereitender Lehrer auf aktuellen Unterrichtseinheiten in Größenordnungen von Tagen oder Wochen.

Andere Tätigkeitsbereiche und Berufsbilder zeigen, daß der „Einzug des Computers“ eine unumkehrbare Entscheidung ist. Sobald Arbeitsprozesse erst einmal vom Rechner unterstützt werden, ist es meist nicht mehr möglich, zu alten, nicht rechnergestützten Methoden zurückzukehren. Die Kenntnisse über traditionelle Verfahren zur Bewältigung der jeweiligen Abläufe schwinden sehr schnell und innerhalb weniger Monate oder Jahre ist der Computer „nicht mehr wegzudenken“. Daraus resultiert für die meisten Tätigkeitsbereiche eine Situation, in der Rechner entweder eingesetzt werden oder nicht. Lösungen dazwischen oder sogar ein Pendeln zwischen Computereinsatz und Arbeit ohne Computer gibt es hier nicht. Im Unterricht sieht dies wieder einmal anders aus und einmal mehr ergibt sich aus dieser Sonderstellung eine Mehrbelastung für die Lehrenden: Um die didaktische Methodenvielfalt zu sichern, ist Computereinsatz im Unterricht nur eine Option und kein Verfahren, das eingeführt wird, Alternativen verdrängt und fortan nicht mehr wegzudenken ist (Ausnahme sind die Unterrichtssituationen, die nur computerbasiert möglich sind, etwa Fernstudiengänge via Internet). Lehrer müssen daher in der Lage sein, Unterricht mit und ohne Computer bereitszustellen, die Entscheidung zwischen traditioneller und computervermittelter Lehre zu treffen und zu verantworten. Sie dürfen das Unterrichten ohne Computer nicht verlernen, wenn sie das Lehren mit Computern erlernen.

Weiterhin verschiebt sich die Rolle traditioneller Curriculum–Entwerfer hin zum Planungen auf einer technischen Meta–Ebene: Wieviel Hardware brauchen Schulen, welche Vernetzung, welche Software, wie viele Lehrende mit welchen technischen Qualifikationen? Bildungs–Planer lenken für den traditionellen Un-

terricht die Auswahl technischer Hilfsmittel und Werkzeuge und beginnen mehr und mehr, diese Auswahl auch für den computervermittelten Unterricht zu koordinieren. So entschlossen sich international zahlreiche Universitäten, WebCT für ihre gesamten Lehrangebote zu lizensieren (wovon allerdings selten im erwarteten Ausmaß Gebrauch gemacht wird). Diese Koordination bringt potentiell Vorteile in bezug auf die Vertrautheit der zu verwendenden Werkzeuge und ihre Kompatibilität, aber auch Nachteile, weil eine Beschränkung der Werkzeugauswahl auch eine Beschneidung der Methodenvielfalt mit sich bringt.

Autorensysteme und ihre Funktionsweisen tragen im computervermittelten Unterricht mit ihrem nachhaltigen methodischen Einfluß teilweise die Verantwortung, die im traditionellen Unterricht den Lerntheorien zufällt. Autorensystementwickler sind somit letztlich in der Verantwortung didaktischer Theoretiker und methodischer Entwerfer. Die mangelnde Akzeptanz, mit dem Lehrende Autorensystemen begegnen, kann sicherlich auch damit begründet werden, daß die Softwarehäuser, die Autorensysteme herstellen, sich ihrer theoretischen Rolle möglicherweise nur in begrenztem Maß bewußt bzw. für sie qualifiziert sind oder dieser Verantwortung sogar bewußt ausweichen.

Die Lösung entwerferischer bzw. unterrichtlicher Probleme erfordert Erfahrung, Kenntnisse über die Qualität alternastiver Lösungsansätze und gesellschaftlicher Diskurse. Da Gesellschaften und Individuen im Laufe der Zeit ihre Weltsichten modifizieren und revidieren, unterliegt auch unser Modell vom Entwerfen epochalen Schwankungen. William Mitchell identifiziert in der Designwissenschaft für den Zeitraum seit Mitte des 20. Jahrhunderts drei grundlegend verschiedene aufeinanderfolgende Paradigmen, auf denen unser Entwurfsbegriff aufbaut[24]:

- Entwerfen als Problemlösen (ab ca. 1960)
- Entwerfen als wissensbasierter Prozeß (ab ca. 1980)
- Entwerfen als sozialer Prozeß (ab ca. 1990)

[24] Mitchell, William J.: [38]

Indirekte Auswirkungen dieser Paradigmenwechsel finden sich über Design und Architektur hinaus auch in Paradigmen in der Lehre und mitunter in den Metaphern, auf denen im Laufe der Jahrzehnte Lehrsoftware aufbaute. Jüngere Methoden des Unterrichts und der Unterrichtsvorbereitung, in der Lehrende nicht mehr isoliert sondern in Gruppen unterrichten, wie z.B. team teaching, belegen eine analoge Tendenz in der Praxis des Unterrichts. Gleichzeitig verlagert sich unser Verständnis von Lernen vom *Wissenstransfer* hin zur *moderierten Umwelt–Exploration.* Somit verlagert sich auch unser Verständnis der Rolle Lernender von der *beplanter Empfänger von Bildung* zu der *aktiver und reflektierender Entwickler eigener Kompetenzen.*

Die aufgezeigten Paradigmenwechsel und die resultierenden Rollenverschiebungen entlarven den Begriff *Autorensystem* als veraltet und methodisch obsolet. Wir verstehen Unterricht heute nicht mehr als einseitig gerichteten Informationsfluß vom Lehrer zum Lerner und wir bemühen uns, ihn nicht als solchen zu praktizieren. Unterrichtsinhalte werden nicht nur von einem Lehrer, der die Rolle eines Autors einnimmt, hergestellt und präsentiert. Im Vordergrund steht zunehmend das gemeinsame Erarbeiten von Inhalten, Erkenntnissen, Fähigkeiten, wobei die traditionellen Rollen von Lehrern und Lernern tendenziell verschwimmen. So ist es angesichts riesiger verfügbarer Wissens– und Materialmengen nicht unüblich, daß Lehrer in ihrem Unterricht nicht fortfahren können, bis sie selbst „ihre Hausaufgaben“ gemacht und ausreichend gelernt haben oder daß *Lernen durch Lehren* praktiziert wird, indem sich Schüler in die Rolle des Lehrers begeben, um Material zu präsentieren. Doch bis heute unterstützen Autorensysteme im wesentlichen nur die traditionelle Form der Unterrichtsvorbereitung und –durchführung, die den Lehrer nur als Erzeuger bzw. Bereitsteller von Unterrichtsmitteln und den Lerner nur als Rezipienten dieser Materialien kennen. Auch das wachsende Bedürfnis nach koordiniertem Unterricht von mehr als einer Lehrperson oder nach der isolierten Wiederverwendung bewährter Inhalte und Methoden durch Anwendung auf mehrere Lernergruppen, finden in diesen Programmen praktisch keine Unterstützung.

Selbst hoch entwickelte Produkte wie der Macromedia Director basieren durchgehend auf Single–User Authoring und einer klaren Trennung zwischen Auto-

ren und Rezipienten erstellter Inhalte, wobei Information nur vom Lehrer zum Lerner fließen kann. Diese Kritik läßt sich auf viele weitere Autorenwerkzeuge wie Toolbook, Question Mark, Metacard usw. übertragen. Auch hier gilt, daß Interaktionsformen, die von Software nicht unterstützt werden, praktisch nicht, nur schwer oder auf Umwegen durchführbar sind und alternative Modelle des Informationsflusses in Lernergruppen auf störende und unproduktive Weise blockiert werden. Schüler erhalten von neueren und netzwerkfähigen Systemen wie WebCT die Möglichkeit, Ressourcen zu annotieren. Möglichkeiten, aus Lernerperspektive Lernmaterial beizutragen oder zu warten, gegenseitig Lernleistung einsehbar zu machen oder zu bewerten oder sonstige Interaktionsformen offener Curricula werden (abgesehen von obligatorischen Online–Chats und Bulletin–Boards) ebenfalls nicht unterstützt.

2.6. Systeme, ihre Oberflächen und Paradigmen

Interface metaphors have limited usefulness.
What you gain now you may have to pay for later.[25]

Das Ziel und die Motivation der technischen Entwicklung von Autorenwerkzeugen von Programmiersprachen hin zu visuell orientierten Autorensystemen war deren vereinfachte, nutzerfreundliche Handhabung. Von zentraler Bedeutung in dieser andauernden Entwicklung ist die Befreiung Lehrender von textueller Programmierung, die als Argument sehr populär ist, aber nur in wenigen Fällen wirklich erreicht wurde. Die schwer zu bedienende *Universalmaschine Computer* soll durch Autorenwerkzeuge in eine problemorientierte Maschine verwandelt werden, die dafür jedoch (wie bereits diskutiert) einen erheblichen Teil ihrer Flexibilität einbüßen muß. Der eigentliche Nutzen, der hierbei entsteht, ist die Vereinfachung der Nutzerschnittstelle zwischen Lehrer und Computer. Die Strategie der Schnittstellen–Vereinfachung läßt sich wie folgt beschreiben: Der in arithmetisch–logischer Hinsicht universelle Kern des Rechners wird in einer Hülle verpackt, die eine begrenzte Auswahl von Bedienelementen aufweist.

[25] Laurel, Brenda: [30], S. 131

Diese Bedienelemente werden mit einer begrenzten Anzahl definierter Prozeduren innerhalb der Software verbunden. Während diese Prozeduren durch die nutzerfreundliche Oberfläche sehr leicht abrufbar sind, stehen nicht vorhergesehene bzw. beim Interfacedesign nicht berücksichtigte Prozeduren dem Anwender (in diesem Fall dem Lehrer) nicht zur Verfügung. Eine Ausnahme bilden scripting–fähige graphische Oberflächen, die jedoch durch eben diese Fähigkeit der Idee rein graphischer Bedienbarkeit widersprechen und das Versprechen graphischer Schnittstellen, rein visuell programmierbar zu sein, nicht halten können. Hinzu kommt das Problem, daß innerhalb einer scripting–fähigen Umgebung nur Elemente innerhalb dieser Umgebung direkt und auf technisch kohärente Weise ansprechbar sind (das sogenannte „Sandbox–Problem“). Zugang zu Elementen außerhalb der jeweiligen Umgebung ist mitunter zwar dank der Bereitstellung entsprechender Schnittstellen möglich, die Herstellung dieser externen Programm–Elemente erfordert ihrerseits aber weiterhin die Kenntnis mindestens einer (weiteren) Programmiersprache. Ein Beispiel hierfür ist das Autorensystem Toolbook Instructor, das Scripting–Sprache *OpenScript* HTML und Java für WWW–basierte Unterrichtsmaterialien herstellt. Diese sind allerdings nicht in der Lage auf der Lerner–Seite mit Software ausserhalb des WWW–Browsers zu kommunizieren (einzige Ausnahme ist die Bereitstellung sogenannter PlugIns, die ihrerseits in C++ oder VisualBasic hergestellt werden müssen. Ältere Versionen von Toolbook, die direkt in einer VisualBasic–Umgebung ausgeführt werden, können Scipte enthalten, die z.B. externe DLLs nutzen können, die in VisualBasic, C oder C++ entwickelt wurden.).

Eine wichtige gestalterische Aufgabe für das Autorensystemdesign ist die Gestaltung der „Hülle“, in der die komplexen Innenleben von Computer und Software verpackt werden und durch die deren Bedienung vereinfacht werden soll: Das sogenannte *Interface–Design*. Hier wird (nicht nur im Fall von Autorensystem–Design) die Methode des *expliziten konzeptionellen Designs* angewendet. Durch Referenzen auf bekannte, z.B. einfache alltägliche Konzepte (die bevorzugt selbst Schnittstellen zwischen Systemen darstellen) werden auf der Nutzerseite existierende kognitive Modelle genutzt, um die Interaktion mit dem Produkt zu erleichtern. Die Schnittstelle zum jeweiligen System soll durch

paradigmatische Anspielung auf vertraute Konzepte möglichst von der technischen Natur der Software abweichen, um sie *menschlicher*, *intuitiver*, *erlernbarer* etc. zu gestalten.

Systeme, die ihre komplexen inneren Strukturen verbergen, und sich ihren Nutzern durch leicht zu bedienende Oberflächen erschließen, sind uns aus dem alltäglichen Leben bekannt. Um zum Beispiel einen Brief zu verschicken, muß lediglich die Funktionsweise eines Briefkastens und nicht die Logistik und Arbeitsweise der Post begriffen werden. Strom „aus der Steckdose“ wird konsumiert, ohne daß irgendein Wissen über seine Herstellung dafür notwendig ist. Auch der menschliche Körper kann als Design verstanden werden, dessen Oberfläche viele interne Systeme und Prozesse nutzbar macht, ihre Komplexität jedoch verbirgt. Um uns zu ernähren, brauchen wir nur zu essen und müssen unser Verdauungssystem nicht verstehen. Die tatsächliche Nutzerfreundlichkeit, die diesen Beispielen zu eigen ist, beweist die Mächtigkeit dieses Verfahrens. Dies gilt jedoch nur für solche Systeme, deren „Inneres“ nur eine endliche Anzahl von Prozeduren und Zuständen kennt. Daher funktioniert dieses Verfahren bei der Anwendung von Computern besonders gut dort, wo durch Operationalisierungen der jeweiligen Problemlösungen nur eine definierte Menge von Verhaltensweisen von ihm verlangt werden.

Solange sich die Dienste des Computers auf eine begrenzte Anzahl einfacher Operationen beschränken, kann die Verwendung simplifizierter Nutzerschnittstellen eine sehr nützliche Erleichterung für den Abruf wiederkehrender Prozeduren ermöglichen, denn jede der bekannten Verhaltensweisen im Inneren des Systems kann an eines der endlich vielen Bedienelemente der Oberfläche genknüpft werden und ist somit leicht abrufbar. Die übliche Bedienung eines Fahrstuhls über ein Interface aus wenigen Tasten ist hierfür ein gutes Beispiel. Nutzer müssen sich keine Gedanken darüber machen, wie der Rechner hinter der Oberfläche des Fahrstuhls dessen Fahrstrecken optimiert oder welchen Regeln die Koordination der Abarbeitung mehrerer gleichzeitiger Anweisungen folgt. Das Erteilen der Befehle an den Rechner durch den Nutzer ist dennoch denkbar einfach. Die Symbole auf den Tasten sind Nummern, die jeweils einem Stockwerk zugeordnet sind sowie einige ikonische Symbole, die relativ intui-

tiv erlernbar, nicht–sprachlich und praktisch international standardisiert sind. Wenn man die Benutzung eines Fahrstuhls erlernt hat, wird man das Erlernte in anderen Fahrstühlen wieder anwenden können. Oberflächen dieser Art erschließen somit zum einen den Zugang zu einer begrenzten Anzahl von Funktionen im Inneren ihres eigenen Kerns und vermitteln zum anderen Kenntnisse über die Nutzung einer ganzen *Systemklasse*.

Wir benutzen den computergesteuerten Fahrstuhl *als Fahrstuhl* und als nichts anderes. Seine Funktionen sind im Vergleich zu vielen anderen Maschinen recht einfach, und daß er im Hintergrund von einem Mikroprozessor gesteuert wird, der ein Programm ausführt, ist den meisten Fahrstuhl–Nutzern eigentlich egal. Die Verpackung komplexer Rechner hinter nutzerfreundlichen Interfaces wird aber auch dort praktiziert, wo wir den Computer in bestimmten Rollen mit vordefinierten Verhaltensweisen benutzen wollen wie etwa bei der Verwendung der graphischen Nutzerschnittstellen (GUIs) unserer PCs. Über derart gestaltete Schnittstellen begegnen wir Computern nicht in ihrer Rolle *als Computer* sondern in der Rolle eines Spiels, eines Aktenschranks oder einer Theaterbühne. Selbstverständlich bleibt ein Computer ein Computer, egal über welche Oberfläche er bedient wird. Daher wirken diese Gestaltungsmetaphern nicht immer besonders überzeugend und konsistent. Assembler und Compiler waren historisch frühe Schnittstellen zu den Operationen der CPUs, durch die Rechner in die Rolle von „Übersetzern“ für Quelltexte versetzt wurden, und heute verwenden wir selbstverständlich GUIs und erleben unsere Rechner typischerweise als „Schreibtischoberflächen“. Analog zu dieser Entwicklung haben sich textuelle Autorensprachen zu visuellen Autorensystemen entwickelt. Dabei handelt es sich um Hüllen, die so wie auch das Tastenfeld in einem Fahrstuhl das universelle Innere des Systems hinter einer begrenzten Anzahl (hoffentlich) intuitiv erlernbarer Bedienelemente verstecken. Die Anordnung und die Gestaltung dieser Bedienelemente orientieren sich dabei neben ergonomischen Gesichtspunkten oft an bekannten Vorbildern und Konventionen mit dem Ziel, die Verständlichkeit und Erlernbarkeit der Systeme zu erleichtern.

Doch was passiert, wenn ein System auf eine Weise genutzt werden soll, die der jeweilige Entwickler bei der Herstellung nicht beachtet oder mit dem Ziel

der „Vereinfachung“ bewußt ausgeschlossen hat? Was passiert, wenn es (um bei unserem Beispiel zu bleiben) absolut erforderlich ist, ein Gewicht zu transportieren, das die zulässige Höchstlast eines Fahrstuhls überschreitet und zu seiner automatischen Abschaltung führt? Ein Experte mag in dieser Situation in der Lage sein, die Belastbarkeit des Lifts zu erhöhen, Beschleunigungs– und Verzögerungsverhalten anzupassen usw. um dieses sehr individuelle Problem zu lösen. Dafür ist es jedoch notwendig, tief in das Innere des Systems vorzudringen und es dort zu manipulieren — die Schnittstelle für normale Nutzer gestattet dies nicht.

Lehrer sehen sich beim Einsatz von Autorensystemen mit Aufgaben dieser Art konfrontiert: Sie sind Experten für die Situation und die verwendeten Mittel und müssen Probleme auf individuelle Weise lösen. Die Bedienbarkeit des Werkzeugs Autorensystem bietet in den heute gängigen Formaten jedoch nicht mehr Flexibilität als das Tastenfeld eines Fahrstuhls. Selbst wenn die technische Expertise für Eingriffe in das Autorensystem–Innere vorhanden ist, bieten kompilierte Pakete ohne Quelltexte keine Möglichkeiten, solche individuellen Lösungen umzusetzen.

Dies ist, wie ich meine, aus didaktisch–gestalterischer Sicht ein schweres Manko. Ich möchte in diesem Zusammenhang jedoch nicht verschweigen, daß auch durchaus gegenteilig argumentiert werden kann. Es gibt schließlich sehr gute Gründe dafür, daß sich Fahrstühle bei Überlastung abschalten. Es ist für Ingenieure nicht sonderlich schwer zu definieren, unter welchen Bedingungen ein Fahrstuhl überlastet ist — dies ist ein völlig zahmes technisches Problem. Unterrichtliche Gestaltungsprobleme sind aber leider bösartig und so ist es Pädagogen leider nicht möglich, analog allgemeingültige Strategien für ihre Werkzeuge und Produkte zu definieren.

Dennoch gibt es Beispiele für derartige Ansätze. So hat sich nach dem Zweiten Weltkrieg in weiterführenden Bildungseinrichtungen der Vereinigten Staaten (insbesondere in den Teilen, die an der Entwicklung der Atombombe beteiligt waren) eine besonders elitäre Haltung entwickelt, die allgemeingültiges Wissen über richtigen und falschen Unterricht für sich proklamiert hat. Ein Resultat war das sogenannte „teacherproof packaging“ von Curricula und Lernmateria-

lien. Die Rolle des Lehrers war in diesem Zusammenhang die eines *Anwenders* von Unterrichtsmaterialien und nicht die eines *Experten* mit Qualifikation zur Herstellung von Unterrichtsmaterialen.

Wenn Lehrern diese Expertise zuerkannt wird (hierzu gibt es meiner Ansicht nach letztlich keine Alternative), wirken die schablonenartigen Designparadigmen von Autorensystemen trotz ihrer vordergründigen „Nutzerfreundlichkeit“ hinderlich. Vor dem Hintergrund des dargestellten Dilemmas zwischen „Nutzerrolle“ und „Expertenrolle“ des Lehrers scheint es mir daher angebracht, die Betonung der Wichtigkeit von „Nutzerfreundlichkeit“ im Fall von Autorensystemen grundsätlich in Frage zu stellen.

Ziele designparadimatischer Referenzen, also zentrale Verfahren oder Vorbilder, nach denen sich die Gestaltung der Nutzerschnittstelle eines Autorensystems im *Autoren–Modus* orientieren, werden als *authoring pradigms* oder auch *authoring metaphors* bezeichnet. Sie nutzen in der Regel mehr oder minder explizite Referenzen auf reale Objekte und Verfahren, die insbesondere das sogenannte „look&feel“, Interaktionsprozeduren, repräsentierte Werkzeuge und die Art der Darstellung und Anordnung der Inhalte im GUI des Autorensystems prägen.

Designparadigmen sind im Kontext des computervermittelten Unterrichts doppelt relevant, da sie sowohl die Schnittstellen von Autorensystemen als auch diejenigen der damit erzeugten Lernumgebungen betreffen. Lernumgebungen werden in den seltensten Fällen *als Computer* benutzt, sondern als Spiele, Textbücher oder als Simulationen realer Prozesse oder Systeme. Eine Lernumgebung für das Erlernen der Bedienung eines Fahrstuhls z.B. könnte visuell und funktionell einen realen Fahrstuhl simulieren. Der Fokus der Lerner–Interaktion liegt eher auf dem Paradigma, an das sich die Schnittstellengestaltung anlehnt, als auf den technischen Details hinter der Oberfläche, die den Dialog nicht stören sollen. Die simplifizierende Wirkung der Oberfläche wird in diesen Fällen, in denen Nutzer nur auf einen kleinen, endlichen Ausschnitt von internen Verhaltensweisen des Systems zugreifen können sollen, in der Regel positiv bewertet.

[26] Falk, Lorne et al. [12]

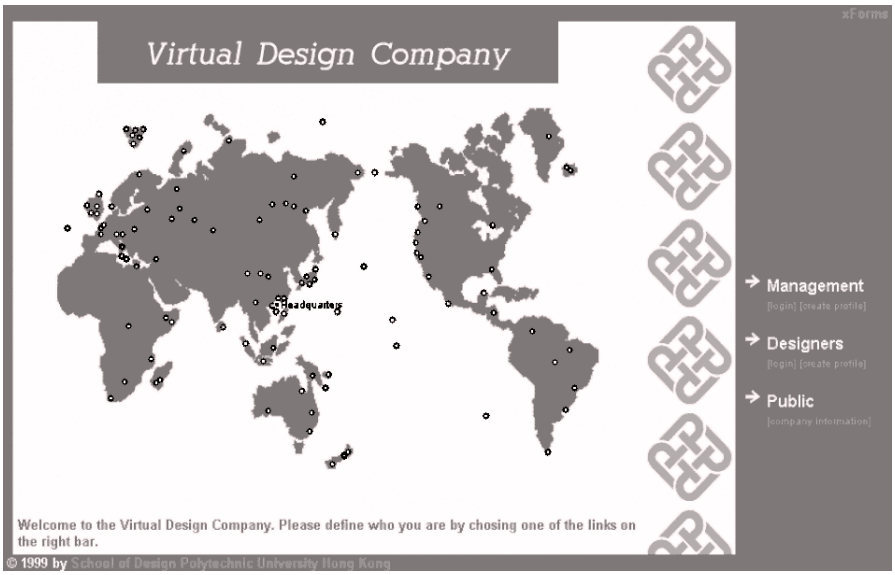


Abbildung 2.5.: Die Homepage der Virtual Design Company[26]

Abbildung 2.5 zeigt die Homepage einer Online–Lehrapplikation „The Virtual Design Company“ die 1999 an der School of Design der Hong Kong Polytechnic University eingesetzt wurde. Diese Software wurde mit zwei Zielen entwickelt: Zum einen sollte sie helfen, vernetztes weltweites Entwerfen zu lehren. Zum anderen wurde sie entworfen,um ein gleichzeitig entstehendes Problem zu lösen: Da Lehrer im computervermittelten Unterricht nicht wie im Klassenzimmer über die Möglichkeit verfügen, Lerneraktivitäten zu beobachten, droht ihnen der Verlust einer zur Leistungsbeurteilung zentralen Informationsquelle. Daher sammelt die Lernapplikation Informationen über die Interaktionen sämtlicher registrierter Nutzer und stellt diese zur Auswertung in einer dreidimensionalen Visualisierung einer Protokoll–Datenbank zur Verfügung (für eine detailierte Besprechung dieses Vorgehens siehe Abschnitt 4.7). Paradigma für die Gestaltung der Oberfläche der Applikation ist ein Design–Unternehmen, dem Gruppen von Designern (Studenten) jeweils als Agenturen angegliedert sind. Die Interaktion mit der Software erfolgt nach der Logik geschäftlicher bzw. kreativer Verhandlungen und Transaktionen. Die interne Funktionsweise aus technischer Perspektive spielt dabei keine Rolle. Die Beobachtung der Lernaktivität und des Lernfortschritts bleibt für die Lerner ebenfalls transparent.

Diese Lernumgebung wird *wie eine Webseite eines Unternehmens* und als nicht anders benutzt.

Im Gegensatz zu den genau definierten *Weltausschnitten*, die in solchen Lernumgebungen modelliert werden, sollen sich Autorensysteme so flexibel wie möglich präsentieren. Weil die Vorbereitung und die Durchführung von Unterricht Designprobleme darstellen, für die es keine eindeutigen und vordefinierten Lösungsstrategien gibt, lassen sich potentielle Intentionen und Strategien, die bei der Herstellung von Lernmaterial eine Rolle spielen, zum Zeitpunkt des Autorensystemdesigns nicht vorhersehen. Das Potential zu erwartender didaktischer Strategien und Aktionen ist ebenso *universal* wie der arithmetisch–logische Kern des Rechners. Die graphischen Software–Oberflächen, die wir heute benutzen, sind jedoch endliche Kompositionen von Menüs, Editoren und Eingabemasken. Sie sind fertig kompilierte Produkte, die oft nur in wenigen (bei ihrem Design) berücksichtigten Eigenschaften anpaßbar sind. Architekturen, die die nachträgliche Installation von PlugIns, die Löschung oder sogar Modifikation vorgesehener Bedienungselemente vorsehen sind praktisch kaum anzufinden. Daher versuchen graphische Oberflächen von Autorensystemen mit der gestalterischen Anknüpfung an sehr allgemeine Paradigmen hohe Flexibilität bei gleichzeitiger hoher Übersichtlichkeit zu erzielen. Das eigentliche Problem wird dadurch aber nicht gelöst: Ein universelles Potential auf Seiten der zu erwartenden Kommandos und ein universelles Potential im Kern des Systems werden moderiert von einem Filter (GUI), der für eine endliche Anzahl von Aktionen gestaltet wurde. Für die Herstellung von Lernmaterial und –umwelten ist es wichtiger, die Universalität der „universellen Maschine“ nutzen zu können als diese in einer bestimmten Rolle zu erleben. Es geht darum, den Computer *als Computer* zu benutzen. Tatsächlich haben die Hersteller der meisten Autorensysteme dies auf die eine oder andere Art erkannt und integrieren Schnittstellen zur textuellen Programmierung in kleinen Makro– bzw. Skriptsprachen in ihre Produkte, die von den entsprechenden Laufzeitumgebungen interpretiert werden können. Ein Beispiel ist die Skriptsprache *Lingo* des Macromedia Directors. Diese Integration erfolgt jedoch auf eine Weise, die von den Nutzern der graphischen Oberfläche bei der Herstellung spezifischer Lösungen

immer auch textuelle Programmierung erfordert. Eine Befreiung von der Notwendigkeit textueller Programmierung durch Autorensysteme ist folglich nicht gegeben.

Da die Gestaltung von Software–Nutzerschnittstellen und die Auswahl der Paradigmen, auf denen Schnittstellen aufgebaut werden, maßgeblich die Nutzerfreundlichkeit von Software bedingen, wird dieses Thema häufig im Kontext von Software–Ergonomie, auch außerhalb rein technischer Debatten, diskutiert. Hier geht es insbesondere um die Frage, inwieweit die für den Nutzer erfahrbare und analysierbare Oberfläche Interaktionen erschwert oder erleichtert. Nachdem Mensch–Computer–Interaktion jahrzehntelang praktisch ausschließlich textuell erfolgte (auch die frühen Lochkarten und Lochstreifen enthielten in Löchern bitweise codierte alphanumerische Zeichen), stellt die Entwicklung von grafischen Oberflächen und beweglichen Fenstern einen wichtigen Meilenstein in der Entwicklung von Mensch–Maschine–Schnittstellen dar.

Erste Experimente mit der Fenster–Technik erfolgten zunächst am MIT. Später wurde sie von Douglas Engelbart bei XEROX PARC weiterentwickelt, also dort, wo auch andere Elemente moderner Nutzeroberflächen wie die Maus und die interaktive Textverarbeitung ihren Ursprung haben. Diese Innovationen wurden dort in einem Computer mit dem Namen *Alto* verwirklicht. Der Alto prägte später das Design des Apple Lisa, der als erster Personal Computer eine graphische Benutzer–Oberfläche erhielt und die Nutzer–Schnittstellen der folgenden Apple–Generationen bzw. deren Betriebssysteme entscheidend prägte. Die Mischung eines „paper look screens“, wie es die Firma Apple nannte, und einer paradigmatischen Abbildung der Funktionalität einer Schreibtisch–Oberfläche mit Ablagen, Werkzeugen, Papierkorb usw. war und ist sehr populär. Sie macht Computer auch solchen Nutzern zugänglich, denen textuelle Programmierung nicht liegt und erschließt Interaktionen, die zuvor lediglich Programmierern vorbehalten waren. Das Konzept wurde mit kleineren Änderungen von Microsoft kopiert und in den Microsoft Windows–Oberflächen vermarktet[27]. Diese

[27] Auf diese Copyright–Verletzung reagierte Apple mit einer Klage, ließ diese jedoch fallen, nachdem Microsoft damit drohte, die Herstellung von Software für die Macintosh–Plattform aufzugeben. Seither steht die Verwendung der graphischen Desktop–Metapher mit rechteckigen

gemeinsamen graphischen Gestaltungsparadigmen der Apple-Macintotsh–Welt und der MS–Windows–Welt sind heute ein kaum mehr wegzudenkender Standard, der auch auf den Plattformen OS/2, Amiga, UNIX mit seinen Varianten und durch GEOS bereits dem heute antiqierten Commodore64 zur Verfügung gestellt wurde. Trotz der vereinfachten Nutzung, die uns Technologien wie Icons und Fenster eröffnet haben, bleibt anzumerken, daß diese lediglich bestimmte Nutzungsformen effektivieren. Sie verleihen dem Computer keine vorher unmöglichen Verhaltensweisen, sie befördern diese nur in den unmittelbaren Zugriff des Nutzers. Verhaltensweisen jenseits nutzerfreundlicher Paradigmen bleiben nicht nur in weiter Ferne, ihre schwere Erreichbarkeit wird durch die Etablierung anderer Paradigmen und ihrer Logik weiter erschwert. Ein praktisches Beispiel für diesen Effekt erleben wir gegenwärtig im Rahmen der Entwicklung interaktiver dreidimensionaler Schnittstellen. So wird die Interaktion etwa mit VRML–Umwelten heute dadurch erschwert, daß die Schnittstellen, die wir benutzen, seit Jahrzehnten auf zweidimensionale Interaktion zugeschnitten wurden.

Brenda Laurel diskutiert das Vorbild *Theater* und seine Qualität als Paradigma (sie benutzt den Begriff *Metapher* — die beiden Begriffe werden in diesem Zusammenhang weitgehend syonymisch verwendet) für die Gestaltung von Software–Oberflächen[28]. Die der Verwendung dieses Vorbildes zugrunde liegende Intention ist einleuchtend: Da Software in der Lage sein soll, potentiell alle in der natürlichen Welt denkbaren Probleme und Interaktionsformen abzubilden, eignet sich das Theater besonders als Modell, da es seinerseits zu dem Zweck entworfen wurde und genutzt wird, potentiell Inszenierungen aller in der natürlichen Welt denkbaren Handlungen und Interaktionen zu beherbergen und diese seinem Publikum zu vermitteln.[29]

Windows und Maus–Bedienung de facto jedem (Betriebssystem–)Entwickler frei. Die Geschichte der Entwicklung graphischer Nutzeroberflächen wird in zahlreichen verschiedenen Quellen dokumentiert. Eine dieser Quellen ist David Gelernter: The Aesthetics of Computing [21], S. 70 ff.

[28] Laurel, Brenda: [30]

[29] Interessanterweise hat auch die Geschichte des Unterrichts ihre Wurzeln in den Theatern des klassischen Griechenlands.

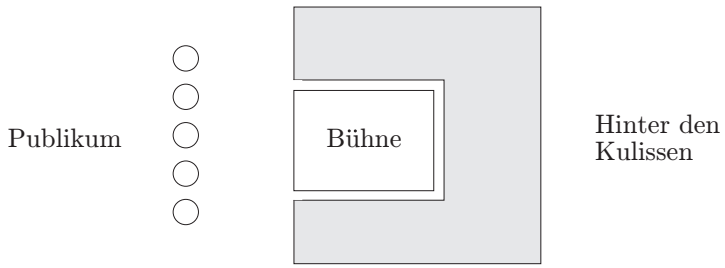


Abbildung 2.6.: Schematischer Grundriß eines Theaters[30]

Abbildung 2.6 zeigt den schematischen Aufbau eines Theater–Baus. Das Publikum blickt auf eine Bühne, die zugleich Ort der Handlung und Schnittstelle zu einer versteckten Welt hinter den Kulissen ist. In einer „funktionierenden" Inszenierung nimmt das Publikum das interne Verhalten des Theaters und seine Mechanismen nicht wahr. Es wird ihm nicht bewußt, welche Technik für Licht und Ton eingesetzt wird, wie zuvor die Schauspieler ausgewählt und die Proben organisiert wurden. Das Publikum muß über diese Dinge nicht nachdenken und sie auch nicht verstehen und kann dennoch die Inszenierung auf der Oberfläche des Theaters, der Bühne, begreifen und genießen. Die Inszenierungen knüpfen dabei mehr oder weniger an Konzepte der realen Welt und somit an die Erfahrungen des Publikums an, wodurch ein Verstehen der Aufführung ermöglicht wird (Ausnahmen bilden diejenigen modernen Inszenierungen, die diese Strategie gezielt nicht verfolgen, um Vertrautheit bewußt entgegenzuwirken).

Das Design–Paradigma Theater wurde bereits im Autorensystem–Design eingesetzt. Jahre vor dem Erscheinen von *Computers as Theatre* entwickelte Ellis Horowitz 1987 an der University of Southern California in Los Angeles ein Autorensystem mit dem Namen SCriptWriter, dessen Nutzerschnittstelle auf der Theater–Metapher aufgebaut ist und Begriffe „lines", „cast", „rehearse" und „play" verwendet (vgl. Abbildung 2.7).

Horowitz entwickelte SCriptWriter als Reaktion auf damalige Autorensysteme (er nennt namentlich das System *PC-PILOT* von IBM), deren einfache Interaktions–Möglichkeiten er als „elektronisches Seiten–Umblättern" bezeich-

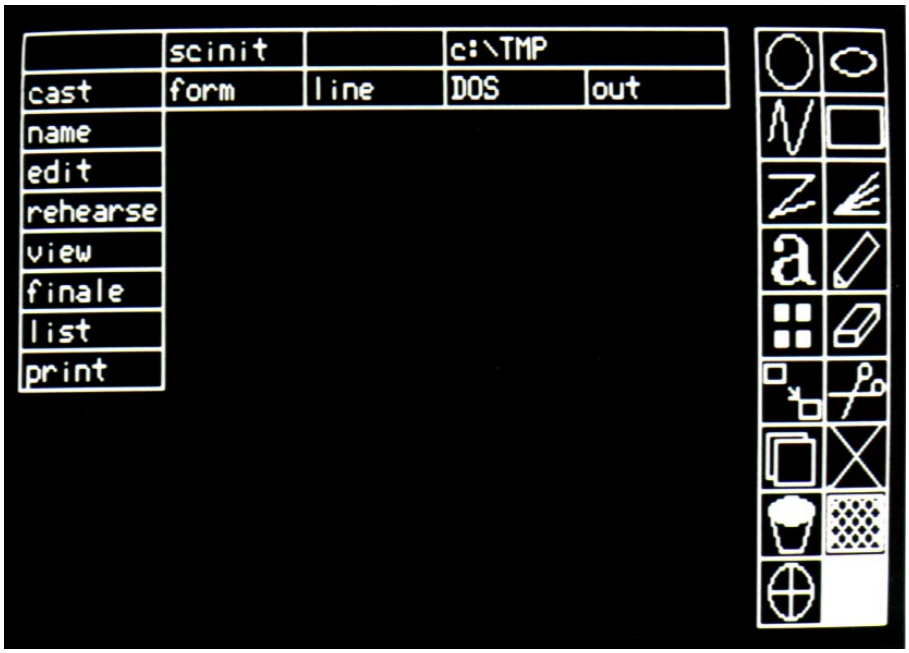


Abbildung 2.7.: Die Nutzerschnittstelle des Autorensystems „SCriptWriter“ im Autorenmodus[32]

net[33]. Neben dieser visuellen Oberfläche können Abläufe, die im Präsentationsmodus stattfinden sollen, in SCriptWriter auch in einer internen Scriptsprache mit dem Namen IQ textuell programmiert werden. Die Schlüsselwörter in IQ führen die metaphorische Theatersprache der graphischen Nutzeroberfläche fort. In dieser Tradition präsentiert sich auch der heutige Macromedia Director: Neben einer auf der Theater– (oder genauer: Film–) Metapher beruhenden graphischen Nutzerschnittstelle mit zusätzlicher Zeitachsen–Ansicht („Score“) kann im Director das Verhalten der Elemente einer Präsentation auch in der internen Scriptsprache Lingo textuell programmiert werden. Die folgende Liste beschreibt kurz die gängigsten Schnittstellen–Metaphern moderner Autorensysteme. Darin wird erkennbar, daß als Designparadigmen (man spricht auch von Interfacemetaphern) technisch geprägte Vorbilder aus dem Bereich des Softwaredesigns, der Informatik und des Screendesigns herangezogen werden. Die Breite des aufgezeigten Spektrums von textuellen bis hin zu rein visuellen Pa-

[33] Horowitz, Ellis: ebd. S. 36

radigmen zeigt, daß die Grenze zwischen Programmiersprachen, Autorensprachen und Autorensystemen trotz der in der Literatur verbreiteten theoretischen Trennung praktisch oft sehr unscharf und fließend sind.

Scripting Paradigma: In der Literatur taucht vereinzelt die Bezeichnung „Scripting Paradigma“ auf[34]. Das so bezeichnete Verfahren beschreibt jedoch gemäß der in 2.1 beschriebenen Familie von Autorenwerkzeugen lediglich das Konzept einer Autorensprache und kann das definitive Merkmal von Autorensystemen — die weitgehende Befreiung von Textorientierung durch visuelle Dialoge — nicht erfüllen. Dennoch gestatten einige Autorensysteme die explizite Formulierung von Codepartikeln, um logische und prozedurale Operationen jenseits der jeweiligen visuellen Programmoptionen bereitzustellen. Diese integrierten Verfahren werden unten näher beschrieben (Card/Scripting– und Cast/Score/Scripting–Paradigma).

Screen– bzw. Frame–Paradigma: Bei Verwendung des Sceen-Paradigma orientiert sich die Nutzung des jeweiligen Autorensystems an der Plazierung (multi)medialer Objekte auf Flächen, die Bildschirmzustände zur Laufzeit repräsentieren. Dabei referiert auch das Designparadigma der Laufzeitumgebung mitunter auf reale, seitenorientierte Präsentationswerkzeuge wie etwa Karteikästen, Dia- oder Overheadprojektoren. Das erzeugte Material kann mit einem Stapel Seiten verglichen werden, der verschiedene interne Verknüpfungen besitzt und zur Laufzeit mit Navigationsinteraktionen durchblättert wird. Das Präsentationswerkzeug MS Powerpoint ist ein Beispiel für diesen Autorensystemtyp.

Timeline–Paradigma: Timeline–basierte Autorensysteme verlangen die Plazierung von Medienobjekten auf einer Zeitachse, wodurch der zeitliche Verlauf der Präsentation zur Laufzeit bestimmt wird. Das Resultat von Navigationsinteraktionen sind „Quasi-Zeitsprünge“.

Flowchart–Paradigma: Flowchart-basierte Autorensysteme erwarten eine schematische Plazierung der Medienobjekte in Diagrammen analog zur

[34] vgl. z.B. Olszewski, Pawel [41]

Flußdiagramm-Notation, die von Neumann und Goldstine bereits Ende der 40er Jahre als Entwurfshilfe für das Programmieren entwickelten. Die Objekte werden durch Kanten (Verbindungen) in Beziehung gesetzt und somit mögliche Laufzeitzustände und –veränderungen festgelegt. Dabei sind alle logischen Relationen denkbar. Multilineare Strukturen werden im Autorenmodus übersichtlich repräsentiert.

Hypermedia–Linkage–Paradigma: Wie beim Screen-Paradigma orientiert sich das Hypermedia–Linkage–Paradigma an der Komposition von Bildschirmzuständen. Jedoch werden die einzelnen Bildschirme („Screens“) in einer zentralen Übersicht ikonografisch repräesentiert und durch Kanten in Relation gesetzt, die die Hypermedia–Verlinkungen zwischen den einzelnen Screens (bzw. Hypertextknoten) visualisieren.

Hierarchical–Object–Paradigma: Dieses Paradigma benutzt das sogenannte *Objekt* als Metapher und erweitert somit das Prinzip der objektorientierten Programmierung um eine visuelle Oberfläche. Es ist relativ komplex und vergleichsweise schwer zu erlernen. Dementsprechend werden sie von Experten eingesetzt, um komplexe, meist hardwarenahe Software zu realisieren. Ein Beispiel für ein (immerhin relativ didaktisch ausgerichtetes) Autorensystem dieser Art ist Dazzler Deluxe.

Card/Scripting-Paradigma: Dieses Paradigma ist dem Screen–Paradigma ähnlich: Medienobjekte werden auf Screens bzw. Frames plaziert, die wiederum als verknüpfter Stapel die Hypermedia-Umgebung repräsentieren. Hinzu kommt, daß den einzelnen Objekten Programmcode (Scripte) beigefügt werden kann. Diese werden über „Eventhandler“ als Interaktionsresultat ausgeführt und gestatten logische Informationsverarbeitungen. Card/Scripting–Autorensysteme sind leicht erweiterbar und vergleichsweise nutzerfreundlich. Sie emfehlen sich für navigationsreiche Anwendungen insbesondere im Unterhaltungsbereich. Sie werden oft zur Prototyperherstellung herangezogen, bevor die jeweiligen Anwendungen (hauptsächlich zur Verbesserung der Laufzeit) in eine Compiler–Programmiersprache portiert werden. Ein bekanntes Beispiel für ein auf

diesem Paradigma aufbauenden Autorensystem ist Toolbook 6.

Cast/Score/Scripting–Paradigma: Das Cast/Score/Scripting Paradigma verlangt die Plazierung von Medienobjekten auf einem Set paralleler Spuren, die zur Laufzeit synchron abgearbeitet werden. Zeitlich parallele Elemente oder Ereignisse befinden sich auf jeweils gleicher Höhe. Diese Form der visuellen Programmierung läßt sich zur Verdeutlichung mit der Struktur einer Musik–Partitur erklären. Die Stärke dieser Metapher liegt in der Fähigkeit, das Verhalten jedes Mitwirkenden (Medienobjekts) auf der Zeitachse zu beschreiben. Bekanntester Vertreter dieser Metapher ist der Macromedia Director, der vielfach für die Erstellung hochwertiger kommerzieller Anwendungen verwendet wird. Dieses Paradigma erlaubt (über logische Anordnungen hinaus) das Anfügen von Scriptelementen an beliebige Medienobjekte zur prozeduralen und/oder ereignisgesteuerten Verarbeitung von Nutzer–Interaktionen.

Textkonverter: Autorensysteme, deren Nutzerschnittstellen auf diesem Paradigma beruhen, zeichnen sich durch minimale Dialoge mit Lehrenden aus. Es werden beliebige Texte importiert und mit vorgefertigten Algorithmen typische Lerneinheiten für die Leseförderung wie z.B. Lückentexte, Zuordnungstests oder Multipe-Choice–Tests generiert. In einem Autorensystem dieser Art kann meist aus mehreren bereitgestellten Testvarianten gewählt werden. Dieser Autorensystem–Typ zeichnet sich praktisch als einziger durch eine explizite und ausschließliche didaktische Orientierung aus. Während der Lehrer bzw. Autor das Autorenwerkzeug als Textkonverter, also als technisch ausgerichtetes Werkzeug zur Datenmanipulation wahrnimmt, orientiert sich die Lernerschnittstelle an dem Erscheinungsbild, das die jeweils verwendeten Übungen zur Förderung von Textverständnis auch in gedruckter Form aufweisen. Hierauf hat der Lehrer praktisch ebenso wenig Einfluß wie auf die Strukturen der Lerner–Interaktion (Reihenfolgen, Timing, Bewertungssysteme, etc.). Da die Herstellung der Tests durch relativ triviale Algorithmen ohne inhaltliche Analyse erfolgt, ist für einige Testvarianten die Bereitstellung detaillierter Zusatzinformationen durch den Lehrer erforderlich, wie z.B. alternative richtige und falsche

Antworten für die Generierung von Multiple-Choice–Tests. Beispiele für Autorensysteme dieser Art sind *WinWida* und *Lectra*.

Kommunikationsplatformen: Da der Begriff „Autorensystem“ mittlerweile eher unverbindlich im Sinn von „Werkzeug für Unterrichtszwecke“ benutzt wird, wird er auch für Software benutzt, die letztlich gar nicht zum *Authoring*, also zur Materialherstellung, genutzt werden kann, aber Interaktions– und Kommunikationsumgebungen für Unterricht bereitstellen. Bestes Beispiel hierfür ist WebCT, das mit eMail–Listen, Whiteboard etc. der netzwerkbasierten unterrichtlichen Kommunikation dient und mit Daten–Archiven lediglich als Sammel– und Distributionsmittel für Unterrichtsmaterial dient. Daß WebCT praktisch ohne Programmieren bedient werden kann, schlägt sich in sehr mangelhafter Flexibilität und starrer Abgeschlossenheit nieder.

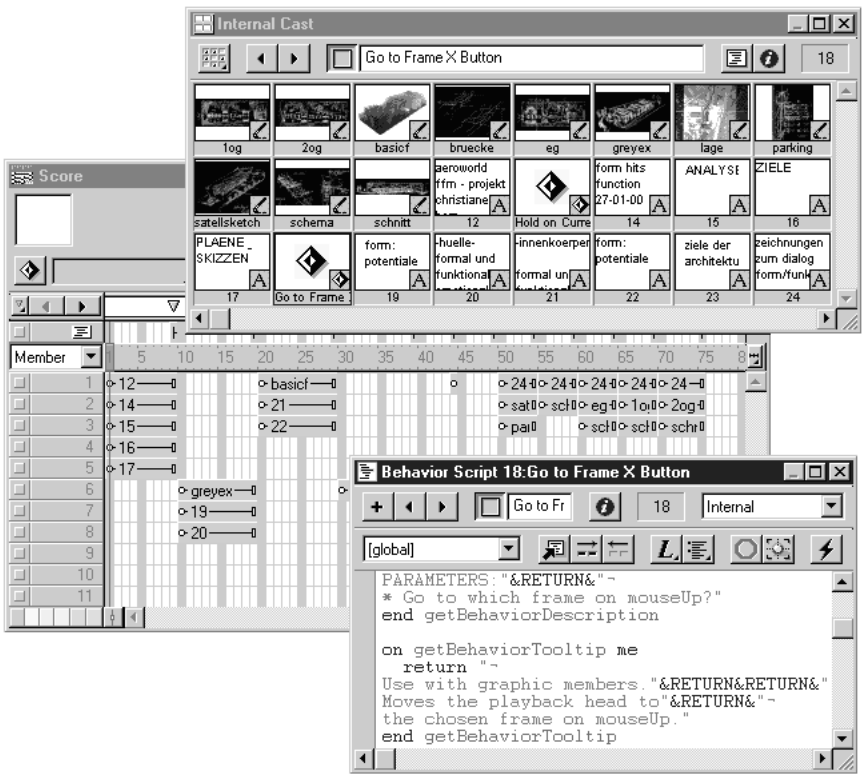


Abbildung 2.8.: Cast- Score- und Scripting–Ansicht im Macromedia Director

Abbildung 2.8 zeigt die Oberfläche von Macromedia Director. Sie enthält Elemente des Cast–Paradigmas, des Score–Paradigmas (dadurch auch des Timeline–Paradigmas), des Scripting–Paradigmas, eine (in dieser Abbildung nicht enthaltene) Leinwand für frame– oder screenorientierte Gestaltung der Produkt–Oberflächen und greift außerdem auf Film– bzw. Theatermetaphorik zurück.

Die Qualität von Nutzer–Schnittstellen ist ein zentraler Aspekt der Software–Gestaltung. Da es sich bei der Gestaltung von Software für die Lösung bösartiger Probleme selbst um ein bösartiges Problem handelt, sind es leider die Nutzer, die in ihrer Arbeit mit dem Computer herausfinden müssen, wie gelungen eine Oberfläche tatsächlich ist. Die Urteile hierüber fallen allerdings selten eindeutig aus, und Meinungsverschiedenheiten bei Urteilen über Nutzer–Schnittstellen führen immer wieder zu sogenannten „heiligen Kriegen“, wie dem zwischen Macintosh– und MS Windows–Nutzern. Obwohl für das bösartige Problem des Schnittstellen–Designs keine *richtige* Lösung existiert, helfen die Begriffe *zahm* und *bösartig* auch hier, einen Erklärungsansatz zu entwickeln:

Das Werkzeug Computer kann prinzipiell sowohl zur Bearbeitung *zahmer* als auch *bösartiger* Probleme eingesetzt werden. Da zahme Probleme in definierte Einzeloperationen zerfallen, können Nutzer–Schnittstellen leicht den Zugriff auf diese Operationen und damit nutzerfreundliche Lösungen zahmer Probleme ermöglichen. Da im Fall bösartiger Probleme nicht vorhersehbar ist, welche Operationen der Nutzer bei der Lösung benötigt, sind die stark begrenzten graphischen Nutzerschnittstellen, die wir heute kennen, keine ideale Hilfe.

Neal Stephenson kritisiert 1999 die Verwendung von GUIs und vergleicht sie mit einem Besuch in Disneyland, bei dem man die Welt nicht so erlebt, wie sie wirklich ist, sondern so trügerisch–verzerrt, wie Disney sie uns mit einer aufwendigen Oberfläche zeigt[35]. Damit widerspricht er Ted Nelson, der ausführt, daß gutes Schnittstellen–Design gerade die Qualitäten erfordert, die für Disneys Produkte

[35] Stephenson, Neal: [56], S. 46ff.

typisch sind[36]. Offenbar denken beide Autoren hier an unterschiedliche Aufgabenbereiche. Während Nelsons Forderungen nach aufwendigen vorproduzierten Dialogelementen auf die Lösung zahmer Probleme anwendbar sind, können sie die Lösung bösartiger Probleme nicht konsistent begleiten. Es wäre nicht einmal möglich, die nicht vorhersehbare Menge potentieller Kommandos, die bei der Bearbeitung bösartiger Probleme anfallen, ausreichend in einer visuellen Schnittstelle zu repräsentieren. Oberflächen von Lernumgebungen mögen mitunter von Disney–artigen Schnittstellen profitieren. Autorensysteme für deren Herstellung würden unter der Endlichkeit heutiger visueller Schnittstellen leiden.

Hinzu kommt ein weiteres Problem: Die in Autorensystemen oft anzutreffenden paradigmatischen Referenzen auf technische Vorbilder sind meist schlecht gewählt. Die Dialoge, die Schnittstellen ermöglichen, werden von der Natur der ihnen zugrunde liegenden Metapher geprägt. Je näher die *Weltausschnitte* bzw. die Konzepte, an denen sie sich orientieren, inhaltlich und formal mit dem jeweiligen Problembereich korrespondieren, desto nützlicher sind sie. Einfachste Autorensysteme, deren Verhalten im Präsentationsmodus nicht wesentlich über das eines Diaprojektors hinausgeht (z.B. MS PowerPoint) brauchen nicht die Fähigkeit, die Komplexität der ganzen Welt so abbilden zu können wie das Theater, und ein Frame–orientiertes Schnittstellenparadigma erscheint ausreichend. Das resultierende Autorensystem wird mit seiner Schnittstelle jedoch nicht das didaktische Potential eines Diaprojektors übertreffen können. Es stellt sich daher die Frage, ob sich Autorensysteme durch die Verwendung von Theater– oder Film–Metaphern wirklich sinnvoll dem Gegenstand „Unterricht“ annähern. Der inhaltlich und formal naheliegende Weltausschnitt *Schule* erscheint aus dieser Perspektive wesentlich sinnvoller.

[36] Nelson, Theodor H.: The Right Way to Think About Software Design. In Laurel, Brenda [29], S. 243

2.7. Unterricht, Programmierung und Design

Die Entwicklung sowohl von computerbasiertem Unterricht als auch von Werkzeugen dafür sind hochgradig interdisziplinäre Aufgaben. Ihre Teilprobleme erstrecken sich über den jeweils zu vermittelnden Gegenstandsbereich hinaus auch über Wissensgebiete der Didaktik und relevanter Fachdidaktiken, der Lernpsychologie, der Mediengestaltung, der Informatik etc. Sie erfordern weiterhin Kompetenzen in den Bereichen Entwurf und Problemlösung. Die aufgezählten Disziplinen versuchen in der Regel, die Herausforderungen des computerbasierten Unterrichts eigenständig aus ihrer isolierten Sicht zu untersuchen und Lösungsstrategien zu entwerfen. Dabei nutzen sie nicht das Potential interdisziplinärer Kooperation und versäumen, wertvolle Erfahrungen anderer Disziplinen in die jeweiligen Entwicklungen einzubringen. Besonders interessant erscheinen mir Parallelen, die ganz offensichtlich zwischen Unterrichtswissenschaft, angewandter Informatik und der Designwissenschaft bestehen. Ich skizziere diese Parallelen anhand der folgenden (sicherlich unvollständigen) Auflistung:

- Unterricht, Programmierung und Design bemühen sich um die Entwicklung von Umwelten für Menschen, die das Leben und die Entwicklung der Menschen unterstützen.
- Unterricht, Programmierung und Design bemühen sich um eine ständige Weiterentwicklung dieser Umwelten: Innovation ist ein typisches (obgleich nicht zwingendes) Element ihrer Produkte.
- Unterricht, Programmierung und Design vollziehen sich typischerweise im Rahmen von Kooperationen: Sie erfolgen nicht in Isolation. Die Aufteilung in „Planer“ und „Beplante“ erfordert vom Entwerfer *Rollenwechsel*.
- Die Ziele von Unterricht, Programmierung und Design sind in hohem Maß fallbasiert: Sie müssen für jeden Entwurfsprozeß individuell definiert und währenddessen ständig überprüft und ggf. angepaßt werden.
- Die Methoden und Ziele von Unterricht, Programmierung und Design sind in hohem Maß an Normen, Werte und ihren gesellschaftlichen Kontext gekoppelt. Da diese sich mitunter wissenschaftlichen Modellen der

Erkenntnisgewinnung entziehen, hat die Praxis in diesen Bereichen *außerwissenschaftliche* Elemente.

- Die Tätigkeiten Unterrichten, Programmieren und Designen erfordern jeweils die Erarbeitung umfassender Kompetenz in den jeweiligen Gegenstandsbereichen. Daher sind die drei Tätigkeiten auch ausgezeichnete Lernmethoden.

- Die Effizienz von Lösungen zunächst konkreter Aufgaben in den Bereichen Programmierung, Unterricht und Design kann wesentlich von abstrahierenden Strategien profitieren. Das Resultat sind *generische* Lösungen, die über konkrete Aufgaben hinaus auf ganze Aufgabenklassen angewendet werden können. Diese Lösungen bieten Chancen für effizientes, methodisches Arbeiten. Da in den genannten Disziplinen jedoch keine Aufgabe der anderen gleicht, sollten diese generischen Lösungen nicht unreflektiert auf neue Probleme angewendet werden.

- In den Fachtheorien aller drei Disziplinen werden die Verhältnisse zwischen Planern und Beplanten als hierarchische Kaskaden dargestellt (vgl. Abb. 2.4). Obgleich derartige modellhafte Repräsentationen von Weltausschnitten für theoretische Darstellungen nützlich sein mögen, treffen sie nicht ganz die Struktur tatsächlich möglicher Verhältnisse von Planern und Beplanten: Lehrende sind immer auch Lerner, Lerner können durchaus auch Lehrende sein oder es irgendwann werden. Programmierer sind auch Anwender usw.

- Der Computer als „universelle Maschine“ kann als Werkzeug und Vehikel für das Unterrichten, das Programmieren und das Designen dienen.

Trotz offensichtlich durchaus vergleichbarer Aufgabenstellungen haben die drei Disziplinen zum Teil sehr unterschiedliche Theorien und Problemlösungskonzepte entwickelt. Eine wesentliche Herausforderung eines interdisziplinär orientierten Projektes besteht darin, die jeweils interessanten Aussagen, Theorien und Modelle der beteiligten Fächer zu identifizieren und zu koordinieren. Auftretende Widersprüche bieten dabei die Chance, die Qualitäten fachlicher

Beiträge zu hinterfragen und neue, ggf. bessere Ansätze zu erarbeiten. Andererseits muß man auch mit potentiellen Verständnis– und Verständigungsproblemen rechnen, weil verschiedene Fächer unterschiedliche Kulturen, Traditionen und Sprachen pflegen. Sie verwenden verschiedene Begriffe und Strategien, die die Kommunikation zwischen ihnen und damit die Qualität eventueller gemeinsamer Produkte bedrohen.

Ein weit verbreitetes Modell zur methodischen Problemlösung in der Informatik, das sogenannte *Wasserfall-Modell*, geht davon aus, daß die Lösung eines Problems zunächst das grundsätzliche Verstehen des Problems erfordert, dann ein Sammeln relevanter Informationen, die anschließend zu analysieren sind. Basierend darauf kann eine Lösung formuliert werden, die es im letzten Schritt zu implementieren gilt. Die aufgezählten Schritte seien nur in dieser Reihenfolge anwendbar.

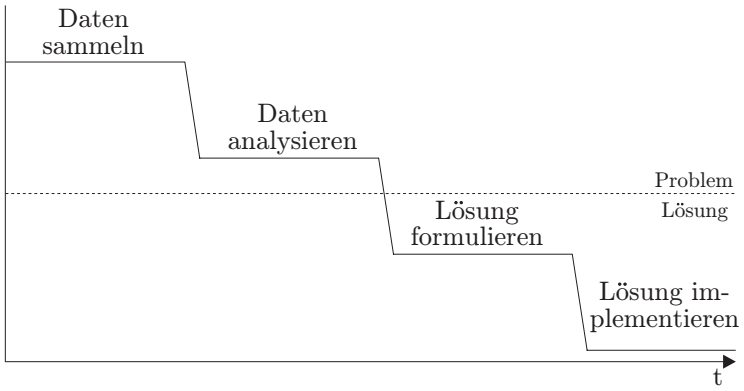


Abbildung 2.9.: Traditionelle Sicht der Lösung komplexer Probleme in der Software-Entwicklung: Der „Wasserfall“[38]

Das Wasserfall–Modell ist ein klassisches *Stufenmodell* zur Beschreibung von Entwurfsprozessen. Stufenmodelle basieren typischerweise auf der Annahme, daß Entwurfsprozesse in aufeinanderfolgende, rationale Einzelschritte zerlegbar und daher auch als solche durchführbar sind. Wird ein Stufenmodell für die Lösung eines bösartigen Problems herangezogen, ergibt sich ein Widerspruch mit der Feststellung Rittels und Webbers, daß Designprobleme nicht eindeutig

[38] nach Conklin, E. Jeffrey und Weil, William: [8]

und endgültig formulierbar sind und daß sich für sie keine endliche Auflistung von Lösungsoperationen formulieren läßt. Stufenmodelle entsprechen z.B. Bauanleitungen oder Spielregeln und sind somit Werkzeuge für die Lösung zahmer (oder besser: zuvor gezähmter) Probleme. Sie sind für die Lösung von Entwurfsproblemen ungeeignet. Nicht nur die Software–Industrie benutzt Stufenmodelle, sondern auch die Didaktik — insbesondere in der Lehrerausbildung. Im folgenden Beispiel aus dem Jahr 1994 beschreibt Witzenbacher den didaktischen Entwurfsprozesses als Stufenmodell:

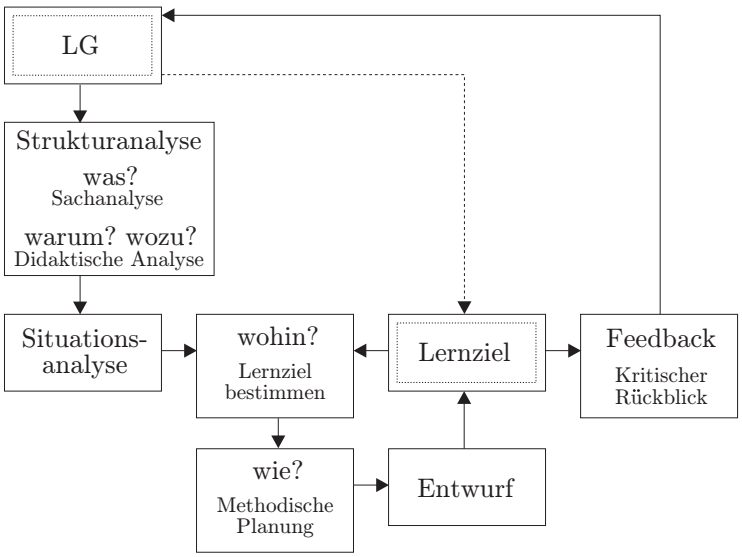


Abbildung 2.10.: Phasen der Unterrichtsplanung[39]

„Die Reihenfolge der Planungsschritte läßt sich nicht umkehren oder verschieben. Zuerst muß der Lerngegenstand (LG) im Hinblick auf den Lehrer (*Sachanalyse*), dann im Hinblick auf die Schüler (*didaktische Analyse*) untersucht werden. Auf diese *Strukturanalyse* folgt eine Untersuchung der Klassen- und Individualsituation im Hinblick auf den ausgewählten Lerngegenstand (*Situationsanalyse*). Erst jetzt kann, unter Berücksichtigung dieser Untersuchungsergebnisse, das *Lernziel* bestimmt und sprachlich präzise formuliert wer-

[39] nach Witzenbacher, Kurt [60]

> den. Die *methodische Planung* kann erfolgen, wenn alle diese Untersuchungen durchgeführt worden sind. Im Entwurf werden die Ergebnisse der Analysen, die Formulierung des Lernziels (der Lernziele) und der geplante Verlauf der Unterrichtseinheit oder Unterrichtsstunde *schriftlich fixiert*. Schließlich findet im *kritischen Rückblick* eine Rückkoppelung (Feed back) statt, die von dieser Planungseinheit zur nächsten überleiten wird."[40]

In der Didaktik sind solche Stufenmodelle für den Unterrichtsentwurf nicht unumstritten[41]. Dieser Kritik schließe ich mich an und weise darauf hin, daß Strategien wie die oben zitierte implizit nicht nur ein hohes Maß an verfügbarer Lehrer–Arbeitszeit sowie ausreichende (empirische) Erkenntnisse etwa über Lernvoraussetzungen der Schüler oder über die Wirksamkeit didaktischer Methoden voraussetzen. Besonders fatal erscheint mir darüber hinaus die Gefahr, daß Lehrentscheidungen, die in einer solchen Unterrichtsvorbereitung getroffen und festgeschrieben wurden, als unverrückbare Konstanten in den Unterricht eingehen, denen sich alles *Unvorhergesehene* unterzuordnen hat. Alltägliche Variablen wie individuelle Wünsche oder Probleme Lernender werden zwangsläufig zu Störfaktoren, da sie nicht Teil des entworfenen Unterrichts sind. Daher wirken sie als Fremdkörper, als nicht förderlich für die Durchführung des Entwurfs und werden letztlich als zu vermeidende Probleme behandelt.

Folglich drohen wichtige Gründe für das institutionale Lehren in Schulen, die soziale– und Kommunikations-Praxis und die sofortige Berücksichtigung individueller Lernbedürfnisse von Lehrenden selbst als Problem für ihre Tätigkeit verstanden zu werden. In der Lehrerausbildung jedoch fehlt ohne vorgegebene Modelle der formale Maßstab anhand dessen Unterricht beurteilt werden kann. Für das Designproblem Unterrichtsentwurf gibt es weder Standardprozeduren noch richtige oder falsche Lösungen, sondern nur bessere oder schlechtere. Es sind auch keine zwei Unterrichtssituationen gleich, da sich Rahmenbedingungen, Inhalte, Schüler und ihre Lernbedingungen usw. immer unterscheiden. Somit sind Unterrichtssituationen letztlich auch schwer vergleichbar. Diese Situation

[40] Witzenbacher, Kurt [60], S. 28-29

[41] vg. z.B. Meyer, Hilbert: [36], S. 337 ff.

macht die praktische Lehrerausbildung zu einem schwierigen Unterfangen.

Nachdem das Problem Unterrichtsentwurf wie in Abb. 2.10 durch ein Stufenmodell gezähmt wurde, kann das Verständnis und die Einhaltung dieses Planungs– und Durchführungsmodells geprüft, verglichen und leicht beurteilt werden. Dabei droht die Qualität von Unterricht an zugrundeliegenden Modellen und nicht an gegebenen Unterrichtssituationen selbst gemessen zu werden. Andererseits bieten generalisierte Modelle für den Unterrichtsentwurf Vorteile — beispielsweise dadurch, daß sie Studierende für die Wichtigkeit umfassender Planungs sensibilisieren, daß ihre klaren Vorgaben Planungen beschleunigen können und daß sie Entwürfe leicht kommunizierbar machen. Letztlich führen Phasen– oder Stufenmodelle zu der Annahme, gestalterische Prozesse ließen sich als lineare Sequenz aufeinanderfolgender Planungsschritte beschreiben. Dies trifft auf reale Entwurfsprozesse aber nur sehr selten (vielliecht überhaupt nicht) zu.

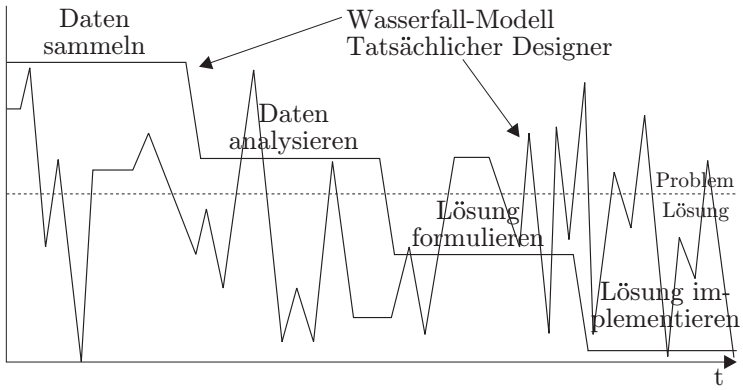


Abbildung 2.11.: Wasserfall–Methode und tatsächlicher Designprozeß[42]

Das Lösen von Designprozessen und somit auch der Unterrichtsentwurf lassen nicht in rationalisierte, standardisierte und sequenzialisierte Schritte zerlegen. Die zum lösen von Designproblemen erforderlichen Schritte können typischerweise weder in ihrer Menge noch in ihrer Gewichtung und Sequenz vorhergesehen werden. Daher verfahren Designer und Lehrer typischerweise nicht so, wie es das Wasserfall–Modell vorschreibt bzw. empfiehlt. Abbildung 2.11 zeigt einen Vergleich zwischen dem idealisierten Wasserfall–Modell und einem cha-

[42] nach Conklin, E. Jeffrey und Weil, William: [8]

rakteristischen Ablauf eines Designprozesses.

Wenn Designer, wie z.B. Architekten, Grafik–, Industire–, oder Mode–Designer, ihre Produkte entwerfen, findet dies als ein Schritt innerhalb komplexer Betriebsabläufe statt. Diese Prozeduren umfassen auch andere Aufgaben wie etwa Rohstoff–Gewinnung und –Transport, Fertigung, Qualitätskontrolle, Auslieferung, Marktanalyse, Rechtsberatung usw. sowie die Koordination aller beteiligten Ressourcen. Im Gegensatz zu diesen Aufgaben, die ein Produkt während seiner Entwicklung, Produktion, Vertrieb und Anwendung begleiten, beschränkt sich die Aufgabe des Designers zeitlich auf einen kleinen Zeitraum zu Beginn dieses Produkt–Lebens, in dem die Eigenschaften der jeweiligen Produkte festzulegen sind. Ein Mantel wird möglicherweise innerhalb weniger Stunden entworfen. Der Fertigungs–Prozeß und die Auslieferung an den Handel erfolgt aber über mehrere Monate oder Jahre, während denen ständig Rohstoffe gewonnen und transportiert, Qualitäten kontrolliert und Marktsituationen analysiert werden müssn usw. Selbstverständlich muß der Designer in diesem Prozeß nicht unbedingt mit allen Methoden und Technologien vertraut sein, die in diesem Betriebsablauf außerhalb seiner eigenen, gestalterischen Zuständigkeit liegen.

Obwohl die zentrale Aufgabe des Online–Lehrens, die Gestaltung lernfördernder Umwelten, als bösartiges Problem der Arbeit des Designers entspricht, findet der Lehrer hier bedeutend schwierigere Bedingungen als der Designer vor. Er teilt sich seine Aufgaben praktisch nicht oder nur in geringem Umfang mit anderen und ist typischerweise selbst für sämtliche Planungs–, Ausführungs– und Reflexionsschritte seines Unterrichts verantwortlich. Daher beschränkt sich seine Zuständigkeit nicht auf einen Teil des Produktlebens wie z.B. den Entwurf einer einzelne Unterrichtsstunde sondern auf das Produktleben gesamter Unterrichtsphasen und er trägt während *aller* Phasen die volle Verantwortung für die Qualität seines Produkts *Unterricht*. Daher muß er sämtliche Methoden und Technologien beherrschen, die während des gesamten Produktlebens erforderlich sind. Dies umfaßt auch Bereiche wie Materialrecherche, Transportlogistik, Nutzerverwaltung, Evaluation etc., für die Designer in anderen Bereichen nicht zwingend verantwortlich sind. Obwohl das Problem des Unterrichtsentwurfs also entwerferischer Natur und insofern mit anderen Designtätigkeiten vergleichbar

ist, sehen sich Lehrer mit vergleichsweise mehr Aufgaben und Verantwortungen konfrontiert. Der Alltag der Lehre ist angesichts der resultierenden bzw. ohnehin gegebenen inhaltlichen und zeitlichen Anforderungen kein idealer Kontext für den Erwerb und die Pflege umfassender technologischer Kompetenzen.

Natürlich trägt der Lehrer nicht in allen Unterrichtssituationen allein die Verantworung für die Qualität der Lehre. Ausnahmen bilden Unterrichts–Modelle, in denen Lehraufgaben auf mehrere Lehrende verteilt sind. Unser Verständnis der Bösartigkeit des Problems Unterrichtsentwurf enthüllt hier eine weitere Problemquelle, die die Arbeitsbedingungen des Lehrers — sowohl im traditionellen als auch im computervermittelten Unterricht — weiterhin signifikant erschweren können. Bild 2.12 zeigt den quasi–chaotischen Verlauf der Designprozesse zweier Entwerfer im Vergleich zum zahmen Wasserfall-Modell. Es ist sehr schwierig, die Planungsstrategien mehrerer an einem Projekt beteiligter Entwerfer zu koordinieren. Darüber hinaus ist es leicht möglich, daß zwischen den beteiligten Entwerfern Konflikte entstehen, weil ihnen dieser Effekt überhaupt nicht bewußt ist — insbesondere wenn sie Entwurfsmodelle wie das Wasserfallmodell verinnerlicht haben.

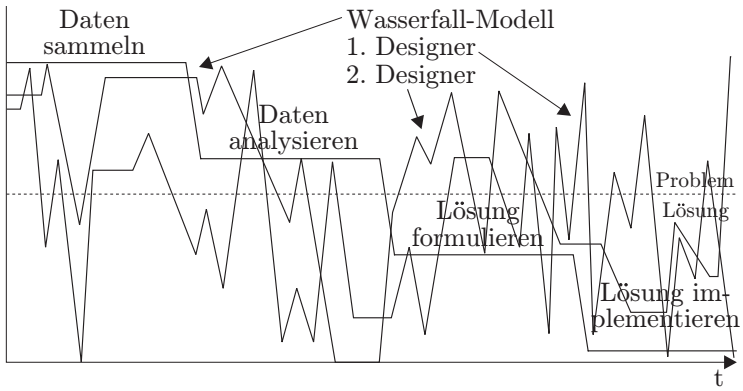


Abbildung 2.12.: Wasserfall–Methode und Verhalten mehrerer Entwerfer[43]

Eine andere Situation, in der Lehrende bei ihrer Arbeit Unterstützung bei der Unterrichtsgestaltung erhalten, finden wir bei der Verwendung von Lehrmitteln von Dritt-Anbietern wie etwa Schulbuch–Verlagen. Es ist eine gängige und

[43] nach Conklin, E. Jeffrey und Weil, William: [8]

bewährte Praxis, daß Lehrer Schulbücher verwenden, um Übungsmaterialien daraus auszuwählen, sie zu neuen Materialien zusammenzustellen und Lernenden z.B. zur Bearbeitung etwa im Rahmen von Gruppen– oder Einzelarbeiten zu übergeben. Dieses Verfahren scheitert in der Praxis des computerbasierten Unterrichts zu oft an der mangelhaften Designqualität sowohl digitaler Lernmittel als auch der Autorensysteme, mit denen diese hergestellt werden. Digitale Lehrmittel sind in der Regel nicht *interoperabel* (darauf werde sich später noch detaillierter eingehen). So ist es, teils aus technischen – teils aus wirtschaftlichen Gründen, oft unmöglich, Lehrmittel so zu verwenden, wie Lehrer es von der Verwendung traditioneller Medien gewöhnt sind — z.B. Teile von Unterrichtsmaterialien isoliert oder in einem anderen Format zu präsentieren, Produkte von Kollegen in ein bevorzugtes, anderes Datenformat zu konvertieren, Elemente anderer Lernumgebungen oder sogar anderer Institutionen in ein eigenes Programm zu integrieren usw. Im traditionellen Klassenunterricht mit papierbasierten (weniger dynamischen) Materialien sind diese Strategien der Lernmittelherstellung (z.B. mit Schere und Fotokopierer) problemlos möglich.

2.8. Gewinnung und Anwendung von Erkenntnissen

Da die praktische Herstellung und Anwendung von Lernsoftware der wissenschaftlichen Sonderstellung der Unterrichtsforschung bis heute nahezu ausnahmslos zuwiderhandelt, erläutere ich an dieser Stelle zusammenfassend das Problem der Übertragbarkeit (empirischer) didaktischer Erkenntnisse und Modelle auf neue Unterrichtssituationen[44].

Wir bemühen uns beim Entwerfen im Allgemeinen und beim Unterrichten im Besonderen — vor dem Hintergrund der sozialen Verantwortung, die damit einhergeht — *gut* zu entwerfen, indem wir unser Wissen und unsere Erfahrung, also unsere gesammelten Erkenntnisse bzw. unser Wissen, in den Designprozeß einbringen. In diesem Sinn soll Unterricht möglichst systematisch, methodisch, interessant und effektiv sein. Der Unterrichtserfolg soll nicht dem Zufall

[44] Dieses Problem wird ausführlich dokumentiert in: Walter, Hellmuth: [58] und Walter, Hellmuth und Edelmann Inge: [59]

überlassen, sondern mit methodischem didaktischen Handeln gesichert werden. Auf diese Weise soll einerseits *effektives Lernen* ermöglicht werden, andererseits kann Unterricht nur auf der Basis methodischer Planung und Handlungsbeschreibungen *verantwortet* werden. Das Fehlen formaler Bewertungssysteme in Designkontexten macht jeden Entwurf — auch Unterricht — zum Versuch. Daher impliziert das Entwerfen immer das Ausprobieren. Dieses Ausprobieren darf jedoch nicht zufällig erfolgen.

In diesem Abschnitt diskutiere ich das Verhältnis vom Design digitaler Lernmittel zur rationalen und methodischen Gewinnung und Anwendung unterrichtlicher Erkentnisse. Wie kann ohne das Vorhandensein eines formalen Bezugsrahmens für die Qualitätsmessung Unterricht effektiv und verantwortbar gestaltet werden? Welche Methoden zum Lernen aus vergangenen Planungen gibt es? Wie hilfreich erweisen sich solche retrospektiven Analysen und Beschreibungen für die Erstellung neuer Unterrichtsentwürfe? Wie zuverlässig sind während der Unterrichtsplanung getroffene Vorhersagen über zukünftiges didaktisches Handeln? Und schließlich: In welchem Verhältnis stehen Merkmale von Werkzeugen zu den Antworten auf diese Fragen? Was bedeutet dies für das Design von Autorensystemen?

2.8.1. Qualitätssicherung durch Feedback

Da die rechtzeitige Berücksichtigung schwer vorhersehbarer Lernsituationen sowohl dem Stufenmodell–basierten Unterrichtsentwurf als auch der Gestaltung computerbasierter Lehrmitteln Schwierigkeiten bereiten, klingen Versprechen methodischer Feedback–Gewinnung etwa seitens der Medienpädagogok und der Unterrichtsforschung für die Entwicklung didaktischer Software sehr vielversprechend. Direkte Formen der Feedback–Gewinnung sind angesichts der sozialen Natur des Problemfeldes Unterrichtsentwurf jedoch methodisch und ethisch sehr problematisch. Ich werde in diesem Abschnitt die drei typischen Methoden der Qualitätsmessung und –beurteilung im Bereich didaktischer Software und ihre Probleme im Hinblick auf das Autorensystem–Design diskutieren.

Abbildung 2.13 zeigt ein Modell der Feedback–Erzeugung und –Verwertung am

Beispiel digitaler Unterrichtsmaterialien. Das Verfahren des praktischen oder simulierten Testens mit dem Ziel der Verbesserung neuer Produkte ist in der angewandten Informatik und in der Mediengestaltung (beides Randgebiete der interdisziplinären Aufgabe der Herstellung computerbasierter Lernmittel) weit verbreitet. Daher liegt der Schluß nahe, dieses Vorgehen sei einfach auf die Lernmittel–Herstellung und das Autorensystem–Design übertragbar.

Bevor ich diese Annahme im Detail diskutiere, möchte ich nicht versäumen, an dieser Stelle kurz auf zwei andere auffallende Merkmale von Bodendorfs Modell eingehen. Zunächst erscheint es nämlich fraglich, ob eine lineare Abfolge von Entwurfsschritten dem Problemcharakter des Unterrichtsentwurfs und ob die implizite paradigmatische Referenz auf die Produktion traditioneller Medien („Drehbuch“) dem didaktischen und gestalterischen Potential von Hypermedia gerecht werden.

Ein weiteres auffälliges Merkmal des hier dargestellten Modells ist neben seinem Stufen–Aufbau die Erzeugung eines Feedbacks durch eine kritische Beurteilung von Teilschritten des Entwurfsprozesses, deren Erkenntnisse auf das jeweilige Projekt mit dem Ziel der qualitativen Verbesserung angewendet werden sollen[45].

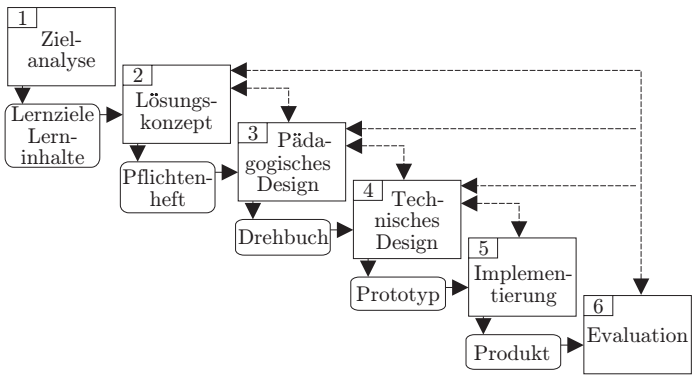


Abbildung 2.13.: Phasenkonzept der Teachware–Entwicklung nach Bodendorf[47]

[45] Um Verwechslungen vorzubeugen, weise ich darauf hin, daß mit dem hier verwendeten Feedback–Begriff die Resultate der kritischen Beurteilung von Gestaltungskriterien gemeint sind und nicht das Lerner–Feedback, das Lernsoftware im Dialog mit Schülern zur Bewertung erbrachter Lernleistung erzeugt.

[47] Bodendorf, Freimut[4], S. 76

Weitere Probleme ergeben sich insbesondere aus dem aufgezeigten Modell der Feedback–Erzeugung und –Evaluation. Wissenschaftliches (oder journalistisches) Feedback über die Qualität von Designprodukten wie Autorensystemen oder digitalen Lernmitteln können in drei Kategorien unterteilt werden:

1. Rezensionen in Textform
2. Überprüfung mit Merkmalslisten bzw. Checklisten
3. Experimentelle Erfolgsmessungen

Rezensionen in Textform unterscheiden sich nicht wesentlich von Literatur–Rezensionen. Sie werden ohne die formale Konstruktion eines Bezugsrahmens verfaßt und haben keinen Anspruch auf Vollständigkeit. Sie sind billig herzustellen, werden in der Regel in Fachzeitschriften veröffentlicht und geben die Sichtweise eines Beurteilenden wieder. Rezensionen richten sich hauptsächlich an Praktiker, die für die Auswahl ihrer Werkzeuge eine Entscheidungshilfe suchen. Das folgende Beispiel stammt aus einem Computer–Magazin:

> „Macromedia Director galt schon immer als extrem leistungsfähiges, aber nicht leicht zu bedienendes Autorensystem. Vor allem das Bewegen von Animationsobjekten wurde durch die strenge Frame–Orientierung der Software bislang unnötig erschwert. Mit Version 6.0 der Software führt Macromedia Frame-übergreifende Objekte, sogenannte Sprites, ein, die die Bewegungen eines Cast–Members (Darsteller) über mehrere Frames hinweg definieren. Dazu lassen sich innerhalb der Lebenszeit eines Sprites beliebig viele Keyframes setzen, die den Zustand des Sprites in bestimmten Frames beschreiben. Position, Größe, Deckung und weitere Parameter sind so punktuell zu bestimmen. Dazwischenliegende Veränderungen des Objekts wie Bewegung oder Skalierung berechnet die Software per Tweening automatisch. So lassen sich auch komplexe Animationen einfach per Maus erstellen. Drei neue Hilfsmittel unterstützen die Arbeit mit Sprites: Die Parameterleiste Sprite Inspektor bietet direkten Zugriff auf die wichtigsten Eckdaten eines Objekts. Die optional einzublendenden Sprite Paths zeichnen den Keyframes-verbindenden Weg eines Objekts als Vektorpfad nach. Sprite Overlay blendet die Parameter des aktivierten Objekts direkt in die Bühne des Director–Films ein. Auch zum Thema ‚Scriptless Authoring' gibt's Neuigkeiten: Mit Director 6.0 können auch jene Autoren interaktive Elemente anfertigen, die mit der Scriptsprache Lingo nichts

> am Hut haben. Zu diesem Zweck wurden Behaviors eingeführt — eine Sammlung vordefinierter ‚Verhaltensweisen', die einem Sprite oder Frame per Drag and Drop zugewiesen werden können — das notwendige Lingo erzeugt die Software automatisch. Die dritte Modernisierung betrifft die Internet–Funktionen. So ist Shockwave komplett in die Entwicklungsumgebung integriert. Und dank Streaming Shockwave lassen sich Filme nun bereits übers Netz abspielen, während im Hintergrund noch der Download von Media–Elementen wie Bildern und Sounds läuft. Auch der Lingo-Befehlssatz erfuhr einige Erweiterungen: So kann man nun problemlos auf beliebige Internet–Ressourcenzugreifen, und auch die Kommunikation eines Shockwave–Films mit seiner Umgebung wurde deutlich verbessert. Alles in allem liegt mit Version 6.0 eine äußerst gelungene und umfangreich renovierte Neuauflage des Authoring–Klassikers vor — und zwar in zwei Formen: Für Puristen gibt's das einfache Director-Paket. Anspruchsvolle Autoren hingegen entscheiden sich für die Multimedia–Studio–Variante. Die enthält neben Director Xres für die komfortable Bildbearbeitung, Extreme 3D fürs fotorealistische Modelling und Rendering in der dritten Dimension sowie Sound Forge XP von Sonic Foundry für die komfortable Audio–Bearbeitung."[48]

Merkmalskataloge zählen in der Regel wünschenswerte oder positiv bewertete Elemente von Lermitteln oder Autorensystemen auf. Sie werden dann als Bezugsrahmen für die Beurteilung weiterer Lernsoftware oder Autorensysteme herangezogen (das Problem der Softwarebeurteilung bzw. –auswahl wird somit gezähmt). Zu prüfende Programme werden untersucht, indem einzelne Merkmale mit geeigneten Mitteln auf Vorhandensein bzw. Quantität und/oder Qualität analysiert werden. Teil– und Gesamtbewertungen werden verfaßt.

Ein Beispiel für eine sehr frühe Publikation dieser Art ist Issing und Tobers *Autorensysteme für die Entwicklung computergestützter Lernprogramme*[49]. Dieser Text stammt aus dem Jahr 1988, als übliche Rechner–Ausstattungen noch nicht multimedia–tauglich waren und Autorensysteme über eine Erweiterungskarte im Computer externe Medien wie z.B. Videorecorder ansteuern mußten.

Bewertungen anhand von Merkmalskatalogen sind vergleichsweise aufwendig und dementsprechend teurer als Rezensionen in Textform. Sie gestatten einen

[48] Matthias Fichtner: Gesteuertes Verhalten. In: PC Professionell, Ausgabe Oktober 1997, S. 58

[49] Issing, Ludwig K. und Tober, K: [26]

genaueren Vergleich von alternativen Technologien, können damit als Hilfe bei der Auswahl dienen und stellen mit der Formulierung von Merkmalslisten (subjektive) Orientierungshilfen über die jeweils als vorrangig bewerteten Anforderungen an die untersuchte Produktgattung dar. Werden Kriterienkataloge als Designkriterien für Lernsoftware und Autorensysteme verwendet, entsteht aus der Abgeschlossenheit dieser Listen wieder das Problem mangelnder Anpaßbarkeit beim Eintreffen unvorhergesehener Unterrichtsvariablen. Darüber hinaus bieten Kriterienkataloge keine Möglichkeiten zu Neuerungen im Unterricht oder im Softwaredesign: Sie sind hilfreich in der Kommunikation aber nutzlos für Innovation. Ein anderes Problem dieser Listen ist die mangelnde *Operationalisierung* der geforderten (oder verworfenen) Kriterien. Oft werden Kriterien wie *Nutzerfreundlichkeit*, *Nutzerorientierung*, *Interaktivität* oder *Adaptivität* genannt und dabei verkannt, daß es sich hier nicht um Designelemente handelt sondern um praktische Auswirkungen von Designkriterien, die aber zumeist nicht analysiert werden. Die für das Softwaredesign eigentlich interessante Frage ist: *Was* macht ein Programm nutzerfreundlich, adaptiv, etc. Einen typischen Merkmalskatalog dieser Art stellt die folgende, von Bodendorf aufgestellte Liste *Anhaltspunkte zur Beurteilung eines Autorensystems*[50] dar.

Hardwareunterstützung
- Hardwareunabhängigkeit: Nutzung von vorhandener Ausstattung
- Unterstützung von Eingabemedien, z.B. Mouse, Touchscreen, Markierungsstift, Graphiktablett
- Unterstützung von Speichermedien, z.B. interaktive Bildplatte, Videorecorder, Audioperipherie, optische Informationsspeicher, CD-ROM.

Handhabung
- Dialogsteuerung, z.B. durch Menüs, Fenstertechnik, Pull–down–Funktionen
- Funktionsumfang der Text–, Graphik–, Bild–, und Toneditoren
- Präsentationsform der Editoren (WYSIWYG, What you see is what you get)
- Umfang und Erlernbarkeit der Autorensprache
- Testhilfen, z.B. Simulation des Schülermodus, Debugger
- Systemdokumentation
- Lernprogramm zur Einführung in das Autorensystem
- Musterkurse, Schulungsangebot

Bildschirmgestaltung
- Freie Bildschirmeinteilung

[50] Bodendorf, Freimut, ebd., S. 87

- Beliebige Texteingabe, freies Zeichnen
- Unterstützung von Standardgraphiken, Symbolbibliotheken
- Beliebige Positionierung von Eingabefeldern

Interaktionsformen
- Definition von Einsprungstellen in das Lernprogramm
- Fortgesetztes Vorwärts– und Rückwärtsblättern
- Frei belegbare Funktionstasten
- Definition von Benutzermenüs (Menügenerator)
- Unterschiedliche Fragetypen, z.B. Ja–Nein–Frage, Multiple Choice–Frage, Lückentests, Freitexteingabe
- Definition und Formatierung von Rückmeldungen, z.B. Nutzung der Fenstertechnik
- Bereitstellung von Hilfetexten, Lexikonerstellung und –einbindung

Befehlsumfang
- Mächtigkeit und Erlernbarkeit der Autorensprache
- Anzahl und Art der Systemvariablen
- Definition von freien Variablen
- Arithmetische Ausdrücke, mathematische Funktionen
- Bildpräsentationstechniken, z.B. Überblendung, Fade–in, Fade–out, Animationen
- Möglichkeiten der Antwortaufbereitung und –analyse
- Makro– und Unterprogrammtechnik mit Parameterübergabe

Softwareschnittstellen
- Möglichkeiten der Einbettung von Fremdprogrammen
- Einbindung eines getrennten Lernprogramms
- Zugriff auf externe Dateien und Datenbanken

Anwendungskontrolle
- Individuelle Test– und Übungsauswertung für den Lernenden
- Zeitermittlung
- Lernerstatistiken und Lernweganalysen
- Lernerfolgskontrollen für den Autor

Obwohl Merkmakskataloge bei der praktischen Auswahl von Software sicherlich hilfreicher sein können als in der Entwicklung von Lernsoftware und Autorensystemen, distanziert sich Fricke auch von dieser Strategie. Er weist auch auf die häufige Alibi–Funktion hin, die die Befolgung von Kriterienkatalogen bei der Anschaffung von Software spielen kann. Sowohl die jeweiligen Entscheidungsträger als auch die Hersteller der Kriterienkataloge gehen offenbar von einer Zähmbarkeit des Auswahlproblems aus.

„Die Durchsicht einer Checkliste, d.h. die Überprüfung, ob das Pro-

> gramm die erforderlichen lernwirksamen Parameter besitzt, würde genügen. Von dieser Vision sind viele Verantwortliche, die über den Kauf und den Einsatz multimedialer Instruktionsprogramme zu entscheiden haben, fasziniert. Die vorhandenen Checklisten [..] sind jedoch in der Regel nicht empirisch validiert worden."[51]

Mit *empirischer Evaluation* wurde nun auch das dritte verbreitete Verfahren der Feedback–Erzeugung genannt: Die *experimentell–empirische Messung* von Lernerfolgen ist das aufwendigste der drei Beurteilungsverfahren. Hierbei werden je nach untersuchtem Gegenstand Lernergruppen oder alternative Softwarelösungen als unabhängige Variable verschiedenen Bedingungen ausgesetzt und der jeweils erzielte Lernerfolg gemessen und verglichen. Dieses Verfahren basiert auf der Anwendung experimenteller Forschungsethoden der neuzeitlichen Wissenschaften auf die Unterrichtsforschung, die erstmals zu Beginn des 20. Jahrhunderts durch Lay und Meumann vorgenommen wurde. Sie fordern die systematisch herbeigeführte Beobachtung von Lehre und Unterricht unter vereinfachten Bedingungen mit dem Ziel, allgemeingültige Erkenntnisse nach dem Vorbild der Naturwissenschaften formulieren zu können [52].

Lernen unterliegt natürlich nicht den Gesetzen und Vorhersehbarkeiten, denen Phänomene der Physik folgen. Ein Beispiel hierfür ist das wissenschaftliche Kriterium der Widerholbarkeit, das in der Physik, nicht aber in der Didaktik erfüllt werden kann. Experimentelle Untersuchungen von Lernprozessen (wie sie beispielsweise in der Lernpsychologie üblich sind) haben daher einen *deskriptiven*, nicht jedoch einen *prospektiven* Aussagewert für individuelle Lernprozesse.

---

[51] Fricke, Reiner: Evaluation von Multimedia, S. 410; In: Issing, Ludwig J. und Klimsa, Paul: Information und Lernen mit Multimedia [25], S. 401-413

[52] vgl.König, Eckard und Zedler, Peter: [28], S. 40-41

Kleider, Fahrzeuge, Werkzeuge, Gebäude und computergestützte Lernumgebungen sind Beispiele für Designprodukte. Sie werden von Menschen unmittelbar im täglichen Leben angewendet und können nur dort umfassend und realistisch ihre Qualität und ihre psychologischen, sozialen, gesundheitlichen usw. Auswirkungen zeigen bzw. beweisen. Vor diesem Hintergrund erweisen sich empirische Evaluationen in zwei Punkten als problematisch: Zum einen formulieren deratige Untersuchungen ihre Ergebnisse als Aussagen über Gruppen. Dort, wo Design jedoch möglichst individualisierte Lösungen bereitstellen möchte (dies strebt Unterricht in der Regel an), sind Entwerfer eher an Aussagen über einzelne Nutzer interessiert. Individuen kommen darüberhinaus oft zu sehr unterschiedlichen Urteilen über Entwurfsqualitäten. Zweitens erfordern kontrollierte empirische Messungen eine Varianz oder bewußte Variation von Variablen, also potentiell eine Inkaufnahme oder sogar bewußte Verschlechterung von Komfort, Gesundheit, Lernen etc. untersuchter Teilgruppen.
So verbietet es sich zum einen moralisch, beispielsweise die gesundheitliche Verträglichkeit von Textilien oder die sozialen Auswirkungen von Wohnstrukturen durch gezielte Variation experimentell in der realen Anwendung zu untersuchen. Zum anderen gleichen sich per Definition keine zwei Entwurfsaufgaben. Hieraus ergibt sich eine Notwendigkeit für Innovation. Die Übertragung empirisch gewonnener Erkenntnisse auf neue Entwurfsprobleme stellt somit zwangsläufig eine fahrlässige Marginalisierung des Entwurfsproblems dar.

In experimetellen Situationen sind reale Anwendungskontexte von Unterricht(–smethoden) oder anderen Designprodukten nur sehr schwer nachzubilden. Rationalisierte Methoden versuchen, ihre in überwachten Situationen gewonnenen Erkenntnisse auf neue Kontexte zu übertragen. Dieses Vorgehen impliziert ein naives Verständnis von Entwurfsprozessen, das eine wissenschaftlich–rationale Vergleichbarkeit und Wiederholbarkeit von Designprozessen voraussetzt. Diese sind jedoch nicht gegeben.

Unterricht und Design gleichen einander in ihrem jeweiligen wissenschaftlichen Sonderstatus. Grundlegende Bedingungen der Wissenschaftlichkeit, wie die Wiederholbarkeit und Vergleichbarkeit von Ereignissen, wie sie in anderen Disziplinen gelten, versagen vor bösartigen Problemen. Ein physikalisches Phänomen wie z.B. eine Pendelschwingung mag einfach wiederholt oder mit

einem anderen verglichen werden. Design hingegen kann weder wiederholt noch objektiv verglichen werden — so wie Unterricht bzw. das Lernen.

2.8.2. Nützlichkeit und Verläßlichkeit von Autorensystem–Merkmalen

[Ich möchte] Design–Richtlinien lieber unkommentiert stehen lassen, denn Sinniges steht neben Unsinnigem, Nicht–Operationalisierbares neben Operationalisierbarem, Spezielles neben Allgemeinem, kurz, der Teufel steckt im Detail. Ich habe den Eindruck, daß man mit guten Beispielen mehr bewirken kann als mit allgemeinen Design–Guidelines.[53]

Planungs– und Entwurfsprozesse gehen ihren Produkten und deren Anwendungen zeitlich zwangsläufig voraus. Es ist nicht möglich, irgendetwas zu verwenden oder es zu untersuchen, bevor es entworfen und auf die eine oder andere Art ausformuliert wurde. Dies trifft auf bösartige Probleme ebenso zu wie auf zahme. Im Zusammenhang mit bösartigen (und daher einzigartigen) Problemen ergeben sich jedoch grundsätzliche Schwierigkeiten in bezug auf die Vorhersehbarkeit von Angemessenheit und Designqualität der jeweiligen Produkte. Wie soll man das Gelingen von Lösungen garantieren, wenn sich dies nur in zukünftigen praktischen Anwendungen ermitteln läßt? Daß bösartigen Problemen und ihren Lösungen soziale Wirkungen anhaften, erleichtert diese Situation ebenso wenig wie die Tatsache, daß es hier keinen normalen Designprozeß sondern eine zweistufige Design–Kaskade von Autorensystemdesign und anschließendem Lehrmitteldesign zu bewältigen gilt.

Wenn Unterricht entworfen wird, werden Vorhersagen (also *hypothetische Aussagen*) über die zu erwartenden Wechselwirkungen zwischen Medien, Methoden und Lernern sowie deren Interaktionsprozessen (aus den genannten Quellen von Feedback–Informationen) aufgegriffen oder neu formuliert. Jede Designentscheidung ist in diesem Prozeß eine Aussage bzw. eine Vorhersage über ihre eigene

[53] Schulmeister, Rolf: [55], S. 364 (Diskussion eines Beispiels für Design–Richtlinien von Thurmann aus dem Jahr 1993)

Nützlichkeit: Eine Lösung oder Teillösung wird angestrebt, wenn sie sinnvoll erscheint oder verworfen, wenn sie zur Erreichung von Zielen wenig geeignet erscheint.

Die Formulierung dieser Aussagen kann prinzipiell in Form *negativer* oder *positiver Sätze* erfolgen. Negative Aussagen beschreiben Handlungen, die nicht durchgeführt werden sollten, um Unterrichtserfolg zu erzielen, positive Aussagen beschreiben hingegen Handlungen, die durchgeführt werden müssen, um Unterrichtserfolg zu erzielen. Eine visuelle Entwicklungsoberfläche kann im Fall von Autorensystemen allerdings nur *positiv* den Zugriff auf *verfügbare Handlungsoptionen* eröffnen. *Nichts* tut jeder Computer ganz automatisch und ohne jegliche Anweisungen. Jeder Befehl, der das Agieren von Computern auslöst und steuert, ist daher eine positive Aussage vom Typ *tue x*. Modelle didaktischen Handelns, die von Autorensystemen unterstützt werden, haben daher immer den Carakter positiver Annahmen über ihren didaktischen Erfolg. Aus wissenschaftstheoretischer Sicht sind jedoch nur negative, ausschließende Sätze verläßlich.

Ich stelle auf den folgenden Seiten dar, inwiefern der *Wahrheitsgehalt* positiver Aussagen und negativer Falsifikationen voneinander abweichen. Das Resultat dieser Ausführungen ist eine Situation, in der nur negative Vorhersagen als verläßlich anerkannt werden, die Technologien zur Lernmittel–Herstellung jedoch einzig die Implementierung positiver Anweisungen zuläßt. Es wird folglich ein Mittel benötigt, um die Reliabilität der als *vage* positive Aussagen einzustufenden Autorensystem–Merkmale so gut wie möglich zu unterstützen. Ich schlage zu diesem Zweck das Heranziehen von didaktischem Kontextwissen vor, über das nur der Lehrer und nicht der Autorensystementwickler verfügt. Lehrer sollten infolgedessen eine aktive Rolle nicht nur bei jeder individuellen Anwendung sondern bei der Entwicklung und Veränderung von Autorensystemen spielen. Das Resultat ist praktisch ein Verschwimmen der Grenze zwischen Autorenwerkzeugen und Lernumgebungen.

Aufbauend auf dem Newtonschen Weltbild ist in unserem neuzeitlichen Wissenschaftsverständnis eine Theorie genau dann wahr, wenn sie mit einem Fakt in der Realität korrespondiert. Darauf aufbauend haben sich in der Vergan-

genheit die drei Paradigmen *Naiver Empirismus*, *Logischer Empirismus* und *Kritischer Rationalismus* einander abgelöst, die dieses Theorieverständnis auf unterschiedliche Weisen interpretieren[54].

Der *Naive Empirismus* nimmt zunächst an, daß alle Erkenntnis aus der Erfahrung kommt und fordert, daß das Beobachten der Welt, das der Erkenntnisgewinnung vorausgeht, frei von Theorie und Erkenntnisinteresse erfolgt, um schließlich über die Methoden der *Generalisierung*, *Abstraktion* und *Isolierung* zu „wahren Erkenntnissen“ über die Natur, sogenannten *Naturgesetzen*, zu gelangen. Dabei werden diese Naturgesetze als etwas in der Natur gegebenes, unabhängig vom Menschen existierendes angesehen. Ein erziehungswissenschaftliches Beispiel, in dem sich diese Position manifestiert, ist die *Programmierte Unterweisung*. Diese entwirft Unterricht auf der Basis von Erkenntnissen der Verhaltens– und Lernpsychologie und faßt diese in vorformulierten Arbeitsschritten, sogenannten *Programmen*, die in weitreichender Abwesenheit von Lehrenden zu bearbeiten sind. Die Abarbeitung eines solchen (als nicht–linearer Text oder Computer–Software implementierten) Programms ist ein gezähmtes Problem, das gelöst ist, wenn das Programm durchlaufen und der Lernerfolg nachgewiesen werden kann. Auf dieser Basis wurde die Programmierte Unterweisung in den 1960er Jahren zum ersten breiten Anwendungsfeld des digitalen Computers im Unterricht.

Der *Logische Empirismus* setzt voraus, daß Forscher unter *Rückgriff auf theoretische Vorstellungen* die Natur beobachten, wodurch formulierte Erkenntnisse nicht mehr als Abbildung einer von der Theorie unabhängigen Wirklichkeit interpretiert werden. Wissenschaft ist in dieser Sichtweise „ein System von Sätzen, die im Verhältnis von Begriffspyramiden oder axiomatischen Systemen zueinanderstehen und nach dem Prinzip der Widerspruchsfreiheit aufgebaut sein müssen“[55]. Somit werden, orientiert an die Methoden der Physik, Sätze der

[54] Die drei genannten Ansätze sind nicht die einzigen wissenschaftstheoretischen Ansätze, die unterrichtliche Erkenntnis und Theoriebildung betreffen. Jedoch erscheinen sie mir in diesem Zusammenhang und in dieser Auswahl als am besten geeignet, das Problem der Integration didaktischer Theorien in Lehrwerkzeuge zu diskutieren.

[55] Holzkamp, K. nach: Walter, Hellmuth, ebd. S. 20

formalen Logik als analytisches Wissen zugelassen und der prognostische Wert einer Hypothese um so höher eingeschätzt, je häufiger es gelingt, diese empirisch zu bestätigen. Dies bedeutet, angewendet auf die Unterrichtsforschung, daß eine Lerntheorie als um so zuverlässiger angesehen wird, je mehr Hypothesen aus ihr abgeleitet werden können, die ihrerseits empirisch bestätigt werden können. Diese Bestätigung erfordert jedoch das Vorhandensein robuster formaler Bezugsrahmen, die in unterrichtlichen Kontexten nur schwer konstruiert werden können.

Die sowohl dem Naiven Empirismus als auch dem Logischen Empirismus implizite Annahme, man könne wahre Aussagen über die „Wirklichkeit“ machen, das sogenannte *Induktionsprinzip*, wird vom *Kritischen Rationalismus* abgelehnt und durch das Prinzip der Falsifikation ersetzt. Wissenschaft, so Walter, schreitet diesem Ansatz zufolge nicht etwa durch abgesicherte „positive“ Aussagen über die Wirklichkeit fort, sondern vielmehr durch *Eliminierung* von an Empirie gescheiterten Hypothesen, weil wissenschaftliche Sätze weder Wahrheit noch Wahrscheinlichkeit erreichen können[56]. Es sei lediglich möglich, mittels logischer Schlüsse von „wahren“ besonderen Sätzen auf die Falschheit von *allgemeinen* Sätzen zu schließen. Weiterhin müssen wir aus Sicht des kritischen Rationalismus deskriptive, also beschreibende Sätze und präskriptive bzw. normative Sätze unterteilen. Letztere unterscheiden sich von rein beschreibenden Aussagen durch enthaltene Wertvorstellungen und Sollaussagen. Sie beruhen auf subjektiven Urteilen und werden daher als *außerwissenschaftlich* betrachtet. Die meisten unterrichtlichen Ziele und Theorien sind jedoch normative Sätze. Ein vielzitiertes Beispiel für den kritisch–rationalistischen Ansatz ist die Tatsache, daß nach Sichtung ausschließlich weißer Schwäne die Aussage „Alle Schwäne sind weiß.“ nicht haltbar ist, da ihre Wahrheit keineswegs bewiesen ist. Nach der ersten Sichtung eines schwarzen Schwans kann jedoch die Widerlegung des Satzes als gesicherte Erkenntnis gelten[57].

[56] K. R. Popper In: Walter, Hellmuth, ebd. S. 22

[57] Vor diesem Hintergrund lohnt es sich, noch einmal kurz über das Design unserer gegenwärtigen GUIs nachzudenken, denen ganz offensichtlich eine naiv–empiristische Theorie zugrundeliegt: Die sich in vielen Software–Oberflächen manifestierende Aussage „Alle Nutzer beherrschen das Doppelklicken.“ z.B. kann nicht bewiesen, sehr wohl aber widerlegt werden.

Lehrpraktiker, die sich von der Unterrichtsforschung oder der Lernpsychologie nun konkrete Handlungsanweisungen für ihre Arbeit erhoffen, erwarten Aussagen vom Typ „wenn Bedingung X der Fall ist, mußt Handlung Y ausgeführt werden um Ziel Z zu erreichen“. Aussagen dieser Art können aber angesichts der beschriebenen Unmöglichkeit positiver wahrer Aussagen nicht erwartet werden. Grell und Grell begründen diese Unmöglichkeit der erhofften Aussage–Typen weiterhin mit den folgenden beiden Argumenten: Einerseits seien Handlungen und Situationen, die auf irgendeine Weise vergleichbar sind noch lange nicht gleich. Andererseits reagieren unterschiedliche Menschen unterschiedlich auf dieselben Reize (Problem der Konstanzannahme in der Wahrnehmungspsychologie)[58]. Wir haben es im Unterricht mit Menschen zu tun und so ist es nicht verwunderlich, daß Aussagen dieser gewünschten Qualität bereits daran scheitern müssen, daß sie nicht absolut sondern lediglich unter Erwähnung definierter Wahrscheinlichkeiten formuliert werden können.

Darüber hinaus sind Lehrpraktiker eigentlich an Prognosen interessiert, die für individuelle Schüler gelten und nicht für Gruppen. Da die in die Handlungsanweisung eingegangenen Erkenntnisse in Situationen gewonnen wurden, die mit der späteren Unterrichtssituation eines Lehrpraktikers nicht identisch ist, muß das präskriptive Versprechen, daß ein Ziel durch empfohlene Handlungen erreicht werden könnte, als normativ, unwissenschaftlich und letztlich haltlos bzw. unbewiesen bezeichnet werden.

Es ist weiterhin weder sicher, ob die in einer Theorie formulierten Rahmenbedingungen mit solchen realer Unterrichtssituationen übereinstimmen, noch ob sich die in einer Handlungsanweisung empfohlenen Methoden mit denen in der jeweiligen Unterrichtssituation angestrebten vereinbaren lassen, noch ob die Kommunikation der Theorie und ihrer Implikationen zwischen Forscher und Praktiker geglückt ist, noch ob das angestrebte Schülerverhalten beobachtbar oder auf andere Weise feststellbar ist usw.

Wenn Wissenschaft Praktikern also weder handlungsanweisende Sätze an die Hand geben noch positive Aussagen über die Wirkung von Handlungen treffen,

[58] Grell, Jochen und Grell, Monika: [23], S. 22-44

sondern lediglich Hypothesen widerlegen kann, und letztlich nur aufzeigen kann, was in der Praxis *nicht* zum Ziel führt, ist das Produkt einer entwerferischen Tätigkeit, also jegliche Innovation inklusive Unterricht, nie vollends das Resultat streng wissenschaftlicher Arbeit. Hier spielen das Versuchen, die Intuition und die Kreativität wichtige Rollen.

Die Tatsache, daß lediglich positive Hypothesen formal widerlegen zu können, anstatt sie auch in präskriptiven Handlungsanweisungen formulieren zu können, widerspricht den Erwartungen von Praktikern an die Theorie, die diese Erwartung enttäuschen muß:

> „Letztlich liefern didaktische Modelle und Theorien [..] keine allgemeingültigen Handlungsanweisungen, sondern vielmehr mögliche Kriterien, die bei didaktischen Entscheidungen handlungsleitend sein können.“[59]

Flechsig betont den retrospektiven Charakter didaktischer Modelle, indem er sie als *Rekonstruktionen von Unterrichtswirklichkeit zweiter Ordnung*[60] bezeichnet. Ein didaktisches Modell sei eine Zusammenfassung vieler einzelner unmittelbarer Praxisbeschreibungen, die in einem späteren Schritt aufgrund gemeinsamer Elemente zu einer Klasse zusammengefaßt werden. Prospektiv können didaktische Modelle keine Aussagen über ihr Erfolgspotential in der unterrichtlichen Anwendung treffen. Von Autorensystemen unterstützte Modelle didaktischen Handelns sind nichts anderes als die von Schrittmatter/Mauel und Flexig beschriebenen didaktischen Modelle. Durch ihren gleichzeitigen Charakter als weitgehend alternativlose positive Annahme über ihre generelle Angemessenheit und ihren unterrichtlichen Erfolg handeln sie jedoch den rein retrospektiven Qualitäten didaktischer Modelle zuwider. Es stellt sich die Frage, unter welchen Umständen in Software implementierte didaktische Modelle prospektiv und verantwortbar als positive Aussagen über ihren unterrichtlichen Erfolg akzeptiert werden können.

[59] Schrittmatter, Peter und Mauel, Dirk: Einzelmedium, Medienverbund und Multimedia, S. 48; In: Issing, Ludwig J. und Klimsa, Paul, ebd., S. 47-61

[60] Flechsig, Karl–Heinz, [18], S. 5-9

Andrew Nolan, Softwareingeneur bei Rolls Royce, bescheinigt positiver Indikation und Falsifikation in seinem Ansatz „Learning from Success“ genau umgekehrte Potentiale im praktischen Entwurf, was dem Standpunkt des Kritischen Rationalismus zunächst zu widersprechen scheint. Sein Ansatz ist eine Reaktion auf das Verfahren der Qualitätssicherung, Produkte und Prozesse durch deren fortwährende, kontrollierte graduelle Veränderung zu verbessern, indem beobachtete negative Konsequenzen zum Verwerfen experimenteller Veränderungen und zur experimentellen Variation anderer Prozeß–Variablen führen. Nolan empfiehlt im Gegensatz dazu, den *Erfolg* herbeigeführter Variation als auf den Anwendungskontext bezogen gesicherte Erkenntnis zu betrachten und darauf aufbauend Handlungsanweisungen für die Praxis zu konstruieren:

> „Failure and success provide different perspectives on improvement: failure tells you what you do not want to do in the future, whereas success tells you exactly what you should do again. Moreover, success leaves clues that provide a more direct route to identifying processes that can lead to a higher level of success in the future.“[61]

Nolans Ansatz ist pragmatisch, leicht nachvollziehbar und einmal mehr ein Beispiel dafür, welche Rolle gesunder Menschenverstand beim Planen und Entwerfen spielt. Und trotzdem steht er in einem scheinbaren Widerspruch zum Aussagewert, den der Kritische Rationalismus positiven und negativen Sätzen zuerkennt. Er bedarf daher einer näheren Betrachtung.

Der Grund für diesen Widerspruch liegt in einem Unterschied zwischen Planungs– und Entwurfsmodellen aus wissenschaftlicher und angewandter, industrieller Entwicklung. Während in wissenschaftlichen Kontexten typischerweise eine personelle, räumliche und zeitliche Trennung zwischen forschenden und anwendenden (entwerfenden) Beteiligten besteht, kann diese Trennung in industriellen Kontexten vergleichsweise leicht aufgebrochen werden. Im Idealfall sind Forscher, Evaluierende und Anwender hier in einer kleinen Gruppe oder nur einer einzigen Person (die mehrere Rollen einnimmt) vereint. Man kann

[61] Nolan, Andrew J.[40], S. 97

davon ausgehen, daß hier alle Beteiligten mit der konkreten Anwendungsproblematik und praxisrelevanten Kriterien vertraut sind. In diesem Fall kann die wissenschaftstheoretische Kritik an normativ–deskriptiven Sätzen weitgehend ausgeräumt werden, da der Entwerfer hier in seiner Forscher–Identität einerseits mit dem Anwendunkskontext und in seiner Anwender–Identität andererseits mit dem Forschungskontext bestens vertraut ist. Das Problem enthaltener Wertaussagen und das Fehlen eines objektiven Bezugssystems stellt kein prinzipielles Problem dar, wenn man dem Entwerfer ein eigenes kontextbezogenes Urteil auf der Basis seines subjektiven Bewertungssystems zugesteht.

In solchen fällen ist die Anwendung von Erkenntnissen auf ein Design weniger kritisch als der sich anschließende Prozeß, in dem aus der Anwendung des Designprodukts Erkenntnisse über dessen Qualität gewonnen werden und dann selbst ihren Weg in neue Designprozesse finden, also die Erzeugung von Feedback. Während die in Abschnitt 2.8.1 diskutierten Formen von Dritt–Parteien–Feedback davon ausgehen, daß neben Designer und Anwender als dritte Instanz der evaluierende Wissenschaftler (oder Journalist, Softwaretester etc.) steht, um Erkenntnisse für spätere Designprozesse zu sammeln, kommt das industrielle Bild der Feedbackerzeugung, wie Nolan es darstellt, ohne diese formal vom Designprozeß entkoppelte Instanz aus. Im industriellen Entwurf sind Entwerfer und Evaluierende institutionell, kulturell und (fach–)sprachlich eng aneinander gekoppelt und sehen sich lediglich mit der Aufgabe konfrontiert, Aussagen über ihren jeweiligen *Problembereich* machen zu müssen. Wissenschaftliche Evaluation setzt sich hingegen zum Ziel, abgekoppelt von konkreten Entwurfsproblemen, Aussagen über *die Welt* zu machen. Positive Sätze oder Handlungsanweisungen wie „alle Schwäne sind weiß“ verbietet der Kritische Rationalismus der wissenschaftlichen Evaluation. Diese reagiert darauf durch die (formal akzeptierte) Formulierung abgesicherter (falsifizierter), negativer Aussagen vom Typ „nicht alle Schwäne sind weiß“, oder aber durch schlichtes Ignorieren des kritisch–rationalistischen Gebots. Letzteres lann sich z.B. in der Formulierung der bereits diskutierten Kriterienkataloge äußern, also in Form von Sammlungen positiver Sätze bzw. Aussagen über *die Welt* auf der Basis positiver axiomatischer Annamen. Um es auf den Punkt zu bringen, kann man sagen: Wenn Feedback

durch den Entwerfer selbst evaluiert wird, müssen konkrete Probleme nicht zwingend generisch gelöst werden, während wissenschaftliche Evaluationen zu generischen Aussagen kommen müssen, die nur in Form widerlegter Thesen als negative Sätze Gültigkeit beanspruchen können.

Die Interaktionsformen, die von Autorensystemen unterstützt werden, sind typisierte und generalisierte Beschreibungen didaktischen Handelns. Insofern entsprechen sie *didaktischen Modellen*. Flechsig [18] bezeichnet didaktische Modelle als abstrakte *Rekonstruktionen von Unterrichtswirklichkeit*, die unterrichtliche Interaktionen rückblickend typisieren. Prospektiv können abstrakte didaktische Modelle jedoch weder konkrete Handlungsanweisungen noch eine Garantie für ihr Gelingen geben. Dennoch versuchen Autorensysteme als Bibliotheken didaktischer Interkationsmuster, Modelle *retrospektiv* rekonstruierter Realität computervermittelten Unterrichts *prospektiv* auf neue Unterrichtssituationen zu übertragen.

Was bedeutet dies nun für die Lehrpraxis? Der Lehrer in der Praxis kennt seinen Unterricht, also sein Produkt, er beobachtet dessen Qualität in bezug auf seine Lernergruppe, eine ihm wohlbekannte Nutzerschaft. Er selbst wertet Feedback aus seinen Beobachtungen aus und greift dieses in späteren Unterrichtsentwürfen zur Sicherung deren Qualität auf. Dabei trifft er seine Entwurfsideen im Rahmen eines vergleichsweise kleinen Weltausschnitts: Sie müssen lediglich innerhalb seiner jeweiligen Unterrichtssituation für einzelne konkrete Probleme gültig sein. Unterstützende Hinweise und Anweisungen etwa von wissenschaftlicher Seite werden hingegen für weitaus größere Weltausschnitte formuliert und müssen für gesamte Problem–Klassen gelten. Sie wurden in anderen Kontexten ermittelt, die nicht identisch mit irgendeiner weiteren (Lehr–)Situation sind. Daher ist es durchaus möglich, daß sich die in probabilistischen Aussagen dargestellten Effekte in einer konkreten Lernergruppe anders, in extremen Fällen möglicherweise sogar genau umgekehrt verhalten. Jedes Evaluations–Ergebnis einer dritten Partei und jede darauf basierende Empfehlung für die Praxis wird ferner vor dem Hintergrund einer Umwelt formuliert, die laufenden Veränderungen unterworfen ist.

Im Bereich des computervermittelten Unterrichts entwickeln sich z.B. neue Technologien ebenso wie die Ausstattungen der Schulen ständig weiter, und Lehrende treffen mit jedem neuen Jahrgang auf eine neue Generation von Lernern, die anders sozialisiert ist als vorherige Jahrgänge. Die in Abschnitt 2.8.1 von Issing und Tober sowie die von Bodendorf formulierten Kriterienkataloge Beispiele für ein mangelhaftes Verständnis dieses Problems, da die enthaltenen konkreten Forderungen an den Grad möglicher Interaktion und an technische Details deutlich die Sprache ihrer jeweiligen Zeit sprechen und heute kaum mehr relevant erscheinen.

Für das Design von Autorensystemen verschärft sich dieser Mangel an Reliabilität weiter: Das letztliche Kriterium, an dem Lehrwerkzeuge gemessen werden, ist ihre Lehrwirksamkeit. Das Urteil darüber liegt jedoch bei demjenigen, der das Lehrziel definiert: dem Lehrer. Dritt–parteiische Einflußnahmen auf das Design von Autorensystemen sind durch deren Werkzeugcharakter vom eigentlichen Unterrichtsgeschehen weitgehend entkoppelt. Die wäre im Fall bloßer Empfehlungen weit weniger problematisch als der tatsächliche methodische Zwang, der von Designentscheidungen in Autorensystemen ausgeht. Mit Autorensystemen werden die eigentlichen Lehrwerkzeuge schließlich nur hergestellt. Dabei ist zu bedenken, daß Lehrende *gute* Autorensysteme durchaus *schlecht* anwenden können und umgekehrt.

Wenn Lehrende schließlich die Lehrwirksamkeit ihrer hergestellten Lehrmittel beurteilt haben und auf der Basis dieses Urteils dann die Qualität des verwendeten Autorensystems bewerten, steht ihnen kein formaler Kommunikationskanal zur Verfügung, diese Bewertung an den Entwickler des Autorensystems für eine Meta–Auswertung weiterzuleiten. Weiterhin ist es praktisch kaum möglich, die beim computervermittelten Unterricht experimentell variierten Variablen isoliert zu betrachten, um Aussagen über deren Effektivität zu formulieren. So treten z.B. die Auswirkungen der in Abschnitt 2.1 aufgeführten typischen Gründe für Unterricht mit dem Computer nie allein sondern gemeinsam auf. Ein gemessener Effekt kann daher nicht sauber an eine einzige Variable geknüpft werden.

Da Unterrichtsevaluationen zum Lernen mit dem Computer typischerweise den

computervermittelten Unterricht mit dem traditionellen Klassenraum oder dem Lernen mit dem Buch vergleichen, kommen entsprechende Studien immer wieder zu dem Ergebnis, daß die Akzeptanz Lernender gegenüber neuen Lerntechnologien sehr hoch sei, auch wenn meistens keine bessere Lehreffektivität nachgewiesen wird. Dies kann damit begründet werden, daß Lerner, die heute noch vorwiegend traditionellen Unterricht erhalten, den Computer im Unterricht als interessante Abwechslung erleben. Dieser Novitätseffekt und andere Hawthorne–artige Effekte können in Evaluationen nur sehr schwer isoliert und kontrolliert werden, und konsequenterweise ignorieren die meinsen Evaluationen diesen Effekt einfach[62].

Vor dem Hintergrund der aufgeführten (und sicherlich nicht vollständigen) Liste von Faktoren, die die Reliabilität von Designempfehlungen für Autorensystemen mindern, scheint es fraglich, ob Meta–Analysen über die Lehrwirksamkeit von Autorensystemen überhaupt nennenswerte Effekte messen. Solche Untersuchungen speziell zur Lehrwirksamkeit von Autorensystemen sind mir nicht bekannt, aber es gibt einige Meta–Analysen zur Wirksamkeit des computervermittelten Unterrichts im Vergleich zum traditionellen Unterricht. Diese Untersuchungen messen typischerweise tatsächlich keine signifikanten Effekte[63].

Da die Anwendung von Autorensystemen um eine Abstraktionsebene weiter vom Unterrichtsgeschehen entfernt ist als die der in den gängigen Meta–Analysen evaluierten digitalen Lernmittel, kann man davon ausgehen, daß auch Evaluationen von Unterricht auf der Basis von Autorensystem–Einsatz keine Effekte messen würden.

Damit machen Computer und Autorensysteme im Vergleich zu anderen Lehrmedien übrigens keine Ausnahme, denn bereits Meta–Analysen zum Lernen mit anderen Massenmedien (mit zentralisierten didaktischen Strukturen) wie Radio und Fernsehen führten zu ebenso mageren Ergebnissen. Nicholas Negroponte lehnt aufgrund von Problemen dieser Art Evaluationsforschungen generell ab und kommt zu der Aussage:

---

[62] Schulmeister, Rolf: [55], S. 375

[63] siehe Schulmeister, Rolf: [55], S. 379 ff.

> „I have little respect for testing and evaluation in interface research. My argument, perhaps arrogant, is that if you have to test something carefully to see the difference it makes, then it is not making enough of a difference in the first place.“[64]

2.9. Die Grenzen der Interoperabilität

Im Rahmen des *Star Wars*–Programms des Pentagon wurde im Juni 1985 das Space Shuttle eingesetzt, um den praktischen Einsatz von Laserstrahlen (mit denen später feindliche Marschflugkörper im Flug zerstört werden sollten) innerhalb und außerhalb der Atmosphäre zu testen. Zu diesem Zweck sollte ein Laserstrahl von einem hawaiianischen Berg aus in den Weltraum gerichtet, dort von einem Spiegel am Bauch des Space Shuttles zurück auf die Erde reflektiert und dort dessen Intensität, Verzerrungen etc. gemessen werden. Daher mußte sich das fast ausschließlich softwarebasiert navigierende Raumschiff über diesem Berg positionieren. Im Rahmen dieser Exkursion und dieses Experiments wurden als Standard–Maßeinheit für Entfernungen nautische Meilen verwendet. Die verantwortlichen Ingenieure hatten jedoch versäumt, die Höhe des hawaiianischen Bergs (10.000 Fuß) zuvor in nautische Meilen zu konvertieren. Aus diesem Grund erwartete der Autopilot des Suttles die Spitze des Bergs in einer Höhe von 10.000 nautischen Meilen über dem Meeresspiegel, also in einer Umlaufbahn weit außerhalb der des Shuttles selbst. Daher positionierte der Autopilot das Space Shuttle so, daß der Spiegel von der Erde weg in den Weltraum gerichtet wurde. Der Laser traf den schwach reflektierenden Rücken des Schiffs und an der Erdoberfläche wurde keine Reflexion gemessen. Die Kosten für das Experiment (U.S.$15Mio.) gingen verloren. Vermutlich wurde für dieses Experiment nur ein schwacher Laser eingesetzt, da das Shuttle und seine Besatzung wohlbehalten zur Erde zurückgekehrt sind[65].

Interessant an dieser Geschichte ist die Tatsache, daß die Höhe des Berges nicht

[64] Negroponte, Nicholas: Being Digital. In: Schulmeister, Rolf: [55], S. 365

[65] Eine ausführlichere Schilderung dieses Vorfalls kann nachgelesen werden in: Chapman, Gary: Taming the Computer. In: Dery, Mark(Hrsg.): [9], S. 297-319

wirklich inkorrekt angegeben war, sondern nur falsch interpretiert wurde. Diese Fehlinterpretation ist aber nicht auf eine Fehlfunktion der Software zurückzuführen. Sie hat korrekt gearbeitet und das Schiff auf den errechneten Punkt ausgerichtet. Es handelte sich vielmehr um einen Fehler in der Wetware[66]: Es war ein menschliches Versäumnis. Warum müssen Menschen eine derart triviale Konvertierung überhaupt durchführen und warum erkennt der Computer das nicht ganz alleine? Was ist für eine Maschine, die riesige Mengen Programmcode unterschiedlichster Sprachen interpretieren kann, so schwer an der Interpretation der Höhe eines Berges, zumal die angegebene Höhe so offensichtlich falsch war?

Als die *Grenzen der Interoperabilität* können wir die Grenzen bezeichnen, an dem unsere gemeinsamen und individuellen Modelle von der Realität enden und das *Unbekannte* beginnt. Wenn unser Denken diesen Grenzen begegnet, können wir nur noch vermuten, was jenseits dieser Grenze existiert; welche Prozesse, Inhalte und Bedeutungen dahinter (inter)agieren. Programme, die wir schreiben, sind auch Modelle von der Realität — oder besser: sie sind Modelle von Welt–Ausschnitten, von Wissens–Domänen. In einem Programm werden die für ein Problem relevanten Elemente der natürlichen Welt re–modelliert und somit im Computer ein Abbild des Problems und Regeln zu dessen Lösung formuliert. Diese programmierten Modelle sind ebenso endlich wie unsere individuellen kognitiven Modelle von der Wirklichkeit und decken sehr kleine Domänen unseres Wissens ab. Wir Menschen verfügen hier über einen Vorteil, der der Erforschung von künstlicher Intelligenz bis heute Kopfzerbrechen bereitet: Gesunden Menschenverstand (oder auf Englisch: *common sense*) und die Fähigkeit, ihn in jeder Situation assoziativ einzusetzen. Die hierfür notwendige Fähigkeit von Software, Informationen über das Ende des eigenen Weltmodells hinaus zu einer anderen Software so zu kommunizieren, daß sie dort verstanden wird, heißt *Interoperabilität*[67].

[66] Wetware: Menschen bzw. das menschliche Gehirn und Nervensystem in Ergänzung zu „Hardware“ und „Software“.

[67] Die Diskussion mangelnder Interoperabilität in designbezogenen Computer–Anwendungen beruht inhaltlich teilweise aus Fischer, T., Burry, M. und Woodbury, R: [16], S. 145-155

In mehreren Computer–Programmen modellierte Wissens–Domänen können einander berühren oder überschneiden. Wenn dies der Fall ist, bietet sich die Möglichkeit, die jeweilige Software miteinander interagieren zu lassen und dabei die jeweils implementierte Funktionalität erheblich zu steigern. Wenn wir beispielsweise im Unterricht vernetzte Computer dazu verwenden, in einem Test gelerntes Wissen zu überprüfen und in diesem Netzwerk auch ein Programm zur Verwaltung von Schulnoten bereitsteht, bieten diese beiden Modelle sich inhaltlich berührender Weltausschnitte die Möglichkeit, sinnvoll zusammenzuarbeiten. Wenn es in dieser Situation einem Lehrer gelingt, diese beiden Programme miteinander „sprechen“ zu lassen, ist das Resultat mehr als die Summe seiner Teile: Eine sinnvolle Ergänzung der zur Verfügung stehenden Software, die das triviale, zeitaufwendige und fehlerträchtige manuelle Übertragen von Testresultaten überflüssig macht und diese Aufgabe bei jeder weiteren Durchführung dieses oder vielleicht auch anderer Tests übernehmen kann.

Ohne Zweifel könnte nach einem solchen *hack* eine System–Integration mit einer Datenbank zur Verwaltung Lernender eine weitere Steigerung der Funktionalität erreichen: Das manuelle Kopieren von Namenslisten und die Zuordnung von Daten könnten auf eine eindeutige Weise automatisiert werden, weiterhin könnte Software automatisch feststellen, welche Lernenden einen Test zu einem bestimmten Zeitpunkt noch nicht erfolgreich bearbeitet haben. Eine weitere Integration mit einem Mailserver könnte dann benutzt werden, um Lehrenden und Lernenden automatisch Hinweise über ausstehende Test–Teilnahmen zukommen zu lassen. Die Liste der Möglichkeiten, die sich hier ergeben, ist sehr lang und die riesige Menge an Software, die bereits für Bildungszwecke hergestellt wurde, erscheint wie eine weite Landschaft kleiner Wissens–Domänen, die darauf warten, miteinander zu sprechen.

Leider sieht die Realität hier sehr traurig aus. Programme sprechen in der Regel nicht miteinander oder zumindest nicht so, wie Lehrer es sich wünschen, und jedes Programm, das Lehrer für ihre Arbeit benutzen, führt sie sehr bald an das Ende ihrer Welt. Software im allgemeinen und Werkzeuge für den computervermittelten Unterricht im besonderen sind wenig interoperabel und darüber hinaus oftmals wenig kooperativ gegenüber Bemühungen, dieses Manko zu be-

heben. Kommerzielle Softwareprodukte liegen oft nur als binäre Programme vor, deren *Quellcode* für Menschen nicht einsehbar ist und von ihnen daher nicht mehr ohne weiteres verändert werden kann. Anbieter kommerzieller Software haben selbstverständlich ein Interesse daran, daß ihre Produkte nicht sinnvoll mit anderen, möglicherweise von der Konkurrenz angebotenen oder von Lehrpraktikern im Eigenbau entwickelten Produkten arbeiten. Lerninhalte, die mit Autorensystemen hergestellt werden, liegen technisch oft in properitären Formaten vor: Sie sind weder für andere Autorensysteme oder deren Laufzeitumgebungen verständlich, noch gestatten sie, inhaltliche oder methodische Elemente daraus einfach zu extrahieren und unabhängig von der Autorensoftware auf andere Lernsituationen anzuwenden. Datenformate von Autorensystem–Produkten sind, wie das in Abbildung 2.14 gezeigte oftmals nicht selbsterklärend, nicht öffentlich dokumentiert oder sogar rechtlich geschützt, so daß sie nicht oder nur unter großen Anstrengungen in andere Kontexte integriert werden können.

Abbildung 2.14.: Ansicht einer PowerPoint–Datei in einem Text–Editor

Daß sich dieses Bild in jüngerer Zeit zögerlich wandelt, ist auch auf den Erfolg des Internet zurückzuführen. Computernetzwerke sind die potentielle In-

frastruktur, die Lehrenden das Potential der Interoperabilität direkt vor Augen führt. Mit der Entwicklung des Internets stellte sich das Problem der Kompatibilität verschiedener Rechnertypen und Software–Architekturen, und das Resultat war eine sehr rasche und erfolgreiche Verbreitung quasi–interoperabler Datenformate, wie z.B. HTML, das heute auf praktisch jedem Computer durch die Verfügbarkeit erforderlicher Software Unterstützung findet. Die Hypertext Markup Language, die zunächst am Kernforschungszentrum CERN in der Schweiz und dann von einem zu diesem Zweck gegründeten unabhängigen Konsortium[68] entwickelt wurde, ist eine Dokumentbeschreibungssprache und gestattet es Lehrern, Medienelemente und formale Verhaltensweisen aus erstellten Inhalten zu isolieren und wiederzuverwenden. Da HTML in lesbarem (Klar-)Text gespeichert und transportiert wird, können kleine, text–erzeugende Programme HTML dynamisch (auf der Basis vorhandener Kontextinformationen) erzeugen und damit automatisch Individualisierungen von Lernmaterialen vornehmen. Diese Eigenschaften macht HTML (trotz einiger Defizite) nicht nur für Lehrende zu einem attraktiven Behälter von Inhalten: HTML gestattet die Verwendung von Daten auch jenseits der Grenzen der Software, mit der es erstellt und verwaltet wird. Es erleichtert derartige Nutzungsweisen durch seine Lesbarkeit und vollständige Dokumentation durch eine unabhängige Institution.

Dieser Erfolg und das weithin ungenutzte Potential interoperabler Standards führt gegenwärtig zur Bildung diverser Arbeitsgruppen und Spezifikationen, die sich um dieses Potential bemühen: Es wurde eine Reihe von Initiativen gegründet, die Konzepte und Datenformate für interoperablen Austausch von Daten ermöglichen sollen. Im Bereich der Architektur und Baukonstruktion ist dies das IAI aecXML Domain Committee[69] und im Bereich der computervermittelten Lehre das IMS Global Learning Project[70], das in den USA mit Unterstützung aus Politik und Wirtschaft ins Leben gerufen wurde und ein Ableger der Nonprofit–Organisation EDUCAUSE ist, die sich um den Technologie–

[68] Das W3-Consortium: http://www.w3.org/

[69] http://www.aecxml.org.index2.htm

[70] http://www.imsproject.org/

Einsatz in der höheren Bildung bemüht. Praktische Arbeitsergebnisse der beiden genannten Initiativen sind XML-Schemata. XML ist eine Auszeichnungssprache in der Tradition von SGML[71] und HTML und soll als Container für Daten Applikationen gestatten, mit anderen Programmen über die Grenzen ihrer jeweiligen Welt-Modelle hinaus zu interagieren und u.a. als Standardformat für architektonische bzw. didaktische Inhalte zu dienen.

XML eignet sich als Format für die Speicherung, Verwaltung und insbesondere den Transport von Daten. Die Programme, die diese Daten verwalten und austauschen sollen, benötigen dazu entsprechende Import– und Exportfilter. Ein einfaches Beispiel für den Transport didaktischer Inhalte in XML könnte wie folgt aussehen:

```
<?xml-version="1.0"?>
<test>
  <item type="multiple-choice">
    <question>Welche Zahlen sind die Ausgangswerte f&uuml;r die
Berechnung von Julia-Mengen?</question>
    <answer status="right">Komplexe Zahlen</answer>
    <answer status="wrong">Primzahlen</answer>
  </item>
</test>
```

Die Anzahl der Daten–Bezeichner, der sogenannten *Tags* („item“, „answer“) ist frei erweiterbar (daher der Name eXtendable Markup Language), und der Umfang von Inhalten ist nicht durch vorgegebene Strukturen limitiert. Dies ist angesichts des bösartigen Charakters des Entwurfsproblems Unterricht auch sicherlich notwendig, da Lerneinheiten aus nicht vorhersehbaren Einzelelementen nicht vorhersehbarer Anzahl bestehen. Der Aufbau dieses Datenformats ist so einfach, daß er praktisch ohne die Zuhilfenahme besonderer Filterprogramme für Menschen lesbar ist. Die XML–Spezifikation ist für jedermann einsehbar und frei benutzbar. Dementsprechend ist es sehr einfach, Software zu entwickeln, die XML–Daten lesen (und schreiben) kann. Aber kann sie diese auch *verstehen*? In der IMS FAQ[72] heißt es hierzu:

[71] „Standard Generalized Markup Language“: Standardformat zur Beschreibung von Dokumenten, Vorläufer sowohl von HTML als auch von XML.

[72] http://www.imsproject.org/faqs.html

> „[..] learningware will be able to share information such as content meta–data, student profile and performance information, and course–structure.“

Auch wenn dies wie die Lösung des Problems nicht vorhandener Interoperabilität klingt, hat dieses Versprechen einen Schönheitsfehler — nämlich die oben beschriebene Endlichkeit jeder existierneden Wissens–Domäne. Wenn in diesem Zitat von *Information* die Rede ist, wäre der Begriff *data* passender: XML transportiert Daten. Information sind *interpretierte* Daten und ohne Interpretation sind Daten eben nur Daten (wie in Abbildung 2.14 unschwer zu erkennen ist). Software kann Daten nur innerhalb der Grenzen der Wissens–Domänen, die sie modellieren, interpretieren. XML soll aber den Austausch von Daten *zwischen* Domänen ermöglichen. Zum Transport von Daten ist es auch recht gut geeignet, aber nicht zum Transport von Semantik. Ein *Tag*, das innerhalb einer Domäne spezifiziert wird, wird nicht automatisch in jeder anderen Domäne verstanden. In unserem kleinen XML–Beispiel zum Beispiel ist als Attribut der Antworten der Bezeichner „status“ verwendet worden. Dieser Bezeichner könnte in einer anderen Domäne verwendet werden, um Zugriffsrechte oder Aktualität oder eine andere Eigenschaft von answer–Tags auszuzeichnen.

Die semantische Schwäche von XML wird besonders anschaulich, wo reale physische Objekte beschrieben werden, wie in der Architektur (dies möchte aecXML ermöglichen): Was *ist* eigentlich ein Fenster? Ein verglaster Teil einer Wand? Mit dieser Beschreibung sind bestimmt nicht alle denkbaren Variationen von Fenstern abgedeckt. Was ist, wenn die Wand komplett aus Glas besteht? Was ist dann Wand und was Fenster? Letztlich kann sogar nur ein Loch in einer Wand als Fenster bezeichnet werden. In diesem Fall ist ein Fenster nichts und es besteht aus keinen Elementen. Unsere oft unscharfen menschlichen Modelle der Realität in Software abzubilden und dann auch noch zwischen Domänen zu kommunizieren ist ganz offensichtlich schwieriger als erwartet. Daß Daten auch im Bildungsbereich ohne einen definierten Bezugsrahmen nicht eindeutig interpretiert werden können, läßt sich leicht anhand der Tatsache illustrieren, daß beispielsweise der Wert „6“ als Schulnote im deutschen und im schweizerischen Schulsystem zwei völlig gegensätzliche Bedeutungen hat.

Die Hypothese, daß Datenelemente zwischen mehreren Wissensdomänen semantisch gleich und daher einfach aufeinander abbildbar sind, wird als Konvergenz–Hypothese bezeichnet. Ich habe in anderen Kontexten dargestellt, wie problematisch diese Hypothese in der praktischen Einrichtung und Wartung verteilter Online–Datenbanken ist[73]. Es sind mittlerweile mehrere Technologien entwickelt worden, die als Ergänzungen von XML diese Abbildung von Datenelementen und deren Wartung gestatten. Ein sehr früher Ansatz ist LMX („Language for Mapping XML“), entwickelt bei IBM[74]. LMX ist eine Sprache, in der Datenelemente in verteilten Wissens–Domänen aufeinander abgebildet werden können. Diese spezielle Technologie hat sich nicht durchsetzen können und ist ein lebhaftes Beispiel dafür, wie zerbrechlich Innovationen sind, die entworfen werden, bevor andere Technologien, auf denen sie basieren, überhaupt endgültig spezifiziert sind, was gängige Praxis ist. Doch es gibt nun andere Verfahren (z.B. XPointer), mit denen Ähnliches erreicht werden soll. Gemeinsam ist diesen Verfahren, daß keines von beiden *automatisch* funktioniert und daher keines von beiden so pflegeleicht ist, wie sie angepriesen (und benötigt) werden. Datenele-

[73] Fischer, Thomas, Burry, Mark und Woodbury, Robert: [16], S. 151ff.

[74] Maruyama, Hiroshi, Tamura, Kent und Uramoto, Naohiko: [33], S. 114ff.

mente müssen noch immer manuell aufeinander abgebildet werden, da Computer nicht einmal entfernt über Weltmodelle verfügen, die ihnen die semantische Auflösung beliebiger Bezeichner für Datenelemente erlauben könnten.

Weiterhin gibt es Situationen, in denen Autoren von Information daran interessiert sind, *unrichtige* Informationen zu kommunizieren. Ein alltägliches Beispiel hierfür sind Metadaten im WWW (ein mit XML eng verwandtes Verfahren zur Auszeichnung von Daten über Daten). Hier werden in versteckten Stichwortlisten, die Suchmaschinen die Indizierung von Seiten erleichtern sollen, oftmals bewußt Begriffe eingetragen, die mit dem Inhalt nichts zu tun haben, um Aufmerksamkeit zu erregen. XML Metadaten wie das IMS XML metadata schema erfordern ferner die Bezeichnung des „content–types“, also des Medientyps bzw. der enthaltenen Textform. Auch hier gibt es ein Problem in bezug auf die semantische Eindeutigkeit: Warum kann ein Text, der als „letter“ deklariert wird, nicht auch gleichzeitig ein „poem“ sein? Hinzu kommt ein weiteres Problem: Elemente von mehreren Wissens–Domänen kollidieren semantisch potentiell nicht nur zur gleichen Zeit miteinander sondern auch zu verschiedenen Zeiten. Was heute wahr und richtig ist, ist dies nicht auch automatisch morgen. Das nächstliegendste Beispiel hierfür ist die Geschichte der Dokumentbeschreibungs–Sprache HTML. Sie wurde zunächst zur logischen Auszeichnung von Dokumenten entworfen. Die gestalterischen Bedürfnisse der Autoren von WWW-Inhalten haben jedoch dazu geführt, daß sich HTML von Version 1.0 zu Version 4.0 von einer Dokumentbeschreibungs–Sprache zu einer Desktop–Publishing Sprache entwickelt hat und heute weder die eine noch die andere Aufgabe sauber löst. Ein *Tag*, das zunächst Absätze logisch voneinander trennen sollte, wird heute auf unterschiedliche Weisen dafür verwendet, einen Zeilenvorschub zu erzeugen, um beliebige Inhalte wie Textteile, Tabellen und Grafiken visuell voneinander abzusetzen. Programme wie Sprach–Synthesizer für Sehbehinderte verwenden die logische Auszeichnung von Absätzen jedoch als Indikatoren für Pausen beim Vorlesen und fehlinterpretieren nun Befehle, die visuell gestaltend eingesetzt wurden.

2.10. Anstelle eines Beispiel–Szenarios

Autorensysteme erkaufen Nutzerfreundlichkeit durch die Vorgabe statischer Vorlagen. Inhaltliche Offenheit wird somit durch formale Abgeschlossenheit erreicht. Vor dem Hintergrund dieser Abgeschlossenheit wäre es nun sehr einfach, Beispiel–Szenarien und Nutzerdialoge zu konstruieren, in denen einzelne Pakete an den Erfordernissen bestimmter Unterrichtsentwürfe scheitern. Jedes Werkzeug kann als unzureichend dargestellt werden, wenn seine Mängel bekannt sind und entsprechende Testfälle konstruiert werden. Systeme und ihre Schwächen zu kennen und auf dieser Basis ihr Versagen zu exemplifizieren, ist daher weder formal akzeptabel noch praxisnah. Außerdem befinde ich mich mit dem Vorhaben, mich als Autorensystem–Entwickler zu betätigen, in der Rolle eines Dritten, der nicht Teil einer praktischen Unterrichtssituation ist und daher auch wenig über die eigentlichen Anforderungen dieser (genauer gesagt *jeder* mir fremden) Situation sagen kann.

Da Autorensysteme unabhängig von konkreten didaktischen Planungen als Pauschallösungen entwickelt werden, entsprechen ihre Qualitäten als Werkzeuge nur in den seltensten Fällen genau den Anforderungen konkreter Unterrichtsplanungen. Wenn Autorensysteme angewendet werden, muß daher eine Harmonisierung zwischen dem didaktischen Entwurf und dem Werkzeug vorgenommen werden. Üblicherweise werden Werkzeuge — aus naheliegenden Gründen — den jeweiligen Problemen untergeordnet und diesen angepaßt. Da die Abgeschlossenheit von Autorensystemen situationsbedingte Anpassungen jedoch nicht gestatten, werden hier in aller Regel die didaktischen Planungen an die Werkzeuge angepaßt. Dies widerspricht dem Sinn und der Aufgabe individuellen didaktischen Planens. Um didaktische Entwürfe für computervermittelten Unterricht so wie entworfen umzusetzen, bleibt folglich nur eine Strategie: Es muß auf die Verwendung von Autorensystemen verzichtet und die Unterrichts–Software ohne Anwendung eines unflexiblen Werkzeugs durch traditionelle textuelle Programmierung hergestellt werden.

Dies war der Fall, als die oben genannte *Virtual Design Company* für DS4 im Herbst 1999 als Reaktion auf die praktischen Mängel von WebCT entwickelt

wurde. Der Leiter dieses Kurses, Assistant Professor Philip Wong, hat seine Planungen, seine Anforderungen und seine Gründe, schließlich eine eigene Lehrumgebung zu entwickeln, in dem folgenden Beitrag zusammengefaßt. Ich zitiere diesen kurzen Praxisbericht hier als realistische Fallstudie anstatt ein hypothetisches Scheitern eines Autorensystems zu konstruieren[75].

Hintergrund

Design Study 4 ist ein Kurs, der von der School of Design an der Hong Kong Polytechnic University angeboten wird. Der Titel dieses Kurses lautet *Information, Interaction and Global Context*. Mit dem Ziel, eine Lernumgebung bereitzustellen, in der Studierende dieses Kurses die Möglichkeit erhalten, online Erfahrungen in den Bereichen Kommunikation, Verhandlung und Kollaboration zu sammeln, stellte sich mir die Aufgabe, verfügbare Optionen zu analysieren und eine Wahl zu treffen. Im Folgenden versuche ich, die Ergebnisse dieser Untersuchungen und den Prozeß der Entwicklung einer eigenen Lehrumgebung zusammenzufassen. Darüber hinaus darstelle ich dar, warum ich eine individuell entwickelte Plattform einem verfügbaren kommerziellen Produkt vorziehe.

Das System soll die folgenden Funktionen bereitstellen:

1. Ein E-Mail–System, das Gruppen und Mailing–Listen unterstützt. Da die Studenten in Gruppen arbeiten, und da wir sie zu kontinuierlichen Verhandlungen anhalten, ist zu erwarten, daß Gruppen–Mitgliedschaften während des Kursverlaufs wechseln. Da ich nicht vorhabe, meine gesamte Zeit am Rechner damit zu verbringen, diese Veränderungen zu beobachten und notwendige Konfigurationen durchzuführen, wird ein System benötigt, das Studierenden die Möglichkeit zur selbständigen Gruppenverwaltung bietet. Dieses soll während des Kurses als wichtigster Informationskanal dienen.

2. In einem bestimmten Maß hängt der Erfolg eines Designprozesses von spontaner Interaktion zwischen Designern und ihrem Ideenfluß ab. Als Reaktion auf die diesbezüglichen Nachteile von E-Mail, rücken Video–Konferenzen in den Mittelpunkt des Interesses. Unglücklicherweise ist die erforderliche Infrastruktur in dem benötigten Umfang (über 100 Studenten) sehr teuer und nicht praktikabel. Daher habe ich mich darauf beschränkt, Chatrooms in Kombination mit Webseiten als Medium für

[75] Es handelt sich im hier um eine Übersetzung, die ich angefertigt habe. Der Originaltext, den mir Philip Wong als E-Mail gesendet hat, findet sich im Anhang.

Kommunikation in beinahe–Echtzeit einzusetzen. Ich ziehe textuelle Kommunikation visueller auch vor, weil sie die automatische Protokollierung von Dialogen deutlich erleichtert. Derartige Protokolle werden zum Zweck der Leistungsbewertung evaluiert (siehe unten). Ferner zwingt textuelle Kommunikation Studenten zu besserer Organisation und Strukturierung ihrer Beiträge, was zu höherer Effizienz führt.

3. Das System muß eine zentrale Möglichkeit zur Datenspeicherung bieten, in der Information, die aus der Kommunikation Studierender resultiert, gespeichert wird. Dies dient zwei Zwecken: Zum einen ist dieser Speicher ein Archiv für die studentischen Arbeiten. Dieses Archiv soll WWW–basiert sein (mit Unterstützung für den Up– und Download von Dateien) und mit einem graphischen Nutzer–Interface das Auffinden von Informationen erleichtern. Der zweite Zweck ist die Bereitstellung eines Schnittpunktes für Ideen, der es jedem Kursteilnehmer gestattet, sich über die Arbeit anderer zu informieren. Dadurch werden Kollaborationen angeregt und redundantes Arbeiten vermieden.

4. Zentrale Plattform für Präsentationen ist ebenfalls das Web. Da alle unsere Studenten mit Multimedia–Dateien arbeiten, muß der Webserver die neuesten Medientypen ohne Begrenzung der Dateigrößen unterstützen. Studenten haben die Möglichkeit, serverseitige Skripts und Datenbanken zu installieren, um die Funktionalität ihrer Webseiten zu erhöhen, wenn sie dies wünschen. Interface–Design ist ein wichtiger Aspekt der studentischen Präsentationen. Daher ist das ausschließliche Arbeiten mit Format–Vorlagen, wie es von kommerziellen Paketen angeboten wird, nicht akzeptabel.

5. Weiterhin wird das System dafür genutzt, die Kommunikation zwischen Studierenden zu protokollieren und die gesammelten Informationen darzustellen. Wenn die Teilnehmer z.B. zu einem Zeitpunkt des Kurses aufgefordert werden, Gruppen zu bilden, sollen die Lehrenden in der Lage sein, quantitative Aspekte individueller Kommunikations–Aktivitäten, wie etwa die Frequenz der Kommunikation, das verwendete Medium und die Anzahl der Kommunikationspartner einsehen zu können. Diese quantitativen Informationen können genutzt werden, um die Qualität der Kommunikation abzuleiten.

6. Das System wird Lehrenden auch als Bewertungs–Werkzeug mit einer Schnittstelle zu einer Noten–Datenbank dienen. Bei kontinuierlicher Leistungsbewertung werden Studierende Lesezugriff auf ihre neuesten Noten bzw. vergleichbare Bewertungen erhalten. Um Mißverständnisse zu vermeiden, erhalten Studierende die Möglichkeit, Beurteilungen, die sie nicht akzeptieren, zu kommentieren. Dies unterstützt Studierende bei der Analyse des eigenen Fortschritts und bei der Verbesserung der eigenen Leistung.

Evaluation

An der Hong Kong Polytechnic University steht WebCT als Plattform für die Implementierung von Online–Unterricht zur Verfügung. WebCT wird hier durch ITS (Information Technology Services), einer unabhängigen Abteilung, die mit der Organisation von Netzwerk– und Hardware–Infrastruktur betraut ist, verwaltet. Ein Vorteil dieses Arrangements ist die Tatsache, daß das System professionell betreut und damit für seine Zuverlässigkeit Sorge getragen wird. Jedoch tendieren *IT–Professionals* in teilweise übertriebenem Maß zur Einrichtung und Pflege kontraproduktiver Kontrollstrukturen. Obgleich ich durchaus einsehe, daß für den Schutz von Lernumgebungen legitime Gründe bestehen, sehe ich in unserem Fall den Punkt erreicht, an dem Lehr– und Lernbedürfnisse eingeschränkt werden. Daher ist die Entwicklung eines alternativen Systems unvermeidlich.

Als kommerzielles Paket bietet WebCT viele der oben genannten Funktionen und Verhaltensweisen. Es umfaßt ein umfangreiches Konzept zur Verwaltung von Nutzerkonten und Zugriffsrechten, bietet Mechanismen für Online–Lernkontrollen und –Benotung, Kommunikations–Werkzeuge wie Web–Hosting, Diskussionsforen und Gruppen–E-Mail sowie zur Archivierung und Recherche von Lehrmaterialien. Insofern ist es durchaus geeignet, die Durchführung von computervermitteltem Unterricht zu ermöglichen.

Für meine letztliche Entscheidung gegen die Verwendung von WebCT habe ich die folgenden Gründe:

1. WebCT bietet keine Chatrooms[76], was aber für die Durchführung unseres Kurses essentiell ist. Der Hauptgrund für dieses Manko von WebCT ist vermutlich die Tatsache, daß Chatrooms leicht zu Server–Überlastungen führen. Mit einem selbstgebauten System können wir jedoch sehr leicht selbst zusätzliche Server an das System anschließen, um bei Bedarf zusätzliche Rechenkapazitäten bereitzustellen. Nach Abschluß des Kurses können diese zusätzlichen Rechner als normale Arbeitsplatz–Rechner genutzt werden. Auch würden Systemabstürze in dieser dezentralen Architektur nur jeweils einen Kurs und nicht den ganzen Campus lahmlegen.

2. Trotz des dargestellten Funktionsumfangs bietet WebCT auf der operationalen Ebene keine ausreichende Flexibilität. Studenten haben keine Möglichkeiten zur Installation serverseitiger Skripts und Datenbanken und eine Plattenqouta rationiert den Speicherplatz, der jedem einzelnen

[76] Anmerkung: Die zum damaligen Zeitpunkt zur Verfügung stehende Version von WebCT umfaßte keinen Chatroom. Da der gewünschte Chatroom selbst nur als Ersatz für die eigentlich notwendige Videokonferenz–Unterstützung dienen sollte, die bis heute nicht Teil von WebCT ist, illustriert dieses Beispiel dennoch das Problem nicht verfügbarer Lösungen in abgeschlossenen unterrichtlichen Werkzeugen. Anstelle eines Chatrooms können didaktische Planungen an *jeder erdenklichen* Anforderung scheitern, die von Autorensystemen nicht unterstützt wird. Hiervon gibt es unendlich viele.

Studierenden zur Verfügung steht. Dies kann als Mangel im Design von WebCT, aber auch als Unzulänglichkeit auf Management–Ebene interpretiert werden. Der gesamte Aufbau (Software, Hardware und Management) reflektiert letztlich ein unzureichendes Verständnis der Bedürfnisse eines „konstruktiven“ Lehrsystems seitens seiner Entwickler.

3. WebCT wurde mit dem guten Vorsatz entwickelt, auch von Lehrern ohne Kenntnisse im Web–Publizieren angewendet werden zu können. Um die Schwierigkeiten der Courseware–Herstellung zu minimieren, werden Formatvorlagen bereitgestellt. Dadurch finden Lernende auch über die Grenzen einzelner Kurse konsistent gestaltetes Material vor. Jedoch bietet WebCT keine Alternative zu Formatvorlagen und macht es somit unmöglich, die Nutzerschnittstelle von Courseware individuellen Bedürfnissen anzupassen, was an einem Design–Fachbereich inakzeptabel ist. Rückblickend stellen wir fest, daß die Freiheit zur Herstellung eines eigenen Interfaces in unserer selbstentwickelten Lern–Applikation dazu beigetragen hat, daß unsere Studenten ihr gegenüber ein Gefühl von Eigentum und Zugehörigkeit entwickelt haben, was sie zu starker Aktivität während des Kurses angeregt hat.

4. Der Fortschritt der Open–Source Bewegung hat es uns ermöglicht, ein qualitativ hochwertiges System praktisch kostenfrei bereitzustellen. Es gibt von unserer Seite aus keinen Zweifel daran, daß frei verfügbare Software in Zukunft eine zentrale Rolle im Bereich der Software–Entwicklung spielen wird. Eine rechtzeitige Ausrichtung auf die sich hier etablierenden Standards wird uns in Zukunft helfen, Anstrengungen und Ressourcen zu sparen.

Folgerung

DS4 steht nach seiner ersten Durchführung vor einem Jahr nun vor seinem Abschluß im zweiten Jahr. Viele der ursprünglichen Anforderungen wurden erfüllt, während andere noch auf ihre Implementierung warten. Durch die Bereitstellung eines eigenen Online–Lehrsystems haben wir sehr viel darüber gelernt, wie ein solches System erstellt und konfiguriert werden sollte. Es wurden viele Ressourcen in das System investiert und neue Ideen durch studentisches Engagement angeregt. Wir mußten uns andererseits jedoch auch Problemen stellen, die bei einer Entscheidung für ein fertiges, kommerzielles Paket nicht aufgetreten wären. In Zukunft werden wir auf der Basis unseres experimentellen Ansatzes jedoch möglicherweise völlig neue Lernformen entwickeln. Die Leitgedanken bei dieser Weiterentwicklung unseres Online–Lehrsystems werden Anpaßbarkeit, Integration und Dezentralisierung sein.

2.11. Zusammenfassung

Am Ende der Problemanalyse angekommen, wird es nun höchste Zeit für eine etymologische Betrachtung der Problemstellung. Sowohl der Begriff *Autor* als auch der Begriff *Design* sind lateinischen Ursprungs. *Autor* bedeutet *Urheber* oder *Schöpfer* und ist semantisch verwandt mit *Autorität*, der *zwingenden Macht des Überlegenen* bzw. der *maßgeblichen, entscheidenden Persönlichkeit*. *Design* wurde in der zweiten Hälfte des 20. Jahrhunderts aus dem Englischen ins Deutsche entlehnt und bedeutet zunächst soviel wie *Muster oder Modell*. Zuvor ist der Begriff über das Italienische und Französische aus dem Lateinischen entlehnt worden. Auf Lateinisch bedeutet *designo* soviel wie *bezeichnen*, *abgrenzen*, *bestimmen* oder *anweisen*, und *designator* bedeutet *Kampfrichter* oder *Anordner*. Im Italienischen heißt *disegnare* soviel wie *zeichnen* aber auch *entwerfen*, *konzipieren*, *beabsichtigen*, *planen*, *beschließen*, *bestimmen*, *vorbestimmen* oder *ernennen*.

Der Begriff transportiert in seiner semantischen Herkunft die zentrale Wichtigkeit der Rolle des Autoren bzw. des Lehrers, der das *Was* und *Wie* im Unterricht vorbestimmt und verantwortet. Ich habe dargestellt, wie Hypermedia diese *Autorität der Autoren* durch die Implementierung Kontextfremder didaktischer Entscheidungen nicht nur potentiell untergräbt und wie Autorensystem–Entwickler in ihren Produkten dieser Autorität oftmals nicht gerecht werden.

Weiterhin habe ich ausgeführt, daß das Design eines Autorensystems das Design eines Designwerkzeugs darstellt und somit zwei entwickelnde Parteien in die Herstellung der letztlichen Unterrichtsmittel involviert sind: Lehrmittel*autoren* (Lehrer) und Autorensystem*designer*. Eine etymologische Betrachtung des Wortes *Design* ist insbesondere deshalb interessant, als dadurch auch der zweiten Partei eine entscheidende Autorität in diesem Zusammenspiel eingeräumt wird.

Dieser Konflikt ist keinesfalls nur sprachlich–historischer Natur sondern das Kernproblem der Anwendung von Autorensystemen im Unterricht. Zur Lösung dieses Konflikts fehlt eine generelle Theorie für das Design von Designwerkzeugen. Von Seiten der Informatik fehlen hierfür Methoden, mit denen wir die Universalmaschine Computer trotz ihres universellen Kerns und trotz der stra-

tegischen Offenheit, mit denen wir bösartigen Problemen begegnen müssen, mit problemorientierten nicht–programmiersprachlichen Nutzer–Schnittstellen versehen können. Diese Forderungen erscheinen zum gegenwärtigen Zeitpunkt utopisch. Die Betrachtung des Autorensystemdesigns von einem designtheoretischen Standpunkt aus hat eine Reihe von Widersprüchen und Wechselwirkungen aufgedeckt, die die Organisation unserer didaktischen Arbeit und den Einsatz von Informationstechnologien im Unterricht angehen.

Neben mangelhaft ausgeprägten Schnittstellen zu ihren Nutzern leiden Autorensysteme auch an mangelhaften Schnittstellen zueinander, zu ihren Daten, zu existerenden Datenbanken und zu anderen Softwareprodukten (der Import und Export von Multimedia–Dateiformaten sei von dieser Kritik teilweise ausgenommen, da sie von einigen Produkten sehr gut unterstützt werden). Ich habe die folgenden konzeptionellen Probleme in den Designs vorliegender Autorensysteme identifiziert:

- Autorensysteme sind weitaus weniger flexibel als Programmiersprachen.
- Autorensysteme bieten keine textuellen Schnittstellen als echte Alternativen zu visuellen Schnittstellen.
- Ihre konventionellen GUIs sind nicht geeignet, die unvorhersehbaren und komplexen Kommandos, die für die Bearbeitung des bösartigen Problembereichs Unterrichtsvorbereitung anfallen, zu repräsentieren.
- Sie sind typischerweise kommerzielle Produkte, die eher den Regeln des Marktes als den Idealen des Unterrichts folgen. Sie sind teuer und untereinander oft inkompatibel.
- Autorensysteme begleiten Lehrende nicht durch alle Phasen ihrer Arbeit. Feedback für Unterrichts–Evaluationen wird ebensowenig unterstützt wie systematische Feedbackschleifen zwischen Lehrpraktikern und Autorensystementwicklern.
- Sie erfordern einen zu hohen Lernaufwand als daß ihre Benutzung von Lehrpraktikern „nebenher“ erlernt werden könnte.

- In Produkten von Autorensystemen werden inhaltliche und methodische Elemente oft untrennbar miteinander verschmolzen. Systematische Wiederverwendung didaktischer Entwicklungsarbeit ist nur schwer möglich.
- Die Produkte von Autorensystemen können nicht dynamisch in Formate für spezielle Output–Medien wie z.B. Braille–Zeilen konvertiert werden.
- Sie sind nicht oder nur schlecht mit Projekt–, Lerner–, Noten–, Schul– und Stundenplan–Verwaltungssystemen integrierbar.
- Die Weltmodelle und somit auch das Interfacedesign von Autorensystemen sind praktisch ausnahmslos nicht an das eigentliche didaktische Anwendungsgebiet sondern an Diaprojektoren, Karteikartenstapel, Theaterbühnen usw. angelehnt.
- Ihre Produkte verbleiben technologisch und lernpsychologisch allzuoft auf behaviouristischem Niveau. Dies kann auch auf mangelhafte gestalterische und infrastrukturelle Freiheit zurückgeführt werden.
- Autorensysteme unterstützen nicht Kollaborationen von Lehrenden.
- Autorensysteme unterstützen nicht Kooperationen von Institutionen.
- Die Qualität von Autorensystemen ist praktisch kaum wissenschaftlich evaluierbar. Das Urteil über ihre Qualität sollten ihre lehrenden Anwender treffen.
- Das Design von Autorensystemen erfolgt nicht im Kontext ihrer Anwendung. Ihre Funktionalität und Schnittstellengestaltung sind in vielen Fällen sachfremd.
- Autorensysteme und ihre Produkte verhalten sich unflexibel und letztlich feindlich gegenüber Unvorhergesehenem im Unterricht. Diese Eigenschaft führt dazu, daß Unvorhergesehenes als störend empfunden und möglichst vermieden wird. Sie können außerhalb geschlossener Curricula inhaltlich keine zuverlässigen Lösungen gewährleisten.
- Autorensysteme gestatten nur in sehr seltenen Fällen eine echte Erweiterbarkeit ihres Funktionsumfangs.

- Es fehlen didaktisch orientierte Hilfen zur Projektverwaltung von Unterrichtseinheiten, die z.B. Lehrer und Lerner über aktuelle Ereignisse automatisch in Kenntnis setzen.
- Autorensysteme verfügen über sehr beschränktes Wissen über die Welt, in der sie agieren. Das menschliche Verständnis der Realität wird benötigt, um bei der Integration mehrerer Systeme semantische Differenzen zwischen Wissens–Domänen zu kompensieren und fortan zu warten.
- Stand–alone Laufzeitumgebungen mit beschränkter bzw. ohne Netzwerkunterstützung verstärken das Problem interoperabler Datenformate und isolieren die Lerner–Interaktion am Einzelarbeitsplatz.

Das Kernproblem und grundsätzliche Defizit existierender Autorensysteme ist die bei ihrer Anwendung zwangsläufig erfolgende unadaptierte Übertragung im Designprozeß definierter didaktischer Entscheidungen und Modelle auf reale, immer andere Unterrichtssituationen. Ursache für dieses Defizit ist das Autorensystemen implizit zugrundeliegende zentralisierte didaktische Planungsmodell, das die Verantwortung und Kompetenz didaktischen Entwerfens vom Lehrer auf die Ebene der technischen Implementierung verschiebt.

3. Gibt es einen besseren Weg?

> *The problem of unrealistic software design is greatly diminished when the designer is the implementor. The implementor's ease in programming and pride in the result is increased when he, in an essential sense, is the designer. Features are less likely to turn out to be of low utility if users are their designers and they are less likely to be difficult to use if their designers are their users.*[1]

Unterricht ist ein zu komplexer Gegenstand, als daß er auf der Basis trivialisierender, abgeschlossener Modelle hinreichend planbar wäre. Folglich blieben diverse Versuche, derartige Modelle für den (traditionellen oder technologievermittelten) Unterrichtsentwurf zu entwickeln, im Lehralltag praktisch ohne Konsequenzen. Es erscheint mir aber dennoch angebracht, sogenannte *Übersimplifizierungen*, *Pseudo– und Partial–Theorien des Lernens*[2] näher zu untersuchen und zu analysieren, wo die Ursachen für ihre Mängel liegen, um eventuell realistischere methodische Modelle zu entwckeln. Wenn Übersimplifizierungen vermieden werden, mangelhafte Teillösungen identifiziert und ggf. korrigiert sind und die hohe Komplexität und Individualität von Unterricht adäquat in die Theoriebildung einbezogen wird, erweisen sich diese Ansätze und Modelle möglicherweise als hilfreich.

Als Beispiel möchte ich das kybernetische Unterrichtsmodell diskutieren, weil es eine historische und inhaltliche Nähe zur Anwendung von Computern im Unterricht aufweist. Die kybernetische These, daß durch das Beschreiten eines

[1] Eastlake, Donald E.: [10]

[2] siehe Schulmeister, Rolf: [55], S. 80 ff.

direkten, prospektiv beschreibbaren Weges ein bekannter Ist–Zustand in einen definierten Soll–Zustand transformiert werden könne, wird von praktischem, menschenvermitteltem Unterricht allzuoft als Übersimplifizierung entlarvt. Wir ziehen im Unterricht ständig Materialien, Medien, mentale Modelle und inhaltliche Exkurse heran, die zunächst nicht Bestandteileil des Unterrichtsentwurfs waren, um spontan angepaßte Lehrziele zu erreichen, die mit geplanten, statischen Strategien nicht erreichbar sind. In diesen Fällen ist der Weg zum Grobziel des Unterrichts nicht direkt — er weicht von der ursprünglichen Planung ab und nimmt Umwege und manchmal auch Abkürzungen. Wie sollte eine kybernetische Maschine, deren didaktisches Weltmodell nicht die Qualität eines Kühlschrank–Thermostats übersteigt, solche Umwege und Abkürzungen entwerfen und beschreiten können? Wie soll eine solche Maschine weiterhin in die Lage versetzt werden, beurteilen zu können, ob es sich bei einer gegenwärtigen Position oder einem tatsächlich erreichten Ziel um ein Optimum handelt, ob es ggf. nur ein lokales Optimum ist und wie der Aufwand zum Erreichen eines Globalen Optimums bezüglich Kosten und Nutzen zu beurteilen ist? Ein kybernetisch gesteuerter Roboter, der in einem Haus ein Ziel aufsuchen soll, wird vielleicht den Raum, in dem sich das Ziel befindet gar nicht aufsuchen, weil er sich in einem anderen Raum, nur durch die Wand vom Ziel getrennt, in einem lokalen Optimum befindet und jede Bewegung im näheren Umkreis negativ bewertet wird. Unsere Lernsoftware scheint Probleme dieser Art noch völlig zu ignorieren. Die Robotik nimmt sich dieser Fragen an und hat Fortschritte aufzuweisen, die interessante Anhaltspunkte für das Design von computervermitteltem Unterricht bieten.

3.1. Ein Exkurs in die Robotik

[..] it might turn out to be too risky to assign a robot to undertake some important, long–range task, without some 'insight' about its own abilities. [..] If we want it versatile enough to solve new kinds of problems, it may need to be able to understand how it already solves easier problems.[3]

Mit diesem Gedanken illustriert Marvin Minsky im Jahr 1982 seine Idee der Zukunft Künstlicher Intelligenz. Dieser Vision zufolge sollen Computer (wie etwa computergesteuerte Roboter) dadurch intelligenter werden, daß sie ihre Fähigkeiten begreifen, also ein *Selbstbewußtsein* besitzen, durch Abstraktion Strategien entwickeln und Aussagen über deren Qualität machen können. Nachdem es uns Menschen bislang nicht gelungen ist, dem Computer *Intelligenz* zu vermitteln, soll der Computer diese nun also selbst entwickeln. Die Analogie zur menschlichen Entwicklung ist unverkennbar, und letztlich wurde diese Vision, die Minsky nach etwa 25 Jahren Forschung auf dem Gebiet der Künstlichen Intelligenz formuliert, ganz ähnlich bereits 1950 von Alan Turing präsentiert[4] (das sei hier nur am Rande bemerkt). Ich vermute, daß Rittels und Webbers Theorie von den zahmen und bösartigen Problemen die Erforschung Künstlicher Intelligenz (ob und mit welchem Resultat sie dort rezipiert wurde, ist mir nicht bekannt) möglicherweise beträchtlich hätte bereichern können. Das menschliche Attribut *Selbstbewußtsein* ist ebensowenig operationalisiert wie die Eigenschaft *Nutzerfreundlichkeit* von Autorensystemen. *Was* macht einen Menschen selbstbewußt? Solange diese Frage nicht beantwortet ist (und hierfür stehen sämtliche Anzeichen wirklich denkbar schlecht!), bleibt die Entwicklung selbstbewußter Computer ein bösartiges Problem und es gibt keinen definierten programmierbaren Algorithmus, der es lösen könnte. Kommen wir also auf das Beispiel der autonom navigierenden Roboter zurück, um zu untersuchen,

[3] Minsky, Marvin: [37]

[4] Turing, Alan: [57]

mit welchen Mitteln die Robotik tatsächlich Fortschritte erreicht hat. Wie zu erwarten, steht am Anfang erfolgreicher algorithmisch beschriebener Strategien eine Problemzähmung. Das formale Kriterium, das mobile Roboter am Ende der andauernden Forschungsarbeit auf diesem Gebiet erfüllen sollen, ist das Bestehen des sogenannten *MNT–Tests*, des Maximum Navigation Test:

> „The robot is placed in an environment that is unknown, large, complex and dynamic. After a time needed by the robot to explore the environment, the robot must be able to go to any selected place, trying to minimize a cost function (e.g. time, energy, etc).“[5]

Frühe Architekturen mobiler Roboter basieren auf dem sogenannten Sense-Model-Plan-Act–Ansatz, nach dem der Roboter seine Umwelt sensorisch erfaßt, daraufhin ein Modell seiner Umwelt anlegt, darin dann einen Weg definiert und schließlich versucht, diesem Weg zu folgen. Die Ergebnisse dieses Ansatzes waren jedoch enttäuschend, da die sensorische Messung der Umwelt nur Ausschnittsweise erfolgen kann und daraufhin angelegte Modelle der Umwelt ebenfalls unvollständig sind. Aus diesem Grund ist auch die erforderliche Selbstlokalisierung des Roboters oft mangelhaft. Weiterhin werden Umweltmodelle oft aus pragmatischen Gründen schematisiert und vereinfacht, indem z.B. dreidimensionale Sensordaten als zweidimensionale Polygone modelliert werden. Die angelegten Modelle sind statisch, und unerwartete dynamische Elemente wie etwa bewegliche Hindernisse sind ihnen fremd. Innerhalb mangelhafter Umweltmodelle kann ein Roboter letztlich nur mangelhafte Wegbeschreibungen definieren, und ohne die Fähigkeit, auf unerwartete Hindernisse reagieren zu können, bewegen sich Roboter dieser Bauart sehr langsam und müssen bei jeder unerwarteten Kollision die Sense-Model-Plan-Act–Prozedur von neuem beginnen.

Als Reaktion auf diese Mängel werden Umwelt–Modelle nach der Devise „die Welt ist selbst ihr bestes Modell“ in späteren Ansätzen abgeschafft und Sensor–Input zur kontinuierlichen Wegdefinition in Echtzeit verarbeitet. Braitenberg beschreibt anschaulich, wie unterschiedliche logische Strukturen in den verwen-

[5] Die MNT–Definition sowie die Grundzüge dieses Exkurses stammen aus: Salichs, M. A. und Monreo, L.: [53]

deten Kontrollsystemen zu unterschiedlichen Verhaltensweisen von derartigen Robotern führen[6]. Nach dem Paradigmenwechsel von planbasierter zu sensorbasierter Navigation bewegen sich mobile Roboter schneller und sind in der Lage, auf unerwartete bewegliche Hindernisse sinnvoll zu reagieren. Ideal ist dieser Ansatz jedoch auch nicht. Nutzer mobiler Roboter wollen Aufgaben erfüllen, was im Fall des MNT dem Erreichen definierter Zielorte entspricht. Eine globale Zielvorgabe kann jedoch nicht einfach in einen lediglich auf der Basis von Sensordaten konstruierten Weltausschnitt integriert werden. Der Roboter kennt seine Umwelt jenseits dessen, was seine Sensoren erfassen, nicht. Das Resultat sind große Probleme bei dem Versuch, in unbekannten Teilen der Umwelt sowohl Zielorte als auch einen Weg dorthin zu modellieren und dabei gleichzeitig die erforderlichen Kosten zu minimieren.

Sowohl modellbasierte als auch sensorbasierte Navigationsmodelle weisen charakteristische Vor– und Nachteile auf mit dem Resultat, daß sich letztlich beide Bauweisen im MNT als unzuverlässig erweisen. Der erforderliche Kompromiß, den aktuelle Forschungsansätze favorisieren, besteht in der Verwendung eines übergeordneten Weltmodells, das in der Lage ist, sowohl in Echtzeit gewonnene Sensor–Daten und daraus dynamisch errechnete Wegbeschreibungen als auch Zielort–Definition von Nutzern zu integrieren. Innerhalb eines solchen Modells kann der Roboter sich sowohl selbst relativ zu seiner Umwelt lokaisieren und in ihr navigieren als auch auf unvorhergesehene Hindernisse oder andere Veränderungen seiner direkten Umwelt reagieren.

Mobile Roboter sind in einigen essentiellen Aspekten mit Autorensystemen vergleichbar. Beide sollen als Werkzeuge in Arbeitsprozessen ständig wechselnde Aufgaben in sich ständig verändernden Umwelten gemäß gegebenenfalls dynamisch definierten Nahzielen erfüllen und dabei so kosteneffektiv wie möglich die ihnen zu Verfügung stehenden Ressourcen nutzen. Maßgeblich für die Definition der gewünschten Verhaltensweisen von Autorensystemen und mobilen Robotern sind ihre Anwender vor Ort, also die Lehrer bzw. die Besitzer von Robotern. Bei ihrer Entwicklungsarbeit haben die Designer von mobilen Robotern und Autorensystemen nur eine sehr grobe Vorstellung von dem Anwendungskontext ihres

[6] Braitenberg, Valentino.: [5]

Produkts.

Autorensystem–Hersteller entwickeln Werkzeuge, die sich einerseits für unterschiedliche Märkte wie die Produktion von Multimedia–CD-ROMs oder Overhead–Projektionen, andererseits für sehr breite Sprektren didaktischer Anwendungen eignen sollen.

Die dafür erforderliche Anwendbarkeit erreichen sie ohne echte Abstraktion durch die Implementierung kleinester gemeinsamer Nenner in Form typischer didaktischer Standard–Anwendungen. Hinreichende Flexibilität, die die (ggf. sogar dynamische, automatische) Anpassung an die Besonderheiten konkreter Kontexte gestatten würden, ist unter den kommerziellen Produkten, die auf möglichst breite Vermarktung ausgerichtet sind, nicht zu finden.

Im Vergleich dazu verfolgt die Entwicklung mobiler Roboter wesentlich intelligentere, abstraktere Strategien. Diese ermöglichen die Bereitstellung von Robotern, die sich zunächst allgemein für bestimmte Problemklassen eignen (etwa für die Reinigung von Kanalsystemen oder Botendienste in Krankenhäusern), darüberhinaus aber in der Lage sind, sich flexibel an die Charakteristika eines konkreten Problems anzupassen.

Mobile Roboter werden zu diesem Zweck mit Sensoren und der erforderlichen Technik ausgestattet, die es ihnen ermöglichen, sich an konkrete Anwendungskontexte anzupassen. Diese Anpassungen erfolgen auf mehreren Ebenen, die als *Weltmodell* und *lokales Modell* bezeichnet werden. Das Weltmodell ist das relativ statische Umweltmodell des gesamten bekannten Anwendungskontextes, das der Anwender im Roboter und innerhalb dessen sich der Roboter bewegt. Das lokale Modell beschreibt die unmittelbare Umgebung eines Roboters während seines Einsatzes mit den für die Navigation relevanten dynamischen Details. Dieses lokale Modell konstruiert der Roboter auf der Basis laufend anfallender Daten seines sensorischen Apparats. Innerhalb dieser beiden, auf unterschiedlichen Größenskalen angesiedelten Umweltmodelle kann ein mobiler Roboter (lokale) Nahziele und (globale) Fernziele verfolgen, wobei er sinnvollerweise die Prioritäten für die Nahziel–Definition von der Fernziel–Definition ableitet.

Dieser Vergleich von Autorensystem–Einsatz und Robotik soll zeigen, daß der

unterrichtliche Einsatz von Autorensystemen auf einer Reihe problematischer Annahmen basiert und daß die Robotik der didaktischen Software–Entwicklung ganz offensichtlich einige Erkenntnis– und Entwicklungsschritte voraus ist. Während die Definition von Nah– und Fernzielen ein selbstverständliches Standardverfahren in vielen planerischen Kontexten und ein zentraler Bestandteil des Entwurfs von Unterricht ist, gibt es von Seiten der gegenwärtigen Autorensystementwicklung keine fundierten Ansätze, diese Strategie für den computervermittelten Unterricht aufzugreifen und zu unterstützen.

Im Laufe der oben dargetellten Entwicklungsgeschichte mobiler Roboter hat es praktisch keinen ernsthaften Ansatz gegeben, ein Design für einen allgemeinen Anwender–Markt zu entwickeln, das aber mit einem konkreten festverdrahteten Weltmodell ausgestattet ist (was zur Folge hätte, daß sich der Roboter nur an einem einzigen, ausgesuchten Ort orientieren könnte). Ein solches irrationales Verfahren ist jedoch gängige Praxis beim gegenwärtigen Autorensystem–Design.

Während mobile Roboter sich Modelle ihres Einsatzgebietes erstellen und ihr Verhalten diesen anpassen können, bleibt Anwendern von Autorensystemen bestenfalls die Auswahl eines von wenigen vorproduzierten didaktischen Modellen. Sie haben vielleicht die Wahl zwischen der Erstellung einiger Typen von Multiple-Choice–Tests oder Diashows. In anderen *Gegenden* als diesen Schablonen kennen sich Autorensysteme nicht aus. Man könnte an dieser Kritik mit einem gewissen technischen Einfühlungsvermögen beanstanden, daß ein mobiler Roboter im Gegensatz zu einem Autorensystem über einen Wahrnehmungsapparat verfügt und daß es ungleich einfacher (zahmer) ist, unvorhergesehene Hindernisse geometrisch zu erfassen, zu abstrahieren und zu modellieren als die (bösartige) dynamische, maschinelle Herstellung eines Unterrichtsmodells. Dieser Einwand ist zunächst richtig, aber er ist sehr kurzsichtig:

Vertraut man der didaktischen Kompetenz des Autorensystem–Anwenders (und dies sollte man angesichts seiner typischerweise absolvierten akademischen oder gleichwertigen Ausbildung wohl tun[7]), dann muß man erkennen, daß Auto-

[7] Die Annahme, daß akademische Kompetenz mit praktischer didaktischer Kompetenz einhergeht oder sogar gleichzusetzen ist, trifft vielleicht nicht in allen Fällen zu. Da unser Modell

rensysteme sowohl über Sensoren für das Unterrichtsgeschehen als auch über Möglichkeiten zur didaktisch fundierten, kontextorientierten und ethisch reflektierten Entwicklung von Unterrichtsmodellen verfügen, nämlich über *den Lehrer*. Die Augen und die Hände des Lehrers sind der sensorische Apparat, der den Unterricht beobachten und analysieren kann, und sein Verstand enthält ein umfassendes Weltmodell und gesunden Menschenverstand, mit dem er den (computervermittelten) Unterricht lenken kann.

Die Unfähigkeit von Software, die zur Bearbeitung bösartiger Probleme erforderlichen profunden Weltmodelle zu fassen, zu verwalten und anzuwenden ist Grund dafür, daß in computergestützten entwerferischen Aufgabengebieten der Entwerfer nach wie vor eine maßgebliche Rolle spielt. Während zahme Aufgaben von Bankkassierern heute selbstverständlich von Automaten bearbeitet werden können, bedarf das computerbasierte Design etwa von Architektur oder Unterricht wohl noch lange eines menschlichen Entwerfers. Das dabei typische Tätigkeitsbild ist der *Dialog* zwischen Entwerfer und Computer. In diesen Bereichen sollten wir die agierenden Algorithmen nicht im traditionellen Sinn als die in Programmcode beschriebenen Verhaltensweisen des Computers begreifen. Vielmehr sollten wir Algorithmen hier als Strukturen verstehen, die sich sowohl in digitalem Programmcode (in *Software*) als auch in den kognitiven Strukturen ihres Anwenders (in *Wetware*) manifestieren (Ich habe diesen erweiterten Algorithmus–Begriff bereits in einem anderen Kontext als Reaktion auf den unbefriedigenden Fortschritt der Erforschung und Entwicklung Künstlicher Intelligenz in Bereichen bösartiger Problemklassen vorgeschlagen: Fischer, Thomas: [16]). Vor dem Hintergrund der Erfordernis derartiger „halbnasser“ Algorithmen ist die Unterstützung universeller (oder zumindest abstrakter, adaptiver) Semantik in der Nutzerschnittstelle von essentieller Bedeutung.

So wie Nolan[8] dazu anregt, direkt im Entwurfskontext kritisch–rationalistisch

der Lehrerausbildung jedoch weitgehend auf ihr basiert, gibt es keine Alternative zu diesem Vertrauen.

[8] Nolan, Andrew: [40], S. 97 ff.

nicht haltbare positive Aussagen und Prognosen zu verwenden, die ihre Relevanz nicht aus der Ableitung von allgemeingültigen Sätzen, sondern schlicht aus ihrer unmittelbaren Nähe zur Anwendung beziehen, empfehlen auch Petersen und Petersen[9] ein Beobachtungs–Verfahren, das formal zwar als naiv–empiristisch bezeichnet werden muß, für das aber *kontextrelevant* meines Erachtens vielleicht eine bessere Bezeichnung ist.

3.2. Vorschlag für ein Entwurfsmodell

Nun stellt sich im Hinblick auf die Entwicklung eines alternativen Designs die Frage „Gibt es einen besseren Weg?“ Gibt es eine Strategie, mit der Lehrende geeignetere Werkzeuge für die Herstellung ihres computervermittelten Unterrichts erhalten? Philip Greenspun antwortet:

> „Es gibt einen besseren Weg: Lernen Sie programmieren (oder beschäftigen Sie jemanden, der es kann).“[10]

Dieser Ratschlag klingt nicht besonders wissenschaftlich und er ist es auch nicht. Er ist pragmatisch, was angesichts der formalen wissenschaftlichen Probleme der Evaluation und Erkenntnisgewinnung im Bereich Lernsoftware–Entwicklung wie ich meine eine naheliegende und durchaus brauchbare Alternative darstellt. Schulmeister zitiert eine Stellungnahme von Gerald Nelson, mit der dieser pragmatisch die Erfahrungen beschreibt, die er bei der Entwicklung einer eigenen Lernumgebung gesammelt hat:

> „In a fit of naivete, I decided to develop my own CAI package to teach principles of economics to a large number of students working on their own in many different computer laboratories [..]. Some lessons I learned:

[9] Petersen P. und E.: Pädagogische Tatsachenforschung. Paderborn 1965, S. 129 ff. Nach: König, Eckard und Zedler, Peter: [28], S. 41 ff.

[10] Greenspun, Philip [22], S. 165

> get lucky and hire a student who combines knowledge of educational techniques with excellent HyperCard programming skills
>
> don't try to be fancy. Someone will find a way to break your stack
>
> test your software again and again and again; or learn how to deal with wiry students
>
> lead the horse to water and force him to drink': tell students exactly how to use the software and how it will improve their grades
>
> be prepared to sacrifice your career
>
> work at a university with good computer facilities and other faculty with interest in computer–aided instruction."[11]

Ich möchte hier einen Entwicklungsmodell entwerfen, das sowohl an die hier demonstrierte moderierte Weitergabe persönlicher praktischer Erfahrung als auch an das der Open-Source–Softwareentwicklung angelehnt ist und dabei sowohl für Nicht–Programmierer als auch für Nicht–Lehrpraktiker analysierbare und modifizierbare Musterlösungen liefert, die in einem kollaborativen evolutionären Prozeß hergestellt und gesammelt werden sollen. Ich verfolge dabei gleichzeitig drei Strategien zur Problemlösung:

- **Bereitstellung besserer Werkzeuge** Die dargestellten Mängel existierender Autorensysteme sollen vermieden und die Merkmale dieser Werkzeuge an ihren Problemraum angenähert werden.
- **Erweiterung der Kompetenz Lehrender** Ziel der Autorensystem–Anwendung ist Nutzerfreundlichkeit, die durch Mangel an Flexibilität erkauft wird. Daher erfordert ein Mehr an Flexibilität Abstriche bei der Nutzerfreundlichkeit. Diese können abgefangen werden, wenn Lehrende bessere Möglichkeiten erhalten, sich die für den Einsatz von Technologie erforderliche Qualifikation anzueignen.
- **Verbesserung der Kommunikation Lehrender Autorensystem–Entwickler** Um die ersten beiden Strategien zu fördern, kann verbesserte Kommunikation unter Lehrenden auf zwei Weisen helfen: Einerseits

[11] Nelson, Gerald In: Schulmeister, Rolf, [55], S. 391

können Unterrichtserfahrungen, Strategien und deren Erfolg sowie einzelne technische Lösungen ausgetauscht werden. Andererseits bietet eine Gemeinschaft mit gemeinsamen Zielen einen Kontext, der das Erlernen von erforderlichen Methoden und Werkzeugen erleichtert.

Wir können *bessere Werkzeuge* bereitstellen, wenn wir die Strategien, auf denen sie beruhen und die Metaphern, durch die wir sie verstehen, an den jeweiligen Problembereich annähern. Hier sehe ich ich zwei Möglichkeiten zur Verbesserung existierender Autorensystem–Technologien.

Zum einen sind die Paradigmen, durch die Autorensysteme versuchen, intuitiv verstanden zu werden, sehr weit von der Natur der Probleme, die sie lösen sollen, entfernt. Mit einem Programm, das versucht, sich wie ein Stapel Karteikarten zu verhalten, kann man sicherlich leicht Lernsoftware herstellen, die sich am Verhalten eines Stapels Karteikarten orientiert. Doch Unterricht ist weit mehr als ein Stapel Karteikarten und leider wissen wir nicht, welche Verhaltensweisen Lernsoftware aufweisen soll, bis ein Lehrer seine Unterrichtssituation analysiert und seinen Unterricht entworfen hat. Daher brauchen wir *universellere* Gestaltungsmetaphern. Mit der Theater–Metapher versuchen einige Autorensysteme universelles Verhalten zu unterstützen, so wie die Theaterbühne flexibel Modelle von Weltausschnitten unterstützt. Resultat dieser Universalität ist jedoch schwere Erlernbarkeit (das Gegenteil dessen, was Schnittstellen–Metaphern erreichen sollen) und das völlige Fehlen jeglicher didaktischer Problemorientierung, wie das Beispiel Macromedia Director zeigt. Ich schlage eine Schnittstellen–Metapher vor, die sich unverbindlich am Vorbild *Schule* orientiert. Auch die Schule ist ein flexibles Konzept (das real auch in der Lage ist, ein Theater zu integrieren), es liefert aber einen Rahmen und eine Terminologie, die eine optimale Nähe zum Problemraum und zu den Kenntnissen Lehrender aufweist. Eine offene System–Architektur soll dabei gewährleisten, daß die Schnittstellen–Metapher mit ihrem natürlichen Vorbild nicht als Korsett wirkt, außerhalb dessen Handlungen und Funktionsweisen nicht unterstützt werden. Denn obgleich traditioneller Unterricht naheliegendes Vorbild für den computer– und netzwerkvermittelten Unterricht ist, sind diese beiden Unterrichts–Modi keinesfalls identisch. Da wir noch nicht wissen, wohin sich di-

gitale Unterrichtspraktiken in Zukunft tatsächlich entwickeln, sollten sie durch die erforderlichen Werkzeuge nicht zu streng an vordefinierte Vorbilder gefesselt werden.

In bezug auf die Gestaltung der konkreten Schnittstelle zwischen Lehrer und Computer schlagee ich eine bedingte Rückkehr zu textueller Programmierung vor. Auf der Basis gegenwärtiger Software–Techologien läßt es sich im Kontext bösartiger Probleme nicht vemeiden, daß graphische Nutzer–Schnittstellen nichts anderes sind als Schnittstellen zu Bibliotheken vage vorausschauend vorformulierter Handlungen. Wenn die Effektivität von Unterricht dadurch gewährleistet werden soll, daß Lehrende fortlaufend Unterrichtssituationen analysieren, um individuell abgestimmte und innovative Strategien zu entwickeln (davon gehe ich aus), ist für erforderliche Werkzuge dieses grundlegende Merkmal graphischer Nutzerschnittstellen nicht akzeptabel. Visuelle Interaktion mit Software hat weniger mit eigenen Entscheidungen zu tun als mit der Auswahl aus wenigen Entscheidungen, die andere zuvor getroffen haben. Neal Stephenson diskutiert genau dieses Problem und beendet sein Buch *In the Beginning...was the Command Line* mit den folgenden Worten:

> „[..] if you don't like having choices made for you, you should start making your own“[12]

Sprachliche Grammatik hat im Gegensatz zur Abgeschlossenheit gegenwärtiger „visueller Grammatiken“ den Vorteil, daß sie es gestattet, mit endlichen Mitteln unendlich viel auszudrücken. So ist es sehr einfach möglich, das problemorientierte Vokabular einer Autorensprache durch das Hinzufügen neuer Objekte ständig zu erweitern, ohne die formalen Strukturen der Sprache zu verändern. Genau dies möchte ich mit der hier entworfenen Mustersprache für computervermittelten Unterricht erreichen. Zur Vermeidung typischer Probleme von Autorensystemen ergeben sich drei grundsätzliche Erfordernisse für das Sprachmodell:

Die Sprache darf nicht in einer „Sandbox“ eingebettet und durch sie eingeschränkt werden. Das Realitätsmodell der Sprache soll nicht wie im Fall

[12] Stephenson, Neal: [56], S. 151

der Skriptsprachen existierender Autorensysteme eine Bibliothek definierter Software–Elemente sein, sondern die didaktische Realität selbst. Wenn die didaktische Realität neue programmiersprachliche Elemente erfordert, sollen sie leicht in die existierende Sprache eingebettet werden können.

Die offenen Standards der Sprache sollten mit offenen Quelltexten einhergehen. Das im Fall von Compiler–Sprachen mögliche Abschließen von Software und die Einbehaltung der zugrundeliegenden Quelltexte ist eine selbstverständliche Praxis kommerzieller Software–Anbieter und daher ein Problem verfügbarer kommerzieller Autorensysteme. Diese können in ihrem Verhalten einerseits nur oberflächlich *nachvollzogen*, nicht aber *verstanden* werden. Die Programme sind nicht oder nur auf vordefinierte Weisen erweiterbar, hierfür ggf. verfügbare Schnittstellen binden Lehrende an die Verwendung bestimmter, machinennaher und somit wenig nutzerfreundlicher Technologien (z.B. C/C++). Neben Problemen bei der individuellen Erweiterung von Software erschweren nicht verfügbare Quelltexte das Erlernen der jeweiligen Technologien, das seinerseits eine zwingende Voraussetzung für individuelle Programm–Erweiterung ist. Wenn Programmstrukturen im Klartext einsehbar sind, gestatten sie Lehrenden problemorientiertes Erlernen von Programmierkenntnissen und fallbasiertes Modifizieren von Software.

Die Sprache braucht eine starke Trennung von Programmstrukturen und Daten. Je besser „didaktische Objekte“ (Lerninhalte, Lerner, Unterrichtsmethoden usw.) formal getrennt werden, desto leichter sind einerseits ihre Wiederverwendung und andererseits die hierfür erforderliche Modifikation. Eine Trennung von Daten und ihrem Verhalten und deren Beschreibung in lesbarem Klartext erlaubt Nachvollziehbarkeit, Interpretierbarkeit und Modifizierbarkeit und somit die Wiederverwendung von bewährtem Expertenwissen über die Grenzen von Wissens–Domänen und Institutionen hinaus. Datenformate, die diesen Kriterien genügen, gestatten die individuelle Herstellung graphischer Nutzeroberflächen für die Material–Herstellung durch Nicht–Experten. Daher schließt das hier vorgeschlagene Sprachmodell die Bereitstellung visueller Schnittstellen nicht grundsätzlich aus. Wichtig ist jedoch, daß in einem unterrichtlichen Kontext *erst* eine didaktische Analyse durchgeführt wird und auf dieser Basis

dann die didaktische Methodik erwogen wird, indem eine graphische Nutzerschnittstelle hergestellt wird, das tatsächlich die Handlungen und die Funktionalität unterstützt, die innerhalb der jeweiligen Unterrichtssituation angestebt werden. Mit denselben Mitteln, mit denen Unterrichtsmaterialien dynamisch aus abstrakt formulierten Inhalten, definiertem Verhalten und Kontextvariablen hergestellt werden können, lassen sich auch visuelle Autoren–Schnittstellen erzeugen. Das Resultat ist eine textuelle Nutzerschnittstelle mit optionalem, kontextspezifischem Autoren–GUI.

Eine Sprache, die diesen Forderungen entspricht, ist die Programmiersprache SeL mit dem Frontend–Generator SeSAMe.

Ich habe das Ziel, die Erweiterung der Kompetenz Lernender und die Verbesserung ihrer Kommunikation als Sekundär–Effekt des beschriebenen Werkzeug–Modells zu erreichen. Grund für die Annahme, daß dies erreichbar ist, ist der Erfolg des Open-Source–Entwicklungsmodells in der jüngeren Geschichte der Software–Entwicklung.

„Nutzerfreundliche“ Autorensysteme wie Toolbook, Metacard oder Frontpage erweisen sich leicht als derart schwerfällig, daß sie im Bildungsalltag kaum Anwendung finden und daher praktisch keinen Effekt haben. Vor diesem Hintergrund wird man kaum erwarten, daß die weitaus schwieriger realisierbare Programmierung „von Hand“ von Lehrpraktikern akzeptiert werden kann. Die Bösartigkeit des Problems Unterrichtsvorbereitung, der Unterricht selbst und die damit verknüpften Verpflichtungen können (insbesondere wenig erfahrene) Lehrende leicht zu zwanzig Stunden Arbeit am Tag zwingen. Ich habe bereits beschrieben, daß Lehrer diejenigen Entwerfer sind, die (etwa im Gegensatz zu Industriedesignern oder Architekten) ihre Produkte selbst herstellen müssen und damit einer enormen Doppelbelastung ausgesetzt sind. Und jetzt sollen sie auch noch ihre eigenen Werkzeuge herstellen? Ausgerechnet in einem sich so schnell wandelnden und lernintensiven Bereich wie der Informationstechnologie?

Natürlich soll nicht jeder Lehrer seinen gesamten Unterricht in einer textuellen Programmiersprache „von Hand“ programmieren. Dennoch werde ich auf den folgenden Seiten Lösungen vorschlagen, deren Umsetzung erfordern wird, daß Lehrende einen deutlich engeren Kontakt zu Computern pflegen und einige von ihnen auch Programme schreiben. Als Ausgangspunkt dafür biete ich im zweiten Teil dieser Arbeit eine Reihe von Programmen und Werkzeugen an, die in der praktischen Lehre eingesetzt werden können. Im Unterschied zu traditionellen Autorensystemen werden diese Programme jedoch im Quelltext lesbar und daher relativ verständliche, modifizierbare, interoperabele, inhaltlich und formal kommunizierbare und wiederverwendbare, aus der Praxis stammende, bewährte Lösungen beinhalten und kostenfrei verfügbar sein.

Diese Musterlösungen wurden im Rahmen mehrerer Online–Lehrprojekte entwickelt und größtenteils bereits erfolgreich eingesetzt. Diejenigen Lösungen, die sich in den genannten als weniger erfolgreich erwiesen haben, sind nicht in der Sammlung enthalten, die mit ihnen gemachten Erfahrungen sind aber indirekte Bestandteile der aufgeführten Lösungen.

Prinzipiell spricht nichts dagegen, in einer zukünftigen Sammlung von Mustern für das Autorensystemdesign auch *schlechte*, nicht *bewährte* und *besonders problematische* Muster als solche zu dokumentieren und zu kommunizieren um das Wiederholen von Fehlern zu vermeiden. Die wissenschaftliche Kultur, in deren Rahmen praktische Ansätze für computervermittelten Unterricht gegenwärtig diskutiert werden, ist praktisch ausschließlich auf die Präsentation und Diskussion erfolgreicher Konzepte fixiert[13]. Obwohl bei der Kommunikation wissenschaftlicher Journal–Papers und Konferenzbeiträge Erkenntnisse in verschiedenen Kontexten gewonnen und ggf. angewendet werden (Wissenschaftler einer Institution versuchen, publizierte Erkenntnisse von Wissenschaftlern anderer Institutionen anzuwenden) findet dies in einem fast ausschließlichen *Learning–From–Success*–Verständnis statt (vgl. Abschnitt 2.8.2). Um auf diesem Wege tatsächlich voneinander zu lernen und zu profitieren wäre die Kommunikation fehlgeschlagener Ansätze eigentlich ein sichereres Verfahren.

[13] Dies stelle ich insbesondere im Hinblick auf den internationalen Diskurs im Bereich der Design– und Architekturausbildung fest.

3.3. Das Autorensystem SeSAMe

Während diese Arbeit entstand, habe ich an einem einjährigen Redesign–Projekt mitgewirkt, in dessen Rahmen die WWW–Präsenz der Universität Gesamthochschule Kassel überarbeitet wurde. Ausgangspunkt zur Zeit des Projektbeginns Anfang 1998 war eine Liste mit Kritikpunkten und Problemen, die das damalige WWW–Angebot der Universität aufgeworfen hatte. Neben dem Bedarf nach einer visuellen Überarbeitung (das damalige grafische Design war bereits seit fünf Jahren online) bereitete es vielen Autoren Probleme, ihre publizierten Inhalte auf dem neuesten Stand zu halten und das Corporate Design der Universität dabei technisch umzusetzen. Ein besonderes Problem bereitete die für das Online–Publizieren erforderliche technische Qualifikation. Schwierigkeiten sowohl beim Erlernen und Benutzen von WYSIWYG–Editoren als auch beim Erlernen und Anwenden von textueller HTML–Programmierung haben dazu geführt, daß Dokumente nicht stabil in den gängigen Browsern dargestellt wurden, Navigationshilfen versagten und eine Reihe weiterer technischer Probleme auftraten.

Die Entwurfsphase des Projektes — so wie die Entwicklung und Verwaltung des GhK–WWW–Services selbst — war ein in hohem Maß kollaborativer, demokratischer Prozeß, in dem (der Natur des Mediums entsprechend) jeder, der konnte und wollte, zur Mitsprache aufgerufen war. An diesem Experiment beteiligten sich über einhundert freiwillige Sachverständige und Autoren, die zunächst etwa ebenso viele Anregungen, Hinweise, Anforderungen und Ideen in verschiedenen Gesprächsrunden, eMail–Befragungen und einer Online–Diskussion beitrugen.

Im WWW–Design gibt es gute Gründe sowohl für maximalistische als auch für minimalistische Strategien. Um öffentlich ein technisch aktuelles und visuell ansprechendes Bild abzugeben, empfiehlt sich der Einsatz von aufwendiger visueller Gestaltung mit vielen grafischen Elementen und neuesten Technologien (Flash, Java)[14]. Andererseits soll das Informationsangebot einer Universität in-

[14] Dieses Kriterium ist unserer informellen Erhebung zufolge den Autoren und Anbietern von Dokumenten wesentlich wichtiger als den Lesern, denen leichte Navigation und das tatsächliche Finden gesuchter Information wichtiger zu sein scheint.

ternational durch sehr heterogene Client–Technologien und Netzwerke rezipiert werden. Output–Medien für Minderheiten wie z.B. sogenannter „Braille–Zeilen“ für sehbehinderte Computer–Anwender sind darüber hinaus lediglich in der Lage, reinen ASCII–Text und keine Grafiken darzustellen.

Auf der Suche nach Kompromissen zwischen diesen beiden extremen Ansätzen stellte sich heraus, daß sowohl die Verwendung bereitstehender WYSIWYG–Editoren (in diesem Fall insbesondere MS Frontpage) als auch textuelle HTML–Programmierung für den überwiegenden Teil der WWW–Autorenschaft keine ausreichenden Mittel darstellen, um technisch und inhaltlich sicher Dokumente in einem vorgegebenen Format herzustellen. Hochschulangehörige klagten darüber, daß ein neues Design sie bei der Pflege ihrer Online–Dokumente technisch überfordern würde und daß es sich darüber hinaus kaum mit ihren täglichen Aufgaben vereinbaren ließe, jede eigene Datei manuell einzeln in das neue Design zu portieren. Es war vorherzusehen, daß das letztere Problem mit jedem weiteren Redesign–Projekt in der Zukunft erneut auftauchen würde. Als Lösung dieses Problems wurde das Autorensystem SeSAMe („SErverbased Style and Authoring ManagEment“) entwickelt[15]. Es basiert auf der Trennung von Daten und ihrem Verhalten und gestattet die Wiederverwendung von Experten–Wissen. Variable Dokument–Elemente werden über eine WWW–basierte graphische Nutzerschnittstelle von Autoren eingegeben und als Variablen–Wert–Paare in Dateien mit der Endung `.ghk`[16] im Dokument–Verzeichnis des WWW–Servers abgelegt.

Beim Aufruf einer solchen Datei durch einen Online–Leser wird diese in Echtzeit in ein beliebiges Text–Ausgabeformat, z.B. in HTML, konvertiert. Das dafür zuständige Programm ist ein Interpreter für die Programmiersprache SeL („The SeSAMe Language“)[17]. Aus syntaktischer Sicht ist jedes SeL–Dokument, al-

[15] Diese Beschreibung des Autorensystems SeSAMe stammt inhaltlich teilweise aus: Fischer, Thomas: [14]

[16] Diese Endung ist letztlich beliebig wählbar. „GhK“ ist im Fall der Universität Gesamthochschule Kassel die naheliegendste Dateinnamens–Erweiterung. Die mittlerweile etwa 10 weiteren Universitäten, die SeSAMe benutzen, nutzen jeweils eigene Abkürzungen.

[17] Maßgeblicher Entwickler der Programmierprache SeL ist Andreas Matthias vom Hochschulrechenzentrum der GhK. siehe http://www.uni-kassel.de/hrz/anwendungen/matthias/sel/

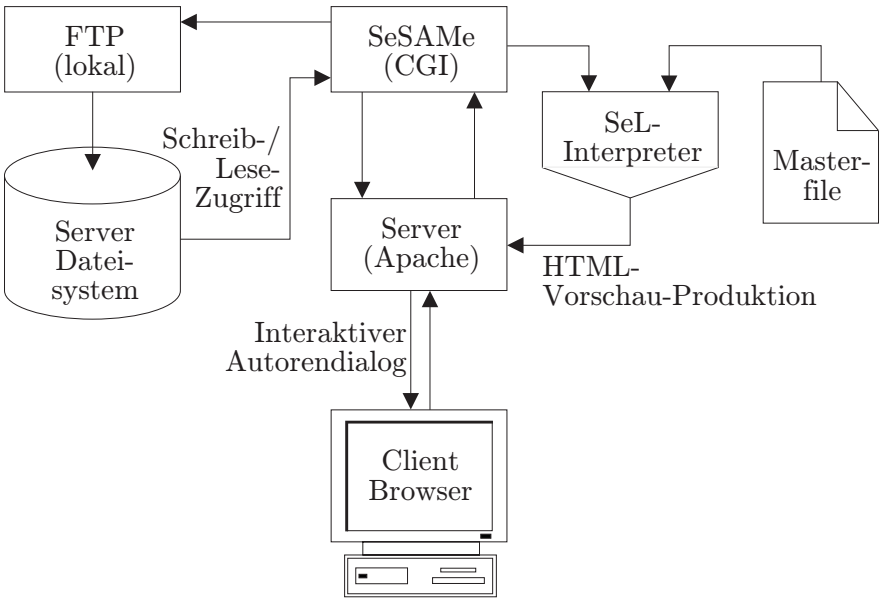


Abbildung 3.1.: SeSAMe–Autorendialog

so jede `.ghk`–Datei, eine Reihe von Variablen–Wertzuweisungen in der Sprache SeL. Die Menge der verwendeten Variablen wird innerhalb einer Domäne (möglicherweise, nicht aber notwendigerweise einer Netzwerk–Domain), z.B. auf einem WWW–Server durch den zuständigen WWW–Administrator spezifiziert. Sie umfassen sowohl die variablen Datenelemente des jeweiligen Dokuments als auch die dazugehörigen Metadaten. Diese Wertzuweisungen haben das folgende Format:

```
=TITLE          <SQD>Gesamthochschule Kassel - Homepage</SQD>;
=AUTHOR         <SQD>Webmaster</SQD>;
=AUTOR.MAILTO   <SQD>www@hrz.uni-kassel.de</SQD>;
```

Hierbei handelt es sich zunächst um eine Dokumentbeschreibungssprache, die funktional eine gewisse Ähnlichkeit mit XML aufweist[18]. Ein syntaktischer Unterschied zu XML besteht darin, daß die logische Hierarchie der Elemente nicht durch Verschachtelung, sondern (nach dem Modell von Sprachen wie JavaScript und Java) durch verkettete Elementnamen mit Punkten als Separatoren gekenn-

[18] Eine bidirektionale Kompatibilität zu XML ist gegeben, experimentelle Filter existieren bereits.

zeichnet wird. Falls Anführungszeichen in den zugewiesenen Werten erwünscht sind, können die zugewiesenen Textkonstanten mit den Sequenzen `<SQD>` und `</SQD>` abgegrenzt werden.

Weiterhin enthält jede `.ghk`–Datei einen Aufruf zum Einbinden einer SeL–Codebibliothek, die wir Masterfile nennen[19]. Das Masterfile enthält das eigentliche SeL–Programm. Der SeL–Interpreter erhält die `.ghk`–Datei und das Masterfile in einem einzigen kontinuierlichen Datenstrom und „sieht“ ein SeL–Programm mit Wertzuweisungen im oberen Teil und SeL–Code im unteren Teil, der diese Variablen zu einem Output verarbeitet. Bei der Ausführung des entstandenen Programms wird im Masterfile vorgesehener Code (z.B. HTML) produziert, der an den variablen Stellen des im Masterfile beschriebenen Dokuments die Elemente enthält, die vom Autor in der `.ghk`–Datei angegeben wurden. Technisch ist das Masterfile folglich eine zentral bereitgestellte dynamische Formatvorlage, die automatisch mit den jeweiligen Inhalten versehen wird und deren Verhalten bestimmt.

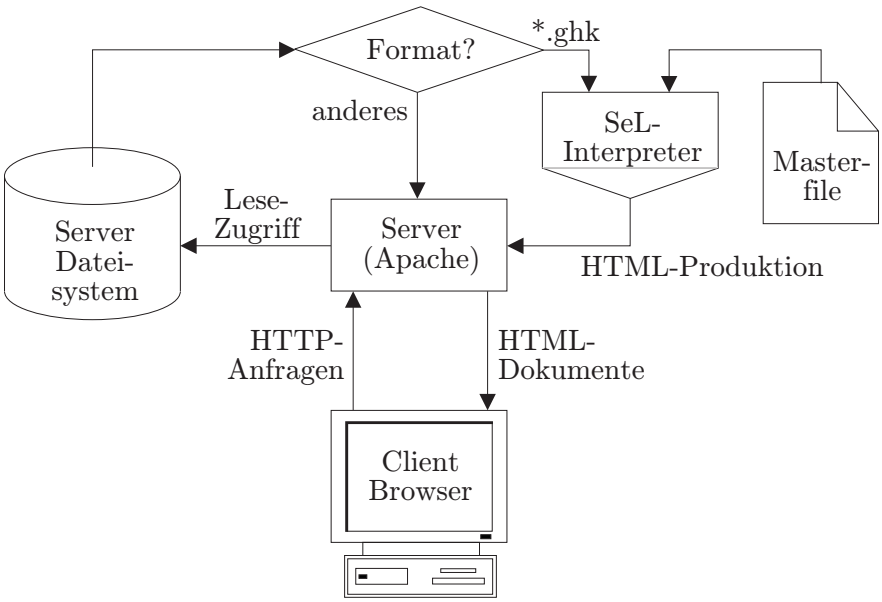


Abbildung 3.2.: SeSAMe/SeL Dokument–Service

[19] Details zur Struktur und Funktionsumfang von SeL können dem Nutzerhandbuch entnommen werden: Matthias, Andreas und Fischer, Thomas: [34]

Diese Architektur gestattet die zentrale technische und gestalterische Verwaltung des WWW–Angebots und gleichzeitig die nutzerfreundliche Herstellung von Online–Dokumenten. Seit SeSAMe Anfang 1999 auf dem WWW–Server der GhK in Betrieb gegangen ist, wurde das zentrale Masterfile mit dem Erscheinungsbild der Universität vielfach überarbeitet und an neue Erfordernisse angepaßt. Solche Eingriffe sind für die Autoren der Dokumente transparent. Das heißt, sie brauchen ihre einzelnen Dokumente nicht zu bearbeiten und können gestalterische und technische Veränderungen erst aus Leser–Perspektive erkennen. Weitere Redesign–Projekte werden auf diese Art ebenfalls keine Arbeit der Autoren an ihren Inhalten sondern lediglich Modifikationen an der zentralen Formatvorlage erfordern.

Die Syntax der Wertzuweisungen in SeL–Dateien ist weitaus einfacher als textuelle HTML–Programmierung. Viele fortgeschrittene SeL–Autoren sind daher in der Lage, SeL–Dateien im Klartext mit Hilfe einfacher Text–Editoren herzustellen und mit einem FTP–Client in das Dokumenten–Verzeichnis des WWW–Servers zu übertragen. Bei diesem manuellen Verfahren entstehen mitunter Syntaxfehler, die viele dieser fortgeschrittenen Nutzer beim *Debugging* der Ausgabe im Browser entnehmen, oder interaktiv auf der UNIX–Kommandozeile des WWW–Servers beheben können. Dieses Verfahren ist jedoch nicht geeignet, um WWW–Autoren mit durchschnittlichen Kenntnissen im Umgang mit dem PC bei der alltäglichen Publikation von Online–Dokumenten behilflich zu sein.

In SeL steht eine sogenannte „Masterfile–Override–Funktion“ zur Verfügung, durch die die Ersetzung der in den `.ghk`–Dateien angegebenen Masterfile–Aufrufe mit Aufrufen alternativer Masterfiles *durch den Leser* gestattet wird. Dadurch wird das Dokument vor dem Servieren nicht mit dem Standard–Masterfile sondern mit einem anderen Masterfile bearbeitet und damit ein anderes Output–Format produziert.

Mit dem Anhängen eines Masterfile–Namens an den URL einer `.ghk`–Datei nach dem Muster `http://www.uni-kassel.de/?style=plain` wird das jeweilige Dokument mit einer alternativen Formatvorlage, also einem grundlegend anderen Verhalten seiner Daten serviert. Mit dieser Funktion war es möglich, transparent für die Nutzer das gesamte WWW–Angebot im SeSAMe–Format

zusätzlich in einem für Braille–Zeilen optimierten Design für Blinde und Sehbehinderte bereitzustellen. Eine ähnliche Erweiterung mit einem Masterfile für WAP–Clients befindet sich zur Zeit in einem experimentellen Stadium. Dies ist möglich, weil SeL eine formatunabhängige Sprache zur Text–Manipulation ist und nicht an ein bestimmtes Dokumentbeschreibungsverfahren oder Protokolle gebunden ist. Es kann außer HTML auch XML, WAP, LaTeX(folglich auch Postscript und pdf) und sonstige textbasierte Dokumentbeschreibungen für beliebige Clients und Ausgabemedien erzeugen.

Die beiden genannten Erweiterungen des SeSAMe–Dienstes belegen, daß es durchaus möglich ist, in autorensystembasierten Dokumentverwaltungen Verhaltensweisen bereitszustellen, die bei Entwurf und Entwicklung des Autorensystems nicht berücksichtigt worden sind. Dies erfordert zwar technische Eingriffe durch Experten mit Programmierkenntnissen, ist aber andererseits ein Schlüssel zur Bereitstellung didaktischer Funktionalität, die zunächst nicht Bestandteil des Autorensystems war. Auf diese Weise kann folglich einer der zentralen Kritikpunkte, die ich im Rahmen der Problemanalyse aufgeführt habe, mit SeSAMe behoben werden.

SeSAMe ist an sich zunächst kein Autorensystem mit didaktischer Intention. Doch es ist ein Modell, anhand dessen der in dieser Arbeit beschriebene Entwurf illustriert werden kann. Dies gilt insbesondere für das Prinzip der Wiederverwendung von Expertenwissen, das Modell der Datenkompatibilität und Interoperabilität, die Trennung von Autorensprache und GUI, die Trennung von Design und Inhalt sowie für die Anpassung an nicht vorhergesehene Designkriterien, die zunächst jenseits des Weltmodells von SeSAMe liegen. Das letztere Problem erscheint angesichts der Limitierungen existierender Autorensysteme als besonders schwierig. SeSAMe wurde wiederholt an Designkriterien angepaßt, die zunächst nicht berücksichtigt wurden und hat seine diesbezügliche Flexibilität mehrfach unter Beweis stellen können. Es wurde fortlaufend an neue Anforderungen angepaßt, hat aber nie frühere Funktionalität verändert oder verhindert.

Zur Zeit wird SeSAMe[20] Version 2 an verschiedenen Institutionen getestet und in Betrieb genommen. SeSAMe2 ist um einen weiteren Abstraktionsschritt von SeSAMe abgeleitet: Es ist ein Autorensystem–Generator, der für die variablen Elemente der Dokumente einer Domäne die Schnittstellendefinitionen (in Form von HTML–Code), Kommentare, zulässige Werte–Bereiche, Hilfetexte und Fehlermeldungen vom Administrator der Domäne erhält und daraus ein auf die Domäne angepaßtes Autorensystem generiert. Mit dieser Entwicklung hat SeSAMe aber noch längst nicht das Limit seines Potentials erreicht. Während die dynamische Code–Generierung für die Autoren–Schnittstellen zur Zeit durch Perl und durch JavaScript erfolgt, kann die Erzeugung des Autoren–Frontends wie jedes normale Dokument durch SeL selbst erfolgen. Hierzu müßte mit der Masterfile–Override–Funktion lediglich ein Masterfile aufgerufen werden, das dem Autor ein normales Dokument nicht in seinem Standard–Design sondern als editierbare Elemente von Formularen präsentiert. In der Terminologie der objektorientierten Programmierung ist das Produkt einer `*.ghk`–Datei und eines Masterfiles nur eine von unendlich vielen möglichen Instanzen des Dokuments. Andere Instanzen (wie z.B. ein Autorensystem, das die Attribute des Dokuments zum Editieren und Abspeichern präsentiert) können durch die Bereitstellung und Auswahl des entsprechenden Masterfiles erzeugt werden. Auf diese Weise wird es in Zukunft möglich sein, sämtlichen dynamisch generierten SeSAMe–Code mit SeL selbst herzustellen.

3.4. Pattern–Design und musterbasierter Unterricht

Für bösartige Probleme gibt es keine einfachen Regeln und Anweisungen, mit denen bewährte Lösungsstrategien auf neue Kontexte angewendet können und mit Sicherheit zu einem „idealen“ Ziel führen. Daher ist die (hoffentlich kompetente und wohlreflektierte) Übertragung bzw. Generalisierung persönlicher Erfahrungen auf neue, (bösartigerweise immer andere) Kontexte die einzige Möglichkeit, methodisch zu entwerfen. Die Formulierung solcher persönlicher Erfahrungen in Musterlösungen gestattet ihre Weitergabe unter Entwerfern ver-

[20] Der Autor von SeSAMe 2 ist Christoph Beck, HRZ Uni Kassel.

gleichbarer Entwurfs–Problemklassen. Der erste, der dieses Vorgehen als praktische Entwurfsmethode beschrieben hat, ist der Architekt Christopher Alexander. Seine Methode blieb, wie ich später ausführlicher diskutieren werde, nicht unkritisiert, jedoch erfährt sie in jüngerer Zeit, gerade in Aufgabenbereichen mit sowohl technischen als auch sozialen Komponenten, eine immer weitere Verbreitung.

In der jüngeren Vergangenheit fanden Mustersprachen über den Informatik–Unterricht sogar ihren Weg in die angewandte Didaktik. Dort wiederum hängt der Wiederverwendung von Methoden und Inhalten ein negativer Beigeschmack an, der erst in den vergangenen beiden Jahrzehnten durch wissenschaftliche, vor allem aber praxisorientierte Argumentation relativiert werden konnte. Grell und Grell[21] diskutieren in ihrem Buch *Unterrichtsrezepte* das erziehungswissenschaftliche Pendant zum Pattern Design.

3.4.1. Eine Mustersprache

Der Architekt und Design–Theoretiker Christopher Alexander hat mit der ersten Auflage seines Buches *A pattern Language* im Jahre 1977 eine Anleitung für das *zeitlose* Entwerfen und Bauen von Städten, Nachbarschaften, Häusern, Räumen und Gärten in Form der ersten als solchen bezeichneten Mustersprache[22] vorgelegt. Diese Mustersprache ist eine stark strukturierte Sammlung von Lösungen wiederkehrender architektonischer und stadtplanerischer Problemstellungen und umfaßt ein Vokabular bestehend aus 253 sogenannter *Patterns*, die sich in drei Kapitel unterteilen: Gebäude, Städte und Konstruktion. Jedes Pattern enthält (auch in den Mustersprachen, die Alexanders Vorbild folgten) mindestens vier typische Elemente: Namen zur eindeutigen Identifizierung, eine Beschreibung des zu lösenden Problems, die Lösung selbst sowie eine Diskussion der Konsequenzen des Musters und seiner Lösung. Alexander erklärt:

> „Jedes Muster beschreibt zunächst ein in unserer Umwelt immer wieder auftretendes Problem, beschreibt dann den Kern der Lösung

[21] Grell, Jochen und Grell, Monika: [23]

[22] Alexander, Christopher Ishikawa, Sara und Silverstein, Murray: [1]

> dieses Problems, und zwar so, daß man diese Lösung millionenfach anwenden kann, ohne sich je zu wiederholen.“[23]

Um einen Eindruck von Alexanders Mustersprache zu vermitteln, zitiere ich in Anhang B ein Muster, das das Problem der Positionierung von Türen in Räumen diskutiert[24].

Von den in diesem und den folgenden Abschnitten erwähntern Disziplinen, die Musterpsrachen hervorgebracht haben, ist die Architektur wohl diejenige, deren Produkte sich gegenüber den strengsten Anforderungen in bezug auf ästhetische Originalität bewähren müssen. Durch Alexander inspirierte „Musterbücher“ für das Bauen waren praktisch nicht in der Lage, Architektur hervorzuringen, die diesem Anspruch gerecht geworden ist. Aus Sicht der meisten Architekten haben Mustersprachen als Entwurfswerkzeuge daher versagt und heute werden Mustersprachen in der Architektur weitgehend abgelehnt. Diese Ablehnung erfolgt ohne genauere Betrachtung der näheren Ursachen für das Scheitern und den Gründen für die Tatsache, daß Programmierer und Lehrer mit diesem Werkzeug erfolgreich arbeiten. Dies hat letztlich zu der paradoxen Situation geführt, daß sich diejenige Disziplin, die das Konzept Mustersprache ursprünglich hervorgebracht hat, von ihm abgewendet hat, während es sich in anderen Fachbereichen wachsender Popularität erfreut.

[23] Alexander, Christopher et al., ebd. S. X

[24] Alexander, Christopher et al., ebd., S. 977-979

3.4.2. Wiederverwendbare Muster im Entwurf objektorientierter Software

Sharing of software [at MIT Artificial Intelligence Lab]
was not limited to our particular community;
it is as old as computers, just as sharing of recipes
is as old as cooking.[25]

Der Aufbau sogenannter objektorientierter Programmiersprachen wie C++ oder Java gestattet im Gegensatz zu funktionalen oder prozeduralen Programmiersprachen die Herstellung modularer, in sich geschlossener Programmkomponenten, die als *Objekte* bezeichnet werden. Neben dieser engeren, fachspezifischen Bedeutung wird dieser Begriff auch in einem allgemeineren, unverbindlicheren Sinn benutzt, um generell Komponenten von Systemen zu beschreiben. Die engere und die weitere Bedeutung des Objektbegriffs haben gemeinsam, daß sie auf Abgeschlossenheit und funktionale Autonomie, definierte und manipulierbare Eigenschaften sowie die Fähigkeit zur Kommunikation mit anderen Objekten als zentrale Eigenschaften von Objekten basieren[26].

Das Programmieren mit abgeschlossenen Komponenten gestattet vergleichsweise effiziente Software–Entwicklungsprozesse, in denen Objekte als abstrakte Lösungen für Problemklassen hergestellt und wiederverwendet werden können. So muß einerseits der individuelle Programmierer wiederkehrende Lösungen nicht in jedem Projekt neu herstellen, andererseits können größere Projekte leicht in ihre Elemente aufgeteilt und von Gruppen bearbeitet werden. Die daraus erwachsenen neuen technischen Möglichkeiten für die Herstellung von Software erzeugen einen Bedarf nach entsprechenden administrativen Methoden, mit denen die neuen Formen der Kommunikation, Archivierung und Dokumentation bewältigt werden können.

[25] Stallman, Richard: [44]

[26] In diesem Sinne kann die hier vorgeschlagene Mustersprache als objektorientiert bezeichnet werden, obgleich sie nicht notwendigerweise in objektorientierten Sprachen implementiert ist.

Erich Gamma und andere[27] haben diesen Bedarf erkannt und 1995 mit ihrem Buch *Design Patterns – Elements of Reusable Object-Oriented Software* einen Entwurf für eine entsprechende Software–Entwicklungsstrategie vorgelegt. Dieses Buch ist sehr schnell zu einem Klassiker im Bereich der Software–Entwicklung geworden und mittlerweile füllen Bücher, deren Titel deb Begriff „Pattern Design“ in sich tragen und die sich ähnlichen Fragestellungen widmen oder sogar direkt an *Design Patterns* anknüpfen, ganze Regale. Es entstanden einige internationale Konferenzen sowie regelmäßig vernanstaltete Workshops zu diesem Thema. Zu nennen sind hier insbesondere die jährliche Konferenz *PloP*[28] in den Vereinigten Staaten sowie deren europäische Schwester–Konferenz *EuroPloP*[29] In einschlägigen Kreisen nennt man Gamma und seine Co–Autoren seit dem Erscheinen ihres Buches die „Gang of Four“.

Ihr Werk referiert explizit auf die Arbeit Christopher Alexanders und knüpft an dessen Konzept und Muster–Format an. Die Idee von *Design Patterns* ist, bewährte Musterlösungen für Klassen wiederkehrender Probleme zu Dokumentieren und zu kommentieren, wie es auch Alexanders *A pattern Language* vorschlägt. Ebenso wie Alexander rufen sie weiterhin zur Organisation eines anhaltenden Prozesses der Muster–sammlung über das eigene Buch hinaus auf.

Die Probleme, für die Gamma Lösungen anbietet, sind sehr technischer Natur. Sie besprechen typische Probleme, wie sie bei der Entwicklung von Software auftreten. Probleme des Aufbaus und und der Organisation von Datenstrukturen beispielsweise vermitteln sich Nicht–Programmierern leider wesentlich schwieriger sich als das Problem von Türen in Raum–Ecken Nicht–Architekten vermittelt. Ich zitiere in Anhang C dennoch exemplarisch eines der Muster aus Gammas Mustersprache um, Lesern aus anderen Disziplinen wie der Didaktik und dem Design einen Eindruck davon zu vermitteln, wie stark das Format und der Anspruch der Kommunizierbarkeit und Wieder–Anwendbarkeit durch die Formulierung abstrakter, generischer Lösungen sowie die Darstellung relevanter

[27] Gamma, Erich, Helm Richard Johnson, Ralph und Vlissides, John: [20]

[28] PloP2001: http://jerry.cs.uiuc.edu/ plop/plop2001/

[29] Die Webseite für die im Juli 2000 veranstaltete EuroPloP2000 ist unter http://www.coldewey.com/europlop2000/ abrufbar.

Kontextinformationen dem Ansatz Alexanders entspricht.

Obgleich nicht explizit, so beziehen sich Gamma und seine Co–Autoren zumindest implizit an mehreren Stellen auf das Konzept freier Software (insbesondere auf das GNU–Projekt[30]). Es ist offensichtlich, daß sich die freie Kommunikation bewährter Musterlösungen nur schwer mit den Prinzipien kommerzieller Software vereinbaren läßt. Daher findet ein wirklich freier Umgang mit Software Patterns hauptsächlich im Beriech freier Software statt, während viele kommerzielle Software–Entwickler vergleichbare Kulturen zumindest betriebs–intern pflegen. Ausnahmen bilden einzelne Projekte kommerzieller Entwickler, die diese explizit als „Open Source"–Projekte deklarieren, wenn sie dadurch ihre wirtschaftlichen Interessen nicht gefährdet sehen.

3.4.3. Pattern Design in der Didaktik

Susan Lilly war zur Zeit der Veröffentlichung ihres Artikels *Patterns for Pedagogy*[31] (etwa ein Jahr nach dem Erscheinen von Gammas *Design Patterns*) Informatik–Lehrerin an der Object Technology University von IBM in Maryland, USA. Gemäß dem Business–Slogan „Use what we sell" begann sie 1996, gemeinsam mit einem Team von weiteren, technisch ausgebildeten Lehrern, die Methode der musterorientierten Software–Entwicklung auf tägliche Aufgaben und Probleme des Unterrichts anzuwenden. Das Ergebnis dokumentierte sie anhand zweier ausgewählter Muster und beschreibt ihre Intention und den Erfolg ihres Ansatzes folgendermaßen:

> „Because we are all 'techies' with teaching experience, rather than professional curriculum developers with education degrees, we decided to 'use what we sell' (or at least what we teach). That is, we have tried to apply object–oriented analysis and design techniques to developing our course. One of the most interesting consequences of this decision was our discovery that the course we were developing was full of reusable *pedagogical design patterns*.

[30] http://www.gnu.org/

[31] Lilly, Susan: [31], S. 96-95

> [..] A course is composed of things (objects) like lectures, examples, exercises, case studies, demonstrations, student work products, examinations, and so on. These types of things (classes) come in different flavours (subclasses); for instance, a 'lecture' might be a traditional stand–up speech with slides, a multimedia–supported presentation, or an interactive, instructor–facilitated session.
>
> [..] My team has found tremendous value in the use of [..] patterns for developing and teaching a long, very challenging course. Pedagogical patterns have given our course a better defined internal structure. They also helped make that structure more explicit, both to the course developers and the students. As our curriculum continues to grow, these patterns provide the developers with a common vocabulary for describing ways to put together the pieces to build new courses and customize versions of our existing courses."[32]

Auch Lillys Mustersprache zitiere ich in Anhang D mit einem Beispiel–Muster für die Planung des Aufbaus von Unterrichtseinheiten[33].

Obgleich Lillys Artikel relativ kurz ist und eher den Charakter einer knappen Fallstudie hat als den einer Grundlegung für einen neuen didaktischen Ansatz, hatte er und das in ihm vorgestellte Konzept eine starke und nachhaltige Wirkung in der anglo–amerikanischen Didaktik der Informatik. Als Reaktion auf Lillys Ansatz wurde das „Pedagogical Patterns Project"[34] gegründet, das bewährte Lösungen didaktischer Probleme sammelt und in einem Format veröffentlicht, das deutlich durch die Vorarbeit von Alexander, Gamma und Lilly geprägt ist.

Das Pedagogical Pattern Project ist eine auf Kommunikation und Austausch ausgerichtete Gemeinde von Lehrern, die hauptsächlich eine Reihe internationaler Konferenzen zum Thema „Object Technology" zur Sammlung, Diskussion, Weiterentwicklung und Verbreitung ihrer Muster nutzt. Diese Konferenzen um-

[32] Lilly, Susan, ebd., S. 96

[33] Lilly, Susan, ebd., S. 93

[34] Pedagogical Patterns Project: http://www-lifia.info.unlp.edu.ar/ppp/

fassen z.B. TOOLS USA[35] und OOPSLA[36] in den Vereinigten Staaten sowie OT[37] und ECOOP[38] in Europa. Auf seiner Webseite stellt das Projekt sich und seinen Ansatz wie folgt dar:

> „The process of training, retraining, and, most of all, educating people in object technology is an ongoing challenge which has many unanswered questions. While many good pedagogical ideas are presented at OO conferences and published in proceedings and journals each year, very little has been done to collate the effective practices of many OO educators into one publication. The purpose of the pedagogical patterns project is do just that, to create a publication which is similar to what Susan Lilly (in 1/96 Object Magazine) refers to as 'reusable pedagogical design patterns'."[39]

Die Webseite enthält eine Datenbank mit einer Sammlung didaktischer Muster, die in vorrangig Probleme des Unterrichts objektorientierter Programmierung behandeln, prinzipiell jedoch durchaus auf generelle didaktische Fragestellungen übertragbar sind. Während die Lehrer, die didaktische Muster veröffentlichen und anwenden, dieses Verfahren als sehr wirkungsvoll und effektiv bezeichnen, wird didaktischen Mustern von Seiten der generellen Erziehungswissenschaft bislang wenig Aufmerksamkeit entgegengebracht. Dies kann unter anderem damit begründet werden, daß hier bereits seit langem vergleichbare Konzepte unter anderen Namen thematisiert werden. Dies gilt insbesondere für den in der Erziehungswissenschaft vieldiskutierten Rezept–Begriff.

---

[35] Technology of Object–Oriented Languages and Systems: http://www.eiffel.com/tools/

[36] Conference on Object–Oriented Programming, Systems, Languages and Applications: http://oopsla.acm.org/

[37] Object Technology: http://www.ot2001.org/

[38] European Conference for Object–Oriented Programming: http://www.ecoop.org/

[39] http://www-lifia.info.unlp.edu.ar/ppp/

3.4.4. Rezepte für das Unterrichten

Die Idee der Bereitstellung einer Methode, die die Weitergabe von angewandtem didaktischem Expertenwissen gestattet, fand bereits Jahre vor Lillys Artikel durch Monika und Jochen Grell Unterstützung in der deutschen Didaktik. Laut Grell und Grell basieren das Unterrichten (so wie auch andere Interaktionsstrukturen) auf Konventionen und Ritualen sowie auf wiederkehrenden Handlungsmustern und Lösungsstrategien. Diese Muster und Strategien bezeichnen Grell und Grell als (Unterrichts–) Rezepte. Zufälligerweise verwenden sie in diesem Kontext mitunter auch (offenbar als Synonym für „Rezept“) den Begriff „Muster“[40]

Der Rezept–Begriff ist in der Didaktik wesentlich älter als die Publikation von Grell und Grell. Er ist in hohem Maß ambivalent und nicht unumstritten. Rezepte sind in der unterrichtlichen Praxis lange Zeit abgelehnt worden, weil sie als unreflektierte, unangepaßte Wiederholungen vergangener Handlungen verstanden wurden. Mittlerweile finden Rezepte mehr und mehr Akzeptanz, weil ihr pragmatischer Wert ebenso erkannt wird wie die Tatsache, daß ihre Anwendung nicht zwangsläufig unreflektiert und unangepaßt erfolgen muß.

Hilbert Meyer führt im Rahmen seiner Diskussion des Rezept–Begriffs eine von Lehramts–Studenten zusammengetragene Liste von 132 sehr knappen (sie umfassen meist nur einen Satz), gängigen Unterrichts–Rezepten als „Rezepte–Steinbruch“ auf[41]. Dieser umfaßt Rezepte wie z.B.:

Wenn die Klasse laut wird, nie hinter dem Lehrerpult sitzen bleiben, sondern aufstehen. Das erhöht die Autorität.

Daß Rezepte dieser Form als eindeutige Handlungsanweisungen mit dem Anspruch auf unreflektierte Anwendbarkeit auf Ablehnung stoßen, ist nicht sonderlich überaschend. Aus Meyers Diskussion des Rezept–Begriffs geht jedoch eine ganze Reihe von Definitionen und Thesen hervor, was Rezepte denn eigentlich sind und wozu sie dienen. Dabei kommt er zu dem Schluß, daß Rezepten praktische und durchaus sinnvolle Funktionen für unterrichtliches Planen und

[40] siehe „Die Tradition des Erarbeitungsmusters“ in Grell und Grell: [23], S. 49 ff.

[41] Meyer, Hilbert: [36], S. 32 ff.

Handeln zukommen. Wichtig jedoch sei der *reflektierte Umgang* mit ihnen.

Lehrer erfinden nicht in jedem Detail ihrer Arbeit das Rad neu. Gute und gewissenhafte Lehrer stellen vielleicht jedes Detail ihrer Arbeit bei jeder neuen Anwendung wieder in Frage, aber sie werden trotzdem Ideen, Strategien und Lösungen anwenden, die nicht völlig neu sind. Dazu gehören die Sprache, die sie im Unterricht verwenden, didaktische Rituale, Unterrichtsformen sowie auch Inhalte und Methoden des Unterrichts. Diese Wiederverwendung von didaktischem Wissen erspart nicht nur viel Arbeit in der Vorbereitung, Durchführung und Evaluation des Unterrichts, zu einem wesentlichen Teil ermöglicht sie den Unterricht erst, da er ohne generalisierende Operationen durch den Umfang des erforderlichen Zeit– und Arbeitsaufwands schlicht nicht praktikabel wäre. Unsere Sinne nehmen von der Umwelt weit mehr wahr, als wir tatsächlich auswerten. Der Prozeß, mit dem wir umfangreiche Umwelt–Reize reduzieren verwendet die Mechanismen der Generalisierung, der Klassenbildung und der Reduktion. Ohne diese Strategien wären wir kaum in der Lage, unsere Sinneswahrnehmungen effizient auszuwerten und auf sie zu reagieren. Wie jede komplexe soziale Tätigkeit ist auch das Unterrichten ohne diese Strategien nur schwer vorstellbar, da selbst gänzlich alltägliche und selbstverständliche Interaktionsmuster wie die Sprache, die wir sprechen, Unterrichtsformen und auch die Prinzipien der Gerechtigkeit und der Gleichbehandlung auf Generalisierung, Abstraktion, Klassenbildung und Konventionalisierung beruhen. Hinzu kommt der weitere Vorteil, daß ein bewährtes Interaktionsmuster als eigenständiges Objekt mit einem Namen versehen, dokumentiert, kommuniziert, diskutiert und wiederverwendet werden kann.

Das Konzept des unterrichtlichen Kochens mit Rezepten wurde etwa in den 1980er Jahren nach einer „*langen Tradition der Verunglimpfung*“ bedingt rehabilitiert. Es wird seither im allgemeinen nicht als eine „*Theorie unterhalb der Gürtellinie*“[42] abgetan sondern vielmehr zu einer *reflektierten Anwendung* unterrichtlicher Rezepte geraten.

Vielleicht ist es Zufall — vielleicht aber auch nicht: Der Zeitpunkt dieser Neu-

[42] vgl. Meyer, Hilbert: [36], S. 27 ff.

bewertung von „Rezepten“ deckt sich ziemlich genau mit dem Moment, in dem Mikrowellentechnologie und die zugehörigen Fertiggerichte ihren Weg in unsere Supermärkte und Küchen fanden. Bis zu diesem Zeitpunkt schien Kochen entweder umfassende Expertise oder die Anwendung von Rezepten zu erfordern. Das zwischen diesen Polen liegende Kontinuum von mangelhaften bis hin zu ausgereiften Kochkünsten wurde durch Mikrowellen–Fertiggerichte um eine neue Dimension erweitert. Solche Gerichte kann jeder kochen — auch ohne tatsächlich kochen zu können, denn zubereitet werden sie eigentlich dort, wo sie hergestellt und verpackt werden. Mikrowellen–Köche kochen daher nicht wirklich selbst, sie wärmen lediglich Vorgefertigtes auf. Als unsere Kochkultur also lernen mußte, daß die Anwendung von Rezepten keineswegs der unqualifizierteste bzw. uneffektivste Ansatz ist, begann auch die Didaktik, dies zu verstehen. Im Bereich des computervermittelten Unterrichts erschienen zur selben Zeit die ersten Lern–CD-ROMs, die den Mikrowellen–Gerichten insofern entsprechen, als sie auch nicht dort „zubereitet“, hergestellt und verpackt werden, wo sie eingesetzt werden. Mikrowellen–Gerichte und Lern–CD-ROMs können von Köchen und Lehrern nicht ohne weiteres an konkrete Bedingungen angepaßt oder individualisiert serviert werden[43]. Sowohl in der Küche als auch im Unterricht macht umfassende Expertise die Anwendung von Rezepten überflüssig und vielleicht sind die Produkte erfahrener Experten manchmal wesentlich besser als Resultate von Rezept–Anwendungen. Doch Rezepte richten sich in beiden Anwendungsgebieten an diejenigen, die einerseits nicht in der Lage sind, individuelle Produkte von Hand herzustellen und andererseits kein Interesse am Aufwärmen unveränderlicher Kost haben. Es ist daher wichtig, Rezepte nicht als Problemzähmungen wie die Regeln des Schachspiels oder die Bauanleitung eines IKEA–Regals sondern vielmehr als anpaßbare Angebote zu verstehen. Kochrezepte können an individuelle Vorlieben oder das Fehlen und die Verfügbarkeit einzelner Zutaten angepaßt werden. Je nach Gelingen sind die Gerichte dann dem Urteil der Kochenden und der Bekochten entsprechend *besser* oder *schlech-*

[43] *Es ist prinzipiell durchaus möglich, Mikrowellen–Gerichte und Lern–CD-ROMs inhaltlich und formal zu verändern. Wenn jedoch die dafür erforderliche Zeit, Qualifikation und Ausstattung vorhanden sind, gibt es diesbezüglich keinen Grund mehr für die Verwendung dieser Fertigprodukte — es können ebenso individuelle Produkte hergestellt werden.*

ter als andere. Eine Abweichung von den Regeln gezähmter Probleme hingegen führt immer zu einem Resultat, das im Urteil der Problemzähmer *falsch* ist, egal wie sinnvoll die Anpassung im jeweiligen Anwendungskontext war.

3.5. Patterns im Unterrichtsentwurf — Pro und Kontra

Pattern–Design ist keine problemfreie Methode. Es eröffnet eine Reihe von Fragen und Zweifeln, und seine Ursprünge (und mitunter auch deren Erfolg) erscheinen weder wissenschaftlich fundiert noch praktisch solide. Bei der Lektüre von Alexanders Pattern Language fragt man sich etwa als Architekt, oder hier vielleicht auch als Lehrer, wie man ein Rezept „millionenfach“ anwenden könne, ohne sich dabei je zu wiederholen. Ganz sicherlich war Alexander von seinen Ideen sehr stark überzeugt. Anders ließe sich auch nicht seine Feststellung erklären, die Richtigkeit seiner Muster sei empirisch abgesichert, was tatsächlich nicht der Fall ist. Ein Kritiker der Pattern Language ist Jean-Marie Protzen, der die Mustersprache in einer Rezension[44] kritisiert.

> „Is this truth amenable to corroboration, revision or refutation through empirical research? I shall argue that [Alexanders'] contention — that the validity of patterns can be empirically tested — is at least, a questionable proposition.“[45]

Die Befragungen, die Alexander nach eigenen Angaben zur Belegung seiner Annahmen durchgeführt hat, sind ganz sicher nicht dazu geeignet, diese als wissenschaftlich abgesichert zu bezeichnen. Ganz im Gegenteil: Seine vagen und wenig operationalisierten Aussagen sind als solche nicht als allgemeingültig belegbar. Seine subjektiven gesellschaftlichen Visionen sind ebenso auf seinen Kulturkreis ausgerichtet, wie einige seiner geographisch abhängigen Aussagen nur auf der Nordhalbkugel des Globus gültig sein können. Protzen wirft ihm folgerichtig die Verkündung einer Ideologie vor. Diese Ideologie könne nur dadurch erfolgreich

[44] Protzen, Jean-Marie: [47], S. 191-194

[45] Protzen, Jean-Marie: [47], S. 192

sein, daß sie nicht, wie Alexander seine Leser glauben läßt, Fakten vorgibt, sondern durch das Vermeiden klarer Aussagen auf einem Niveau verbleibt, das im Gegenteil vage genug ist, nicht wiederlegt werden zu können.

Obgleich ich dieser Kritik zustimme, möchte ich dem entgegnen, daß eine Mustersprache, wenn sie sich als Antwort auf bösartige Probleme versteht, a priori nicht unideologisch sein kann. Die Objektivierung, die für die Formulierung wertfreier, allgemeingültiger Aussagen und Handlungsanweisungen erforderlich ist, ist hier durch das Fehlen formaler Bezugsrahmen nicht möglich. Entgegen Alexanders Überzeugung kann eine solche Methode nicht objektiv sein und sich somit auch nicht dem Vorwurf der ideologischen Vorbelastung entziehen. In diesem Sinne können wir auch Prognosen, Aussagen und Handlungsanweisungen etwa im Bereich der Politik nicht als objektiv und ideologiefrei bezeichnen, was ebenso auf Prognosen, Aussagen und Handlungsanweisungen in unterrichtlichen Rezepten zutrifft. Die jeweils anvisierten Problembereiche sind eben bösartig.

Um einen weiteren problematischen Gesichtspunkt in Alexanders Aussagen zu diskutieren, möchte ich Vitruvius, den Urvater der Architekturtheorie heranziehen. Er unterteilt im antiken Rom um 80 v.Chr. das bauliche Entwerfen und dessen Produkte in drei Aspekte:

- Utilitas: die Nützlichkeit oder Problemangemessenheit
- Firmitas: die Haltbarkeit oder Robustheit
- Venustas: die Schönheit oder Aesthetik

Der nach meiner Ansicht prekärste Anspruch Alexanders ist seine implizite Garantie für das Erreichen von Venustas: Seine Mustersprache verspricht (gemeinsam mit seiner theoretischen Grundlegung, *The timeless Way of Building*) nachdrücklich einen Weg zur Schaffung von Schönheit und Harmonie. Unser Schönheitsbegriff ist jedoch in hohem Maße subjektiv, und der offensichtliche Einfluß gesellschaftlicher Ideale der 1970er Jahre auf Alexanders Theorie zeigt, daß dieser Begriff in ebenso hohem Maße temporär ist und damit nicht allgemeingültig sein kann. Begriffe wie Nützlichkeit und Haltbarkeit sind leichter operationalisierbar, da sie widerspruchsfreier an neutrale Kriterien geknüpft

werden können: Ein Haus ist nur *schön*, wenn ein jemand es *schön findet* aber ein Haus kann als *robust* bezeichnet werden, wenn es z.B. einen Sturm *übersteht* und ein Kochplatz kann als *nützlich* bezeichnet werden, wenn er etwa für die Ernährung einer Familie *benutzt* werden kann.

Susan Lilly ist sich der subjektiven Qualität ihrer Erkenntnisse bewußter als Alexander. Sie räumt ausdrücklich deren Angreifbarkeit ein:

> „One might argue that these patterns are not new or original, that they reflect designs that may be found in other courses.“[46]

Sie begegnet diesen Einwänden im voraus mit der Feststellung, daß didaktische Musterlösungen einzig aus ihrem Recycling–Charakter ihren hohen praktischen Wert schöpfen. Mit Verweis auf Erich Gamma betont sie, daß dessen Verdienst nicht die Erfindung der von ihm dokumentierten Patterns ist, sondern die Bereitstellung einer Methode, die bewährtes Expertenwissen kommunizierbar und wiederholt anwendbar macht. Noch deutlicher wird dieser Gedanke am Beispiel Christopher Alexanders, der das Plazieren von Türen in Raumecken schließlich nicht erfunden hat.

Weiterhin ist anzumerken, daß es sich bei Utilitas, Firmitas und Venustas um *Produktqualitäten* handelt. Entwürfe können sich jedoch auch durch Qualitäten der *Prozesse* auszeichenen, die zu dem jeweiligen Produkt führen. Daher versuchen wir, Produkte nicht nur nützlich, haltbar und schön zu entwerfen, sonderen die ihnen zugrundeliegenden Herstellungsprozesse auch effizient, schnell, ressourcensparend und koordiniert zu gestalten. Dadurch steigt der Wert von Produkten (wie zum Beispiel die erzielte Nützlichkeit) relativ zu dem Preis, den man für sie zahlen muß. Daß dies durch Mustersprachen begünstigt wird, thematisieren die genannten Autoren leider nur indirekt. Die Weitergabe bewährter Musterlösungen mit dem Ziel, Nützlicheit und Entwicklungseffizienz zu erreichen, erscheint mir letztlich weitaus vernünftiger als das Versprechen von Schönheit.

[46] Lilly, Susan, ebd. S. 95

Die hier vorgestellten muster– bzw. rezeptorientierten Entwurfsansätze ähneln sich sowohl in ihrem gemeinsamen Anspruch auf praktische Anwendbarkeit als auch in Aufforderungen an ihre Leser, selbst aktiv Muster zu entwickeln, sie auszuformulieren und sie zu kommunizieren.

> „Wir hoffen natürlich, daß viele Leser und Benutzer unserer Sprache den Versuch machen werden, diese Mustersprache zu verbessern — daß sie sich der Mühe unterziehen werden, echtere, profundere Invarianten zu finden —, und wir hoffen, daß diese echteren Muster, die mit der Zeit entdeckt werden, nach und nach in eine gemeinsame Sprache Eingang finden, die wir alle teilen können.“[47]

Ein zentraler Aspekt der zitierten Beispiele — und zugleich eine offensichtliche Stärke von patternorientiertem Design — ist, daß Entwerfen und die praktische Umsetzung von Wohn–, Arbeits– und Lernumgebungen als Prozesse sozialer Interaktion verstanden werden und als solche zum Erreichen gesetzter Ziele führen. Patterns bieten entwerfenden Gruppen eine Kommunikationsbasis in Form einer gemeinsamen Sprache. Wenn diese wie in den aufgezeigten Beipielen detailliert dokumentiert werden, bieten sie eine Chance zur Bereicherung von Entwurfsprozessen über die Grenzen der ursprünglichen Entwerfer– Gruppe hinaus und werden letztlich zu Fundamenten von Entwurfskulturen.

Die Unterschiede zwischen bösartigen und zahmen Problemen können — wie ich in Abschnitt 2.4 bereits dargestellt habe — vereinfachend auf die Frage reduziert werden: *„Existiert für die Beurteilung einer (Sub-)Lösung für ein Problem ein formaler Bezugsrahmen oder nicht?“*. Kann diese Frage in bezug auf ein konkretes Problem mit „ja“ beantwortet werden, kann das Problem als zahm bezeichnet werden. Falls nicht, ist es ein bösartiges Problem.

Einen solchen Bezugsrahmen liefern sowohl Alexander, Gamma, Lilly als auch Grell und Grell. Sie zähmen ihre jeweiligen Entwurfsproblematiken und gestatten sowohl Entwerfern als auch deren Kritikern, die Qualität vorgeschlagener oder realisierter Lösungen anhand eines wachsenden und sich evolutionär entwickelnden gemeinsamen Maßstabes zu bewerten. Dennoch unterscheidet sich

[47] Alexander, Christopher et al., ebd., S. XV-XVI

das Konzept der Mustersprache von strengen Problemzähmungen wie Gesetzen, DIN-Normen oder Spielregeln:

So rufen alle genannten Autoren explizit zur Variation und zur Erweiterung ihrer Sprachen und deren Muster sowie zu deren Verbreitung auf. Dies macht das Konzept der Mustersprache zum Austauschmedium für praktisch gesammeltes Wissen. Somit sind Mustersprachen ideal geeignet als Hilfe für Designer und Entwickler, die etwa am Anfang ihrer Tätigkeit noch ohne weitreichende praktische Erfahrung sind oder sich in sich sehr schnell entwickelnden Wissensgebieten zu behaupten haben. Dies macht sie zu einem interessanten Verfahren in dem sich schnell wandelnden Bereich der didaktischen Software–Entwicklung.

Ästhetische Qualität (die Alexander neben Nützlichekeit und Haltbarkeit verspricht und besonders betont) kann nicht intersubjektiv beurteilt werden (über Geschmack läßt sich schließlich streiten!) und folglich wurde Alexanders Mustersprache insbesondere innerhalb der Entwurfswissenschaften kritisiert: Eine Mustersprache sei mit ihren impliziten Vorgaben a priori ideologisch, nicht allgemeingültig und basiere auf dem Geschmack oder den Ideen weniger. Protzen kritisiert insbesondere Alexanders Allgemeingültigkeits- und Wahrheitsanspruch und begegnet ihnen mit den strengen Maßstäben wissenschaftlicher Problemzähmung, deren Erfüllung Alexander und seine Co–Autoren zwar für sich in Anspruch nehmen, angesichts derer ihre Musterprache jedoch versagen muß.

Die Qualität von Mustersprachen liegt darin, daß sie rational auf bewährten Erfahrungen aufbauende *Innovation* und die Formulierung praxisorientierter Handlungsanweisungen gestatten, wozu Wissenschaft streng genommen nicht in der Lage ist. Dabei trägt allein der Lehrer (wie in der Problemanalyse in dieser Arbeit gefordert) die zentrale Verantwortung für Auswahl und Einsatz von Inhalten und Methoden. Es ist daher von hoher Wichtigkeit, die Anwender von Mustersprachen über ihre Rolle und Verantwortung in Kenntnis zu setzen. Hellmuth Walter schreibt in diesem Zusammenhang bereits 1979:

> „[Rezepte] sollten eindeutig unter dem Vorbehalt angeboten werden, daß Wissenschaft, auch *praxisbezogene* Wissenschaft gegenwärtig vorzüglich nur dazu dienen kann, den Lehrer für Probleme

> und *Handlungsalternativen* zu sensibilisieren. Darüber hinausgehende Handlungsanweisungen (Rezepte) sollten auf dem Hintergrund eben dieser kritischen Problemsensibilität vom Lehrer sorgfältig abgewogen und in ein auf ein konkretes unterrichtliches Feld bezogenes Handeln umgesetzt werden (ein Vorgang, der mit einem ‚wissenschaftlichen' Unterricht nichts zu tun hat!). Letzteres bleibt dabei immer dem Lehrer überlassen, hier kann kein Psychologe und — im Speziellen — Lerntheoretiker generell gültige Handlungsanweisungen liefern."[48]

Die folgende (von mir übersetzte) Liste ist ein Format für didaktische Muster. Es wurde entwickelt und veröffentlicht vom „Pedagogical Patterns Project"[49] und wird bereits in der Lehrpraxis angewendet.

Name: Der Name des Patterns

Datum: Datum der letzten Änderung

Autor: Name der Person, die das Pattern entwickelt hat

Zusammenfassung: kurze Beschreibung (Zusammenfassung) des Patterns

Zielgruppe/Kontext: Für welche Lerner in welchem Kontext ist das Pattern geeignet?

Zwänge: Was macht das Problem zu einem Problem?

Lösung: Die Lösung, die das Pattern für das Problem bietet

Diskussion: Entstehende Fragestellungen zum Inhalt, Umsetzung und Konsequenzen

Besondere Hilfsmittel: Hilfsmittel, die für den Einsatz des Patterns erforderlich sind (Dinge, die Nutzer des Patterns normalerweise nicht zur Hand haben)

[48] Walter, Hellmuth, [59], S. 19

[49] http://www-lifia.info.unlp.edu.ar/ppp/format.htm

Gegenanzeigen: Wann ein Muster nicht benutzt werden sollte, auch etwaige kulturelle Abhängigkeiten

Verwandte Muster: Autoren von Mustern mögen die WWW–Seite des „Pedagogical Pattern Projects“ verfolgen und eventuell verwandte Muster unter diesem Stichwort erwähnen

Beispiele: Exemplarische Anwendungen (wer, wo usw.)

Referenzen: Jegliche Zitate oder Referenzen, die durch ihren Beitrag zu einem Pattern zu nennen sind

3.6. Open Source

We did not call our software „free software“,
because that term did not yet exist;
but that was what it was.[50]

Unter Open–Source wird im allgemeinen „freie Software“ verstanden, was beide Bedeutungen des englischen Wortes *free* impliziert: *Frei verfügbar* und *gratis*. Freie Software hat es jedoch schon lange gegeben, bevor man von Open Source gesprochen hat. Genau genommen hat es einige Jahrzehnte gedauert, bis die Industrie überhaupt auf die Idee gekommen ist, Software zu *verkaufen*. Bis in die 70er Jahre hinein waren Betriebssysteme und Software–Werkzeuge selbstverständliches Zubehör von dem, was eigentlich verkauft wurde, nämlich Hardware.

Die Open Source–Bewegung hat ihre historischen Wurzeln in dem Bestreben, Programmierern und Anwendern (die Grenzen sind hier fließend) Software mit möglichst wenig Einschränkungen zur Verfügng zu stellen und gleichzeitig ihren Fortbestand und ihre Weiterentwicklung zu sichern oder zumindest zu vereinfachen[51]. Zu diesem Zweck wurden einige Lizenzen (z.B. die GNU Public License,

[50] Stallman, Richard: [44]

[51] vgl.: O'Reilly: [42], S. 9

kurz: GPL) verfaßt, die nun einfachheitshalber alle mit dem Oberbegriff „Open Source“ bezeichnet werden.

Das Open Source–Entwicklungsmodell ignoriert Unterschiede zwischen Entwicklern und Nutzern weitgehend. Menschen in beiden Rollen werden als schöpferische, denkende Wesen verstanden, die Werkzeuge brauchen, sich über dieses Bedürfnis austauschen und mitunter auch Werkzeuge entwickeln. Letzteres funktioniert entgegen den im Bereich Software–Entwicklung zuvor gesammelten Erfahrungen vermutlich dadurch, daß die beteiligten Entwickler keine von der Anwenderschaft losgelöste Gruppe darstellen: *Entwickler sind eine (wachsende) Untermenge der Anwender*. Der überraschende Effekt, etwa der Entwicklung des Open Source–Betriebssystems Linux, beruht auf der Tatsache, daß eine große Anwenderschaft, die man zunächst weder in bezug auf ihre Qualifikation noch auf ihre organisatorischen Strukturen für die Entwicklung eigener Software–Werkzeuge imstande hält, die erforderlichen Ressourcen dennoch aufweist. Umfangreiche Software–Projekte gingen zuvor nur aus streng zentralisierten und hierarchisierten Strukturen hervor. Die Linux–Entwicklung erfolgt jedoch (vom Testen und der Versionsverwaltung des Kernels abgesehen) völlig dezentral.

Das Resultat sind Werkzeugentwicklungen, in denen eine unmittelbare Beziehung zwischen Problem und Problemlöser gegeben ist, und die Möglichkeit, Design–Kriterien von Software durch den Anwender im realen Anwendungskontext zu bewerten, wie ich es oben für die Entwicklung und Anwendung von Autorensystemen fordere.

Die kommerzielle Software–Industrie hat bislang ein weniger emanzipiertes Bild von ihrer Kundschaft, was sich an zahlreichen Stellungnamhmen und Strategien festmachen läßt. Nachdem sich z.B. das Open-Source–Betriebssystem Linux bereits sehr früh als besonders leistungsstarke Netzwerk– und Internet–Plattform erwies, reagierte Microsoft auf diese wirtschaftliche Bedrohung mit verstärkter Online–Präsenz und verstärktem Angebot von Online–Inhalten in Form des „Microsoft–Networks“. Der ausbleibende Erfolg dieses Netzwerks belegt, daß es keine angemessene Antwort auf ein Entwicklungsmodell ist, deren Stärke in einem aufgeklärten Nutzermodell liegt. Diese Fehleinschätzung kann wohl kaum

besser illustriert werden als durch das folgende Zitat der Vize–Präsidentin des Microsoft Networks Laura Jennings:

> „The idea that people know what they want is wrong. They just have to be pulled through the web.“[52]

Ein weiterer Unterschied zwischen der „Kathedrale“ industrieller Software–Entwicklung und dem-Open Source–Modell besteht in den jeweiligen Kontakt– und insbesondere Kommunikationsstrukturen. Während die Software–Industrie intern in erster Linie auf vertikalen, hierarchischen Kommunikationsstrukturen basiert und in externem Kontakt mit Anwendern eine Pflichtaufgabe sieht, der mit mehr oder weniger kompetenten, Kostenpflichtigen „Helplines“ genüge getan wird, strebt der vergleichsweise chaotisch strukturierte „Bazar“ der Open Source–Gemeinde nach möglichst freier Verbreitung verfügbarer Information. Da es hier praktisch keine organisatorischen Hierarchien gibt, gibt es auch keine ressourcenhungrige, überdimensionale vertikale Kommunikation. Um freien Informationsfluß zu ermöglichen, haben Entwickler freier Software in verschiedenen Subkulturen eine Reihe von Dokumentations–Formaten entwickelt, über die kostenlose Einführungen und Handbücher, Referenzen etc. kommuniziert werden. Beispiele sind sogenannte „man–pages“, „Readme“–Dateien oder „Linux–HowTos“.

Die Veröffentlichung von Quelltexten als solche ist jedoch kein Allheilmittel für alle Probleme von Software–Design. Das Dilemma zwischen vordefinierten Interaktionsmodellen und schwieriger textueller Programmierung bei der Herstellung von Lernsoftware ist nicht auf das Entwicklungsmodell zurückzuführen, mit dem ein Autorensystem hergestellt wird. Auch scheint es von großer Wichtigkeit, daß Open Source–Projekte sehr früh eine möglichst große Öffentlichkeit und damit beitragende Programmierer finden[53]

[52] Zitat von Laura Jennings, nachzulesen z.B. unter http://home.earthlink.net/~idesomarch/

[53] Der WWW–Browser Netscape z.B. wurde nach einer mehrjährigen industrieellen Entwicklung Open Source (das sogenannte „Mozilla“–Projekt). Die Resultate der andauernden offenen Entwicklung scheinen wenig überzeugend. Während diese Arbeit verfaßt wird, gibt Sun Microsystems seine Pläne zur Veröffentlichung der Quelltexte von StarOffice bekannt. Auch dies ist

Open–Source ist ein Entwicklungs–Modell für die Herstellung von Software. Auf dieses Modell trifft auch Flechsigs Beschreibung unterrichtlicher Modelle zu: Es ist eine (retrospektive) *Rekonstruktion von Realität*[54] So, wie didaktische Modelle (wie z.B. *Frontalunterricht* oder *Fernunterricht*) Handeln zwar retrospektiv beschreiben, nicht aber verläßlich prospektiv anleiten können, hat auch der Begriff Open–Source nur retrospektive, deskriptive Qualitäten. Da sich keine zwei Open-Source-Entwicklungen gleichen und da jedes typische Merkmal von Open-Source in jedem praktischen Projekt anders umgesetzt wird, kann dieses Entwicklungsmodell weder konkrete Handlungsanweisungen liefern, noch das Gelingen eines Projekts garantieren. Genau dies wünschen sich Software–Entwickler jedoch ebenso, wie sich Lehrer konkrete Handlungsanweisungen und Erfolgsgarantien von didaktischen Modellen erhoffen. Sowohl konkret erforderliche Handlungen als auch die jeweiligen Maßstäbe, an denen Erfolg zu messen ist müssen jedoch von den genannten Entwerfern selbst im Rahmen ihrer unmittelbaren Planungskontexte definiert werden.

3.7. Zusammenfassung

Die Erziehungswissenschaft hat das in der Zusammenfassung des vorigen Kapitels unter 2.11 dargestellte grundsätzliche Defizit von Autorensystemen prinzipiell erkannt und beschrieben.

Zunächst stellt Flechsig fest, daß es sich bei didaktischen Modellen um retrospektive *Rekonstruktionen von Unterrichtswirklichkeit* handelt[55]. Hieraus können formale Probleme bei der prospektiven Anwendung didaktischer Modelle in der unterichtlichen Planung abgeleitet werden.

Meyers Diskussion des Rezept–Begriffs und sein Ratschlag zum *reflektierten Rezept–Einsatz* dokumentieren die Wichtigkeit der Rolle des reflektierenden Lehrers und seines Urteilens in konkreten Unterrichtssituationen. Die situa-

ein sehr großes Projekt, das nach einer langen industriellen Entwicklungsgeschichte eine offene Entwicklergemeinde sucht. Der erwartete Erfolg dieser „Kulturschocks“ blieb bislang aus.

[54] vgl. Flechsig, Karl-Heinz: [18], S. 5

[55] Flechsig, Karl-Heiz: [18], S. 5

tive Anpassung zuvor generalisierter unterrichtlicher Handlungsbeschreibungen durch den Lehrer ist von zentraler Wichtigkeit, da sie das von Flechsig beschriebene Defizit statischer didaktischer Modelle kompensieren kann.

In der Theorie und Entwicklung von Autorensystemen bleibt die hieraus erwachsene notwendige Konsequenz jedoch bislang aus: Autorensysteme sind bis heute nichts anderes als Bibliotheken von direkten, unreflektierten Handlungsanweisungen, die ohne die Möglichkeit freier methodischer Modifizierbarkeit auf neue Lernsituationen angewendet werden müssen.

Nachdem es gelungen ist, den Rezept–Begriff durch Überdenken, durch Modifikation seiner Bedeutung und durch das Fordern reflektierter Anwendung zu läutern, halte ich es für möglich und erforderlich, den Begriff *Autorensystem* ebenso grundsätzlich neu zu überdenken, es durch neue Konzepte neu zu definieren, Autorensystem–Anwender zur reflektieren Anwendung und Anpassung aufzurufen und die dazu erforderlichen technischen Mittel bereitzustellen.

Für die Entwicklung alternativer Konzepte ist es nötig, das in Autorensystemen implizite zentralisierte didaktische Planungsmodell durch ein dezentralisiertes Planungsmodell zu ersetzen, das der Rolle sowie der individuellen Verantwortung und Kompetenz Lehrender gerecht wird. Mustersprachen und offene Quelltexte stellen für diesen Zweck angemessene Mittel dar.

Zur Lösung der durch das in Abbildung 2.4 dargestellten Modells der Autorensystem–Entwicklung und –Anwendung schlage ich daher das in Abbildung 3.3 dargestellte Modell vor. Es weist zur gegenwärtigen Praxis zwei grundsätzliche Unterschiede auf:

Lehrer und Autorensystem–Entwickler entwerfen und implementieren Autorensysteme gemeinsam. Lehrer spielen dabei die Rolle von Entwicklern während sie selbst die prospektiven Anwender ihrer Produkte sind. Diese Konstellation ist typisch für Open-Source–Entwicklungen. Es entsteht eine Schnittmenge zwischen Programmierern und Lehrern: Einige Lehrende entwickeln Software–Lösungen für andere Lehrer, ohne diese selbst im eigenen Unterricht anzuwenden. Einige Programmierer entwickeln bei der Herstellung konkreter Werkzeuge ein so tiefes Verständnis der jeweiligen didaktischen Situation, daß sie teilweise

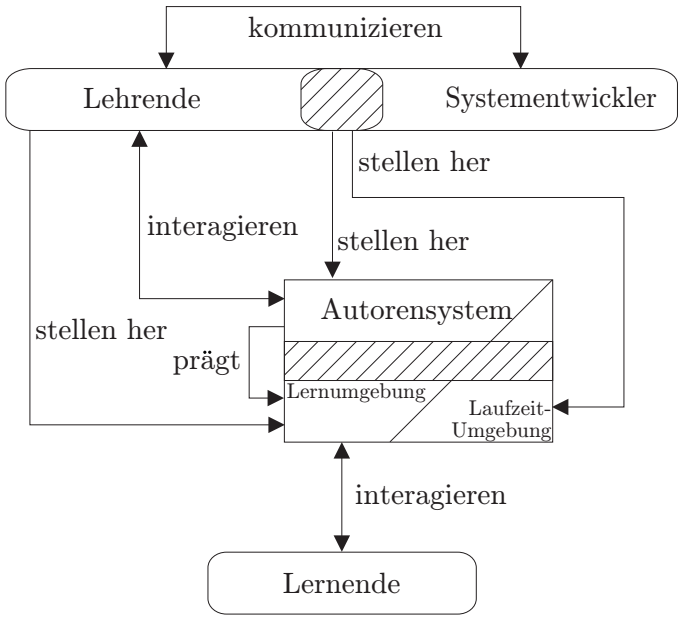


Abbildung 3.3.: Vorgeschlagenes Verständnis des Ablaufs vom Autorensystemdesign bis zum Unterricht

Lehr–Verantwortungen des jeweiligen Unterrichts wahrnehmen[56].

Produkte dieser Entwicklungen sind einerseits speziell für die Anforderungen bestimmter Lehrsituationen hergestellt. Andererseits resultiert aus der Erfordernis von Wiederverwendungen sowohl ein methodischer Prozeß der Generalisierung, Abstraktion, Moderation, Dokumentation und Archivierung als auch die Entwicklung abstrakter Werkzeuge für die Adaption konkreter Lösungen für neue Situationen. Es ist daher nicht möglich, diese Produkte eindeutig in „Autorensysteme“ (abstrakte Werkzeuge) und Lernumgebungen (konkrete Lehrmittel) zu unterteilen. Lernumgebungen und Werkzeuge zur Herstellung von Lernumgebungen wachsen zusammen und können nicht mehr eindeutig voneinander unterschieden werden.

[56] Beides habe ich während des Entstehens dieser Arbeit sowohl mehrfach beobachtet als auch selbst praktiziert.

4. Startvokabular der Mustersprache

Dieses Kapitel enthält eine Liste von 16 Musterlösungen für den computervermittelten Unterricht, die aus mehrjähriger Arbeit auf diesem Gebiet erwachsen ist. Sie bilden eine Auswahl, mit der ich neben der Dokumentierung meiner eigenen Forschung auch den kollaborativen Charakter der Entwicklungsmodelle *Mustersprache* und *Open Source* dokumentieren möchte, indem ich auch Beiträge von Kollegen zitiere, die ebenfalls in entwerferischen Kontexten mit vergleichbaren Methoden computervermittelt unterrichten.

Auf der Basis der theoretischen Diskussion in den vorangehenden Kapiteln dürfte es nun selbstverständlich sein, daß die Musterlösungen im praxisorientierten Teil dieser Arbeit keinen Anspruch auf endgültige Richtigkeit und Wahrheit haben. Es handelt sich um bewährte Lösungen, die sich bei erneuter Anwendung wieder bewähren, aber scheitern können, wie jeder Unterricht. Um die Wahrscheinlichkeit des Scheiterns zu minimieren, diskutiueren die jeweilugen Autoren auch ihrer Meinung nach relevante Kontext–Informationen, anhand derer erneute Anwendungen und ggf. erforderliche Anpassungen abzuwägen sind.

Die enthaltenen Muster sind zwar in der vorliegenden Fassung (wie ich hoffe) bereits hilfreich in der Planung, Durchführung und Auswertung computervermittelter Lehre. Sie bieten jedoch vielfältige Möglichkeiten zur Verbesserung und Weiterentwicklung. Als Beispiel hierfür sei die Beispiel–Implementierung des Musters „Automatische Erzeugung von Keyword–Listen“ (4.11) genannt, die sicherlich durch schönere und auch leistungsstärkere Algorithmen ersetzt werden könnte. Mit den Muster „Transklusion“ (4.5) und „Anzeige gleichzeitiger Nutzung von Online–Ressourcen“ (4.8) sind weiterhin auch Lösungen ent-

halten, bei denen Bedarf und Potential für Ergänzung besteht und die als Anregung zur Weiterentwicklung verstanden werden kann. Wie ich weiter oben bereits diskutiert habe, gibt es letztlich keinen Grund, nicht auch Fehlschläge und gescheiterte Lösungen als solche zu dokumentieren und in die Sammlung aufzunehmen.

Für die Auswahl der hier aufgezählten Muster sind insbesondere zwei Kriterien ausschlaggebend: Einerseits sind sie in fast allen Fällen an Fachbereichen für Design und Architektur, also in entwerferischen Kontexten entwickelt worden[1]. Bereits ein oberflächlicher Blick in den Unterrichtsalltag dieser Fachbereiche läßt deutliche Unterschiede zur Lehre in anderen Fachbereichen deutlich werden: So unterscheidet sich das im Bereich Design praktizierte didaktische Modell *Studiounterricht* deutlich vom reinen Vorlesungs– und Seminarbetrieb, wie er in anderen Disziplinen praktiziert wird. Ganz offensichtlich sind unterrichtliche Probleme, Planungen und Strategien — sowohl im Fall traditionellen Unterrichts als auch im Fall computervermittelten Unterichts — nicht unbedingt von anderen didaktische Aufgabenbereichen auf den Design–Unterricht übertragbar.

Daher habe ich mich bemüht, mit der hier vorgestellten Auswahl eine Sammlung von Mustern anzubieten, die leicht von entwurfs–orientiertem Unterricht auf andere didaktische Problembereiche übertragbar sind.

Das zweite Auswahlkriterium ist die Aufmerksamkeit, die den angesprochenen didaktischen Problemen durch verfügbare Autorensysteme und Lernumgebungen entgegengebracht werden. Obwohl es sich in allen Fällen um generische Lösungen wiederkehrender Probleme handelt, bietet meines Wissens heute kein kommerzielles Paket adäquate Lösungen für sie an.

Daher illustriert diese Mustersprache, wie ich meine, neben ihrer eigenen unmittelbaren praktischen Anwendbarkeit auch sowohl die Probleme kommerzieller, abgeschlossener Werkzeuge als auch die Fähigkeit, diese Probleme zu lösen.

[1] Ein Nebeneffekt des entwerferischen Entwicklungskontexts dieser Mustern ist die Betonung von Kommunikation und Kollaboration. Daher lösen viele der dargestellten Muster und Beispiele Probleme des Lernens im WWW. Diese Mustersprache kann aber selbstverständlich auch auf computervermittelten Unterricht ohne Nutzung des WWW angewendet werden.

4.1. Dynamische Erteilung von Zugriffsrechten

Name: Dynamische Erteilung von Zugriffsrechten	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich eingesetzt
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Zugriffe auf WWW–basierte Lernmittel können üblicherweise nur Verzeichnis–orientiert mit einem oder mit wenigen Passworten beschränkt werden, die redundant im Server–Dateibaum verteilt liegen und als solche manuell gewartet werden müssen. Dadurch werden selbst einfache inhaltliche und methodische Anpassungen von Lernmitteln zu arbeitsintensiven Prozessen mit hoher Fehlerwahrscheinlichkeit.	
Kurzdarstellung der Lösung: Das unten genannte Modul für den Apache Webserver gestattet die individuelle Ausführung eines beliebigen Programms, dem die Nutzerkennung und das Paßwort eines Nutzers (Lerners) übergeben werden kann. Der im Beispiel enthaltene Quelltext stellt als solches externes Programm Verbindungen zu einem FTP–Server her und überprüft dabei die Gültigkeit der Zugangsberechtigung.	
Gegenanzeigen: Die hier vorgestellte Lösung nutzt einen FTP–Server zur Feststellung der Gültigkeit von Passwörtern. Dies ist überflüssig, wenn in dem jeweiligen Netzwerk andere Mechanismen zur Authentifizierung (z.B. NIS oder Kerberos) zur Verfügung stehen.	
Besondere Hilfsmittel: `mod_auth_external` für den Apache Webserver liegt unter der folgenden Adresse zum Download bereit: `http://www.wwnet.net/~janc/software/mod_auth_external-2.1.5.tar.gz` Das Perl–Modul `Net::Telnet` liegt unter der folgenden Adresse zum Download bereit: `http://cpan.valueclick.com/authors/id/JROGERS/Net-Telnet-3.01.tar.gz`	
Referenzen: Das Modul `mod_auth_external` wurde von Jan Wolter entwickelt. `Net::Telnet` wurde von Jay Rogers entwickelt.	
Verwandte Muster: –	
Anmerkungen: Ich habe dieses Muster im Februar 2000 bei der Programmierung des Online–Lernsystems *The Silkroad Interactive Encyclopedia* an der School of Design der PolyU in Hong Kong entwickelt und erfolgreich eingesetzt.[2]	

[2] Ceccato, Cristiano et al. [7]. Weiterhin habe ich es im Herbst 2000 in der Lernumgebung *HermesUniverse* am selben Fachbereich erfolgreich eingesetzt.

Diskussion

Inhalte wie Lernmaterialien, Lerner–Rückmeldungen oder Testresultate werden in der Regel nicht völlig öffentlich zugänglich gemacht. Es gibt viele gute Gründe, ungewollte Zugriffe auf unterrichtliche Ressourcen zu untersagen — sei es beispielsweise, um persönliche Daten zu schützen, um einen Server vor der Last zu vieler Zugriffe zu bewahren oder Lerner vom gegenseitigen Kopieren ihrer Arbeitsergebnisse abzuhalten. Praktisch erfordert dies eine komplexe und sich laufend verändernde Matrix von Zugriffsbeschränkungen, die Lese– und Scheibrechte, Ausführungsrechte und Kommunikationsoptionen anhand folgender Gesichtspunkte beschreibt:

- Identität des Lerners und Gültigkeit seines Passwortes
- Unterrichtsphase (Kurs, Lektion etc.)
- Zeitpunkt während einer Lernsituation
- Lerninhalt
- Präsentationsform oder Medientyp

Die hier präsentierte Musterlösung basiert auf der Annahme, daß Lehrinstitutionen über irgendeine Form von Studenten–Datenbank verfügen. Dies ist in der Regel der Fall, auch wenn Lehrende diese Einrichtung, die typischerweise auf der Ebene der institutionellen Verwaltung betrieben wird, nicht selbst aktiv nutzen können. Gehen wir davon aus, diese Datenbank stellt im Netzwerk einen Dienst bereit, der die Überprüfung von Nutzerkennungen und Passwörtern gestattet. Dies gestattet eine Zugriffskontrolle auf institutioneller Ebene, mit der Angehörige der Institution von Nicht-Angehörigen unterschieden werden können. Nachdem auf diese Weise ein Zugriff grundsätzlich erlaubt wurde, steht im Verlauf des weiteren Dialogs die Nutzerkennung bei jedem Serverzugriff als Umgebungsvariable bereit und gestattet eine höher auflösende Schreib- und Leseberechtigung basierend auf konkreten Daten der jeweiligen Lernsituation.

Praktisch stellen Verwaltungen von Lehrinstitutionen den oben beschriebenen Dienst nicht zur Verfügung. Ein „Nachbau“ eines derartigen Dienstes etwa auf

Kursebene ist zeitaufwendig und führt zur Pflege redundanter Datensätze. Um dies zu vermeiden, können Lehrende aber Dienste mitbenutzen, die Lernenden auf institutioneller Ebene angeboten werden. In diesem Beispiel ist das der FTP–Dienst, der es Lernenden erlaubt, Daten auf einem zentralen Server zu lagern oder eigene Homepages vom Desktop–Rechner zum Webserver zu transportieren. Ein solcher FTP–Dienst für die Angehörigen einer Bildungseinrichung ist mittlerweile praktisch üblich. Die hier präsentierte Musterlösung kann selbstverständlich auch mit anderen Diensten wie *rlogin rsh* oder *ssh* implementiert werden. Dies hätte den Vorteil der automatischen Verschlüsselung bei der Weitergabe von Paßwörtern vom WWW–Server zu dem authentifizierenden Server. Es ist heute jedoch noch nicht gemeinhin üblich, allen Angehörigen von Schulen oder Universitäten Shell-Accounts einzurichten.

Nachdem der Webserver vom Client eine Nutzerkennung und ein Passwort erhalten hat, muss er diese von einem externen Programm, das den Standard–Authentifizierungsprozeß des Webservers ersetzt, prüfen lassen. Dieses Programm kann nun die Gültigkeit der Nutzerkennung und des Passwortes überprüfen, indem es damit eine Verbindung zu einem Service aufbaut, der nur bekannten Mitgliedern der Lehrinstitution bereitsteht. Basierend auf dem Gelingen oder dem Fehlschlagen dieses Verbindungsaufbaus kann das Authentifizierungsprogramm eine `1` oder eine `0` an den Webservser zurückgeben, um den Zugang auf institutioneller Ebene zu gewähren. Der Webserver Apache gestattet den Aufruf eines solchen externen Programms zur Authentifizierung, wenn er mit dem Modul „`Mod_Auth_External`“ kompiliert wird.

Der erste Schritt dieses Mechanismus ist eine Nutzeranmeldung zu Beginn einer Sitzung. Nachdem der Zugriff eines Nutzers auf diese Weise akzeptiert wird, wird die Benutzerkennung (nicht mehr das Paßwort) für den Rest der Sitzung gespeichert, bei der Abfrage jeder Ressource dem Server übergeben und steht für eine weitere Zugriffskontrolle auf Ressourcenebene zur Verfügung.

Zwar existiert ein bewährtes Protokoll zur Benutzer-Authentifizierung, das sowohl von gängigen Webservern als auch von gängigen WWW-Clients unterstützt wird — auf der Serverseite wird die erforderliche Integration des Autentifizierungsprozesses jedoch standardmäßig nicht ausreichend für die kom-

plexen Konstellationen in Lernsituationen unterstützt.

Lehrende benötigen einen Kontrollmechanismus, der auf der Basis von Informationen über Lernsituationen und über Lerner Zugriff auf Ressourcen gewährt oder untersagt.

Beispiel

```
#!/usr/bin/perl

# pwauth.pl: Institutional Access Authenticator Pattern
# (c)2000 by himself@tfischer.de
# Prerequisites:
# -Net::Telnet:
#  http://cpan.valueclick.com
#  /authors/id/JROGERS/Net-Telnet-3.01.tar.gz
# -mod_auth_external:
#  http://www.wwnet.net
#  /~janc/software/mod_auth_external-2.1.5.tar.gz
#
################## CONFIGURABLES ####################
# FTP host IP:
$HOST="127.0.0.1";
# FTP port:
$PORT=21;
# FTP server's 'connected' return code
$CONNECT=220;
# FTP server's 'password required' return code
$PASS_REQ=331;
# FTP server's 'logged in' return code
$LOGGEDIN=230;
#####################################################

use Net::Telnet ();

# get the name of this program
$PROG= $0;

# get the user name and password from STDIN
$USER=<STDIN>; chomp $USER;
$PASS=<STDIN>; chomp $PASS;

# talk to host...
$aut=new Net::Telnet (Telnetmode => 0);
$aut->open(Host => $HOST, Port => $PORT);
$line=$aut->getline;
if($line =~/^$CONNECT/){
  $aut->print("user $USER");
  $line=$aut->getline;
```

```
  if($line =~/^$PASS_REQ/){
    $aut->print("pass $PASS");
    $line=$aut->getline;
    # exception handler for guest access might be added as follows:
    # if(($line =~/^$LOGGEDIN/)||(($USER eq "guest")&&($PASS eq "g\
uest"))){
    if($line =~/^$LOGGEDIN/){
      $aut->print("QUIT");
      print STDERR "$PROG: id/passwd valid on $HOST - Accepted\n";
      exit 0;
    }
    else{
      $aut->print("QUIT");
      print STDERR "$PROG: passwd provided for $USER not valid on \
$HOST - Rejected\n";
      exit 1;
    }
  }
}
```

Für eine Beispiel–Installation dieses Programms mit dem WWW–Server Apache wird weiterhin eine Datei namens `.htaccess` in der obersten Ebene des zu schützenden Astes im Dateisystem mit dem Inhalt

```
AuthType Basic
AuthName "[Name der Lernumgebung]"
AuthExternal mykey
require valid-user
```

sowie der folgende Eintrag in der Konfigurationsdatei `httpd.conf` des WWW–Servers benötigt (an einer beliebigen Stelle oberhalb der Abschnitte für die Konfiguration „virtueller Hosts“):

```
AddExternalAuth mykey [pfad/zu/auth.pl]
SetExternalAuthMethod mykey pipe
```

4.2. Trennung von Autorensprache und Autoren–GUI

We want GUIs largely because they are convenient and because they are easy — or at least the GUI makes it seem that way. Of course, nothing is really easy and simple, and putting a nice interface on top of it does not change that fact. [..] By using GUIs all the time we have insensibly bought into a premise that few people would have accepted if it were presented to them bluntly: namely, that hard things can be made easy, and complicated things simple, by putting the right interface on them.[3]

Name: Trennung von Autorensprache und Autoren–GUI	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich eingesetzt
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Die Herstellung und Benutzung von Autorensystemen wird durch das Dilemma zwischen intuitiver, aber unflexibler visueller Programmierung und flexibler, aber schwierig erlernbarer textueller Programmierung erschwert. Eine Strategie, die oft als Lösung dieses Dilemmas dargestellt wird, ist die Implementierung von Scriptspachen innerhalb visuell orientierter Autorensysteme. Dies löst einerseits nicht das Problem der Erfordernis textueller Programmierung, andererseits sind eingebettete Scriptsprachen ebenso wie die visuellen Elemente von Autorensystemen an Entscheidungen gebunden, die zuvor von Systementwicklern getroffen wurden und schwer bzw. überhaupt nicht erweiterbar.	
Kurzdarstellung der Lösung: Die Bereitstellung einer problemorientierten Programmiersprache und darüberhinaus optionaler visueller Programmierumgebungen, die intern die Programmiersprache nutzen, kann die Abgeschlossenheit der Scriptsprachen aufheben und gleichzeitig einfache visuelle Programmierung ermöglichen.	
Gegenanzeigen: –	
Besondere Hilfsmittel: Eine Möglichkeit zur Implementierung dieses Prinzips stellt das SeSAMe/SeL–Paket dar.	
Referenzen: SeL und SeSAMe wurden an der Universität Gesamthochschule Kassel entwickelt und stehen unter `http://www.uni-kassel.de/hrz/anwendungen/matthias/design2/` zum Download bereit.	

[3] Stephenson, Neal: [56], S. 69

Verwandte Muster: Dieses Muster ist eine besondere Form der Anwendung des Musters „Trennung von Inhalt und Methodik“
Anmerkungen: –

Diskussion

Textuelle Programmierung ist flexibler als visuelle[4]. Durch die Einbettung textueller Programmiersprachen in Autorensysteme unterliegen diese Sprachen oftmals denselben Limitierungen, die auch die Flexibilität des visuell orientierten Autorensystems beschränken. Umgekehrt kann das grundlegende Versprechen von Autorensystemen, daß textuelle Programmierung durch sie überflüssig werde, mit dieser Strategie nicht gehalten werden.

SeSAMe und SeL nehmen sich dieses Problems an, indem sie die Lösungsstrategie umkehren: Anstatt eine Sciptsprache in eine visuelle Umgebung zu integrieren, erzeugt SeSAMe Autorensysteme, die auf SeL basieren und SeL–Code ausgeben: Visuelle Autorensysteme werden hier in eine textuelle Programmiersprache eingebettet.

SeL ist als relativ leistungsstarke Programmiersprache, die die Ausführung und Nutzung beliebiger externer Programme gestattet, sehr flexibel. Da die externen Programme (insbesondere unter UNIX–ähnlichen Betriebssystemen) in jeder beliebigen Sprache implementiert werden können, ist es flexibler als die meisten erweiterbaren MS-Windows–basierten Systeme, die die Herstellung externer Erweiterungen (falls überhaupt) meist nur in C++ oder Visual Basic gestatten.

Da dieses Muster nicht die Erfordernis textueller Programmierung aufzuheben vermag, sondern dies lediglich dort mindert, wo Experten Anfängern visuelle Material–Eingabemasken erstellen, besteht sein primärer Nutzen in der beschriebenen Flexibilität.

[4] vgl. Schiffer, Stephan: [54], S. 333 ff.

Beispiel

Ein SeL–Dokument, das einen minimalen Multiple–Choice–Test enthält, könnte wie folgt aussehen:

```
lib "test.mas";
=question    <SQD>Welche Zahlen sind die Ausgangswerte f&uuml;r die
Berechnung von Julia-Mengen?</SQD>;
=rightanswer <SQD>Komplexe Zahlen</SQD>;
=wronganswer <SQD>Primzahlen</SQD>;
```

Diese Beispiel–Datei trägt den Namen `julia.ghk`. Im Masterfile `test.mas` kann das Verhalten für die Präsentation des enthaltenen Tests sowie für die Auswertung von Antworten implementiert werden, das auf alle *Objekte* dieser Test–*Klasse* gleichermaßen anwendbar ist:

```
fwd question rightanswer wronganswer;

!runlevel1(void) /* present question */
<SQD><HTML><HEAD></HEAD><BODY>
Frage: </SQD>
$question
<SQD>
<FORM METHOD="POST">
<INPUT TYPE="HIDDEN" NAME="RUNLEVEL" VALUE="1">
<INPUT TYPE="RADIO" NAME="ANSWER" VALUE="</SQD>$rightanswer<SQD>">
</SQD>$rightanswer<SQD><BR>
<INPUT TYPE="RADIO" NAME="ANSWER" VALUE="</SQD>$wronganswer<SQD>">
</SQD>$wronganswer<SQD><BR>
<INPUT TYPE="SUBMIT" VALUE="Abschicken!">
</FORM>
</BODY></HTML></SQD>
;

!runlevel2(void) /* evaluate  answer */
if($_cgival("ANSWER")==$rightanswer)then
<SQD><HTML><HEAD></HEAD><BODY>
Die Antwort ist richtig!
</BODY></HTML></SQD>
else
<SQD><HTML><HEAD></HEAD><BODY>
Die Antwort ist falsch!
</BODY></HTML></SQD>
fi
;

!main(void)
$_cgiparse($_read())
```

```
if($_cgival("RUNLEVEL")=="")then
$runlevel1()
fi

if($_cgival("RUNLEVEL")=="1")then
$runlevel2()
fi
;

$main().
```

Das durch das Masterfile beschriebene Verhalten der Daten im Lernerdialog kann mit Hilfe der Masterfile–Override–Funktion von SeL[5] unterdrückt und mit dem in einem anderen Masterfile beschriebenen Verhalten ersetzt weren. Dies gestattet die Bereitstellung einer visuellen Maske zur Eingabe und zur Bearbeitung von Inhalten dieses Test–Typs.

Durch den Aufruf von `julia.ghk` mit dem URL `http://server/path/julia.ghk?author` wird die Datei nicht mit der Verwendnung von `test.mas` sondern mit `author.mas` serviert. Das folgende Beispiel einer entsprechenden Datei `author.mas` zeigt, wie für die Eingabe und Bearbeitung weiterer Tests dieses Typs eine visuelle Oberfläche erzeugt werden kann:

```
fwd question rightanswer wronganswer;

!runlevel1(void) /* present authoring GUI */
<SQD>
<HTML>
<HEAD></HEAD>
<BODY>
<FORM METHOD="POST">
Frage:
<INPUT TYPE="TEXT" SIZE="80" NAME="QUESTION" VALUE="</SQD>
$question
<SQD>"><BR>
Richtige Antwort:
<INPUT TYPE="TEXT" SIZE="40" NAME="WRONG" VALUE="</SQD>
$rightanswer
<SQD>"><BR>
Falsche Antwort:
<INPUT TYPE="TEXT" SIZE="40" NAME="RIGHT" VALUE="</SQD>
$wronganswer
<SQD>"><P>
Frage speichern in Datei:
```

[5] siehe Mattias, Andreas und Fischer, Thomas: [34]

```
<INPUT TYPE="TEXT" SIZE="10" NAME="FILE" VALUE="</SQD>
$_env("PATH_INFO")
<SQD>">
<INPUT TYPE="SUBMIT" VALUE="OK">
<INPUT TYPE="HIDDEN" NAME="RUNLEVEL" VALUE="1">
</FORM>
<BODY>
</HTML>
</SQD>
;

!runlevel2(void) /* store input */
=QUESTION $_cgival("QUESTION");
=WRONG $_cgival("WRONG");
=RIGHT $_cgival("RIGHT");
=FILE $_cgival("FILE");
=HELP $QUESTION<SQD>%%</SQD>$WRONG<SQD>%%</SQD>$RIGHT\
<SQD>%%</SQD>$FILE;
$exec("savefile.pl",$HELP)
<SQD>Die Frage wurde unter der Adresse "</SQD>
$FILE
<SQD>" gespeichert.</SQD>
;

!main(void)
$_cgiparse($_read())
if($_cgival("RUNLEVEL")=="")then
$runlevel1()
fi

if($_cgival("RUNLEVEL")=="1")then
$runlevel2()
fi
;

$main().
```

4.3. Trennung von Inhalt und Methodik

Name: Trennung von Inhalt und Methodik	**Letzte Änderung:** 5. Januar 2001
Synonyme: Trennung von Daten und ihrem Verhalten	**Status:** erfolgreich eingesetzt
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Die digitale Aufbereitung von Lernmaterial unter Zuhilfenahme von Autorensystemen führt zu zu Resultaten, in denen Lerninhalte und Methoden (etwa der Präsentation, Navigation oder Lernkontrolle) praktisch untrennbar miteinander verwachsen sind. Eine abstrakte und diskrete Beschreibung von Inhalten und Methoden und die jeweilige flexible Mehrfachnutzung ist nicht möglich. Das Resultat sind redundante Arbeitsprozesse in der Unterrichtsvorbereitung, reundanter Verbrauch von Speicherkapazität und mangelnde dynammische Anpassung der Lernmittel während der Lernprosesse. XML gestattet potentiell, Inhalte abstrakt und diskret auszuzeichenen, die zugehörigen, auf der Client–Seite interpretierten Dokumentbeschreibungen und „Style Sheets“ können dynamisches Verhalten von Lernmitteln jedoch nur sehr begrenzt beschreiben.	
Kurzdarstellung der Lösung: Inhalte können in SeL logisch ausgezeichnet und durch die Einbindung eines SeL–Masterfiles mit einem Standard–Verhalten ausgestattet werden. Durch die Masterfile–Override–Funktion kann dieses Standard–Verhalten übergangen und durch ein beliebiges anderes ersetzt werden, das in einem anderen Masterfile (etwa durch den jeweiligen Lehrer) bereitgestellt wurde und durch einen Anhang am jeweiligen URL aufgerufen werden kann. Einmal beschriebene Inhalte können somit auf beliebig viele Weisen dargestellt und navigiert werden.	
Gegenanzeigen: –	
Besondere Hilfsmittel: SeL–Interpreter	
Referenzen: –	
Verwandte Muster: Das Muster „Trennung von Autorensprache und Autoren–GUI“ ist ein Sonderfall dieses Musters.	
Anmerkungen: –	

Diskussion

Eine Unterrichtsmethode allein ist kein Unterricht: Sie braucht einen Inhalt, durch den sie sich in Unterricht manifestieren kann. Die in der Erziehungswissenschaft diskutierte *Interdependenz der unterrichts–strukturellen Momente* nennt im Hinblick auf unterrichtliche Planung *intentionale*, *inhaltliche*, *metho-*

dische und *medienbezogene* Entscheidungen[6]. Im Rahmen des computervermittelten Unterrichts läßt sich die Grenze zwischen *methodischen* und *medienbezogenen* Entscheidungen im Resultat schwer voneinander unterscheiden. Letztlich gilt die genannte Interdependenz unterrichts–struktureller Momente auch für den computervermittelten Unterricht, wobei sich durch die technisch dominierte Natur der Unterrichtspräsentation eine Verengung auf *inhaltliche* und *methodische* Interdependenz ergibt. Es resultiert auf der Meta–Ebene der Werkzeugentwicklung die zentrale Anforderung an die Architektur unterrichtlicher Autorensysteme, die (dynamische) Konstitution von Unterricht aus beiden Strukturelementen zu unterstützen.

Unterrichtsmethoden sind abstrakte Beschreibungen von Handlungen, die erst bei Anwendung auf konkrete Inhalte zu Unterricht werden — oder (um es in der Terminologie der objektorientierten Programmierung auszudrücken) erst durch die Verfügbarkeit von Daten können Unterrichts–Instanzen von Methoden gebildet werden. Inhalte sind in der täglichen Arbeit Lehrender typischerweise flüchtiger als Methoden: Während die Inhalte des Unterrichts praktisch täglich wechseln, verwenden Lehrende methodisches Wissen entweder ganz, in Teilen oder modifiziert wiederholt und bemühen sich gleichzeitig um eine Erweiterung ihres methodischen Repertoires.

Selbstverständlich werden auch Inhalte mehrfach verwendet, sei es im Rahmen von Wiederholungen oder beim Unterricht mehrerer Lerner(–gruppen). Mehrfach–Anwendung von Inhalten oder methodischen Verfahren erfolgen weiterhin zwischen einzelnen Lehrern, wenn sie sich gegenseitig mit Materialien aushelfen oder Anregungen zur Lösung methodischer Probleme geben. Eine Abbildung dieses nicht nur ökonomischen sondern schlichtweg praktisch notwendigen Recycling–Verfahrens im computervermittelten Unterricht scheitert an den gängigen Formaten, in denen digitale Lernmittel vorliegen: Ihre Elemente sind schwer zu extrahieren, methodische und inhaltliche Elemente sind weder als solche gekennzeichnet noch isolierbar und damit nicht *wiederverwendbar*. Folglich müssen sie für neue Unterrichts–Situationen immer wieder neu programmiert werden. Dieses Problem besteht unabhängig davon, ob textuell (in einer Pro-

[6] vgl. Jank, Werner und Hilbert Meyer: [27], S. 203 und S. 233 ff.

grammiersprache) oder visuell (in einem Autorensystem) programmiert wird.

Die erste Designentscheidung auf dem hier vorgestellten Weg zu praxisorientierten computerbasierten Lehrwerkzeugen ist die klare Trennung von Unterrichtsmethoden und Unterrichtsinhalten. Für die Software–Entwicklung bedeutet dies eine Trennung von Daten und Verhalten.

Beispiel

Als Beispiel kann jede Anwendung und Dynamische Auswahl von SeL–Masterfiles gelten. In diesen Fällen enhält das jeweilige angesprochene Dokument anstrakte Beschreibungen von Inhalten und Präsentationsmethoden. Das Verhalten wird in wählbaren Materfiles definiert. Die SeL–Dateien, die etwa auf dem Webserver der Universität Gesamthochschule Kassel unter `http://www.uni-kassel.de/` abrufbar sind, verhalten sich aus diesem Grund je nach ausgewähltem Masterfile unterschiedlich. Während ihr Standardverhalten die Darstellung als normale Webseiten vorsieht, gestattet das Anhängen entsprechender Masterfile–Aufrufe das Servieren derselben Inhalte z.B. in einem für blinde Leser optimierten Format.

Das im Muster „Trennung von Autorensprache und Autoren–GUI“ (4.2) ausführlich beschriebene Beispiel zeigt eine didaktische Anwendung dieses Prinzip, in welchem sich Inhalte Lernern und Lehrern insofern unterschiedlich präsentieren, als Lerner die Inhalte als Lerntests serviert bekommen, während Lehrer dynamisch erzeugte graphische Autorensysteme zur Pflege der Lerninhalte erhalten.

4.4. Tippfehler–Toleranz

Name: Tippfehler–Toleranz	**Letzte Änderung:** 5. Januar 2001
Synonyme: String–Approx, Typo Tolerance	**Status:** verfügbar
Autor(en): Thomas Fischer, Andreas Matthias	**Zielgruppe:** Autorensystem–Designer
Problem: Ubliche Formen maschineller Auswertung von Lerner–Eingaben haben nur sehr wenig mit menschlicher Auswertung von Texten oder vergleichbaren Lerner–Leistungen gemeinsam. Typischerweise überprüft der Computer lediglich eingegebenen Text mit vorgegebenen (als richtig definierten) Zeichenketten auf exakte Gleichheit. Die binäre Natur unserer Computer hat zur Folge, daß derartige Vergleiche als *true* oder *false*, also 1 oder 0 ausgewertet werden. Folglich werden selbst kleinste Abweichungen zwangsläufig als Nicht–Übereinstimmungen und entsprechende Lerner–Eingaben somit als nicht mit der jeweiligen „richtigen“ identisch und daher als *falsch* interpretiert. Dieses maschinelle Urteilen deckt sich nicht zwangsläufig mit Bewertungsmaßtäben, die menschliche Lehrer anlegen. Letztere urteilen (insbesondere außerhalb von Sprachunterricht) oftmals sehr tolerant in bezug auf Satzstellungen, Schreibweisen, Rechschreibung und Interpunktion. Während es also sehr einfach ist, Zeichenketten mit einfachen maschinellen Vergleichen auf *exakte* Gleichheit zu testen, ist es sehr schwierig, Texte auf *ungefähre* Gleichheit zu überprüfen.	
Kurzdarstellung der Lösung: Dieses Muster bietet eine Lösung an, mit der *unscharfe* Text–Vergleiche möglich sind. Die bei solchen unscharfen Zeichenketten–Vergleichen jeweils ermittelte Ungenauigkeit (bzw. Unschärfe) führt nicht zu einem binären Ergebnis in Form von *true* oder *false* sondern zur Rückgabe eines numerischen Wertes, der die minimale Anzahl von Transformationsschritten angibt, mit der die jeweilige Lerner–Eingabe in die als richtig definierte Zeichenkette verwandelt werden kann. Dieser Wert kann von der jeweiligen Software auf beliebige Weise verwendet werden.	
Gegenanzeigen: –	
Besondere Hilfsmittel: ANSI-C–Compiler	
Referenzen: Das unten aufgeführte Beispiel–Programm wurde von Andreas Matthias (HRZ/GhK) implementiert. Es wurde durch ein von Prof. Wilhelm Sanke (GhK) entwickeltes Programm zur Auswertung von Tippfehlern inspiriert, das über die Funktionalität dieses Beispiels hinausgehend auch eine visuelle Tippfehler–Indikation als Feedback an den Lerner bietet. Dieses unter Metacard entwickelte Programm wurde von Prof. Sanke am 10. Februar 1999 im Rahmen des Workshops „Multimedia in Lehre und Studium“ an der Universität Gesamthochschule Kassel präsentiert.	

Verwandte Muster: In Kombination mit dem Muster „Generisches Format zur Leistungsbewertung“ kann sehr einfach ein visuelles Feedback über die prozentuale „Richtigkeit“ einer Lernereingabe implementiert werden. Für die automatische Identifizierung von Wörtern, etwa für unscharf auswertende Lückentests aus Fließtext, kann das Muster „Automatische Erzeugung von Keyword–Listen“ herangezogen werden.
Anmerkungen: –

Diskussion

Im computervermittelten Unterricht ist es nur in sehr seltenen Fällen sinnvoll möglich, Lerner–Eingaben in Form von freiem Text maschinell auszuwerten. Dieses Problem wird in der Praxis typischerweise dadurch gelöst (bzw. gezähmt), daß vom Lehrer eine „richtige“ Antwort formuliert wird und die Lerner–Eingabe auf Gleichheit mit dieser Vorgabe überpfüft wird. Dieses Verfahren ähnelt der Konstruktion von Multiple–Choice–Tests, bei denen vom Lehrer ebenfalls „richtige“ und „falsche“ Antworten definiert werden. Während die Auswahl von Antworten eines Multiple–Choice–Tests (z.B. durch sogenannte „Checkboxen“) relativ eindeutig ist, kann die normalerweise übliche Überprüfung *exakter* Gleichheit freier Zeichenketten zu streng sein. Es ist daher mitunter durchaus sinnvoll, etwa bei der Abfrage gelernter Vokabeln bei wenigen Tippfehlern „ein Auge zuzudrücken“, um dem Lerner trotz der Fehler mitzuteilen, daß seine Antwort zumindest nicht ganz falsch war. Die unscharfe Auswertung alphanumerischer Lerner–Eingaben kann mindestens auf vier unterschiedliche Weisen genutzt werden:

Direktes Lerner–Feedback: Lerner–Eingaben werden im Rahmen von *Selbst–Tests* mit vorgegebenen Zeichenketten verglichen und aus der gemessenen Unschärfe eine unmittelbare Rückmeldung an den Lerner erzeugt.

Lehrer–Feedback: Lerner–Eingaben werden im Rahmen von *Lernkontrollen* mit vorgegebenen Zeichenketten verglichen und die gemessene Unschärfe an Lehrende übermittelt.

Filter für Lehrer–Auswertung: Die bislang obligatorische *menschliche Auswertung* von Fließtexten kann *effektiviert* werden, indem mit Hilfe unscharfer Vergleichs–Operationen „Toleranz–Schwellen“ definiert werden, oberhalb derer Lerner–Eingaben maschinell als „akzeptabel“ oder „bestanden“ ausgewertet und von der arbeitsintensiven menschlichen Auswertung ausgenommen werden.

Intelligente Suchmaschinen: Insbesondere in Lernsituationen, in denen Lerner nicht in ihrer Muttersprache unterrichtet werden, kann es sich als sehr hilfreich erweisen, wenn Material–Datenbanken nicht nur nach exakten

Treffern durchsucht werden können sondern bei erfolglosen Suchen maschinelle Hypothesen über naheliegende, eingegebe Zeichenketten ähnliche Suchbegriffe erzeugt werden können.

Beispiel

Das folgende Beispiel zeigt, wie die Funktionalität von SeL genutzt werden kann, um in einer CGI–basierten Online–Lernumgebung Eingaben auszuwerten und ein direktes visuelles Feedback über die Qualität der Eingabe zu erzeugen. Dieses Beispiel nutzt zur visuellen Ausgabe das Muster „Generisches Format zur Leistungsbewertung“ und zur Ermittlung von Test–Partikeln das Muster „Automatische Erzeugung von Keyword–Listen“.

Quellcode

```
/* ****************************************************************
fuzzymatch.c                          (C) Andreas Matthias, 2000.

see main() for a description of parameters. This is demonstration
code, so all error checking was left out to make it more readable.
Coding style favors readability over efficiency.

This is not production quality code.
**************************************************************** */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct retval_t {
  char *word;
  int val;
};

struct retval_t retval;

/* ****************************************************************
int exact_diff( char *w1, char *w2 )

Computes the "difference" between two strings (the number of
characters that are different). This is the value we are trying to
minimize  throughout the program.

w1, w2: strings to compare
return: difference value (0=strings are identical)
**************************************************************** */

int exact_diff( char *w1, char *w2 )
{
```

```
  int diff1=0, diff2=0, diff;
  char *p1, *p2;

  for( p1=w1,p2=w2; (*p1!='\0'&&*p2!='\0'); p1++,p2++ )
    if (*p1!=*p2)
      diff1++;

  for( p1=w1+strlen(w1),p2=w2+strlen(w2); (p1!=w1-1&&p2!=w2-1); \
p1--,p2-- )
    if (*p1!=*p2)
      diff2++;

  diff=(diff1<diff2?diff1:diff2);

  return diff+abs((strlen(w2)-strlen(w1)));
}

/* ************************************************************
char *insert_char( char *w, char c, int pos )

Inserts a character c into a string w at position pos. Operates
on a copy of w and does not modify the original string.

return: a pointer to the new string
************************************************************ */

char *insert_char( char *w, char c, int pos )
{
  char *ret, *p1, *p2;
  ret=malloc( strlen( w )+2 );
  for( p1=w,p2=ret; (*p1!='\0'&&(p1-w)!=pos); p1++,p2++ )
    *p2=*p1;
  *p2=c, p2++;
  for( ; *p1!='\0'; p1++,p2++ )
    *p2=*p1;
  return ret;
}

/* ************************************************************
char *remove_char( char *w, int pos )

Removes one character from the string w at position pos. Operates
on a copy of w and does not modify the original string.

return: a pointer to the new string
************************************************************ */

char *remove_char( char *w, int pos )
{
  char *ret, *p1, *p2;
  ret=malloc( strlen( w )-1 );
```

```
  for( p1=w,p2=ret; (*p1!='\0'&&(p1-w)!=pos); p1++,p2++ )
    *p2=*p1;
  *p1++;
  for( ; *p1!='\0'; p1++,p2++ )
    *p2=*p1;
  return ret;
}

/* ************************************************************
char *replace_char( char *w, char c, int pos )

Replaces the character in string w at position pos with c.
Operates on a copy of w and does not modify the original string.

return: a pointer to the new string
************************************************************ */

char *replace_char( char *w, char c, int pos )
{
  char *ret, *p1, *p2;
  ret=malloc( strlen( w )+1 );
  strcpy( ret, w );
  ret[pos]=c;
  return ret;
}

/* ************************************************************
char *rotate_right( char *w, int count )

Rotates the string w count times to the right. Characters "falling
off" the right end enter the string again from the left. Operates
on a copy of w and does not modify the original string.

Example: rotate_right("ABCD", 1) gives: DABC.

return: a pointer to the new string
************************************************************ */

char *rotate_right( char *w, int count )
{
  char *ret, backup;
  int i;
  ret=malloc( strlen( w )+1 );
  strcpy( ret, w );
  count=count%strlen(w);

  for (i=0; i<count; i++ )
    {
    backup=ret[strlen(ret)-1];
    ret=remove_char( ret, strlen(ret)-1 );
    ret=insert_char( ret, backup, 0 );
```

```
    }

  return ret;
}

/* ************************************************************
char *insert_loop( char *w1, char *w2, struct retval_t *retval )

Tries to insert (one after the other) all characters (from ASCII
32 to 127) into the string w2 at all possible positions, until the
exact_diff() value of the modified string against w1 is minimized.

return: a retval_t structure containing the best match and the
number of modifications made on the string (always 1 in this
function).
************************************************************ */

struct retval_t *insert_loop( char *w1, char *w2, struct retval_t\
 *retval )
{
  int pos, ch, newretval, oldretval;
  char *new;

  if ( strlen(w1)<=strlen(w2) )
    {
      retval->word=malloc( strlen( w2 )+1 );
      strcpy( retval->word, w2 );
      retval->val=0;
      return retval;
    }

  oldretval=99999;
  retval->word=NULL;

  for( pos=0; pos<=strlen(w2); pos++ )
    for( ch=32; ch<127; ch++ )
      {
new=insert_char( w2, ch, pos );
newretval=exact_diff( new, w1 );
if ( (newretval<oldretval) || (newretval==0) )
  {
  oldretval=newretval;
  retval->word=malloc( strlen( new )+1 );
  strcpy( retval->word, new );
  if (!newretval) break;
  }
      }
  retval->val=1;
  return retval;
}
```

```
/* ************************************************************
char *remove_loop( char *w1, char *w2, struct retval_t *retval )

Tries to remove (one after the other) each character from the
string  w2, until the exact_diff() value of the modified string
against w1  is minimized.

return: a retval_t structure containing the best match and the
number of modifications made on the string (always 1 in this
function).
************************************************************ */

struct retval_t *remove_loop( char *w1, char *w2, struct retval_t
\ *retval )
{
  int pos, newretval, oldretval;
  char *new;

  if ( strlen(w1)>=strlen(w2) )
    {
      retval->word=malloc( strlen( w2 )+1 );
      strcpy( retval->word, w2 );
      retval->val=0;
      return retval;
    }

  oldretval=99999;
  retval->word=NULL;

  for( pos=0; pos<strlen(w2); pos++ )
    {
      new=remove_char( w2, pos );
      newretval=exact_diff( new, w1 );
      if ( (newretval<oldretval) || (newretval==0) )
{
  oldretval=newretval;
  retval->word=malloc( strlen( new )+1 );
  strcpy( retval->word, new );
  if (!newretval) break;
}
    }
  retval->val=1;
  return retval;
}

/* ************************************************************
char *replace_loop( char *w1, char *w2, struct retval_t *retval )

Tries to replace (one after the other) each character from the
string  w2 with one other character in the range from ASCII 32 to
127, until  the exact_diff() value of the modified string against
```

```
w1 is minimized.

return: a retval_t structure containing the best match and the
number of modifications made on the string (always 1 in this
function).
********************************************************** */

struct retval_t *replace_loop( char *w1, char *w2, struct \
retval_t *retval )
{
  int pos, ch, newretval, oldretval;
  char *new;

  oldretval=99999;
  retval->word=NULL;

  for( pos=0; pos<strlen(w2); pos++ )
    for( ch=32; ch<127; ch++ )
      {
new=replace_char( w2, ch, pos );
newretval=exact_diff( new, w1 );
if ( (newretval<oldretval) || (newretval==0) )
  {
  oldretval=newretval;
  retval->word=malloc( strlen( new )+1 );
  strcpy( retval->word, new );
  if (!newretval) break;
  }
      }
  retval->val=1;
  return retval;
}

/* ************************************************************
struct retval_t *rotate_right_loop( char *w1, char *w2, struct \
retval_t *retval )

Tries to rotate right the string w2 a variable number of times n
(from n=1 to n=strlen(w2)) until the exact_diff() value of the
modified string against w1 is minimized. A rotate operation
always counts as ONE modification, regardless of the value of n.

return: a retval_t structure containing the best match and the
number of modifications made on the string (always 1 in this
function).
********************************************************** */

struct retval_t *rotate_right_loop( char *w1, char *w2, \
struct retval_t *retval )
{
  int count, newretval, oldretval;
```

```
  char *new;

  oldretval=99999;
  retval->word=NULL;

  for( count=0; count<strlen(w2)+1; count++ )
    {
      new=rotate_right( w2, count );
      newretval=exact_diff( new, w1 );
      if ( (newretval<oldretval) || (newretval==0) )
{
  oldretval=newretval;
  retval->word=malloc( strlen( new )+1 );
  strcpy( retval->word, new );
  if (!newretval) break;
}
    }
  retval->val=1;
  return retval;
}

/* ************************************************************
struct retval_t *gnr_loop( char *w1, char *w2, struct retval_t \
*retval )

"grab-n-run": Picks a character from the string w2 (each one
character  in turn) and "runs" with it through the string,
pausing at every  position to determine a value for exact_match()
of the new string  against w1. The running character is removed
from w2 during the run,  so that this function will be able to
match strings which have characters  transposed. This function
runs until the exact_diff() value of the  modified string against
w1 is minimized. A grab-n-run operation always counts as ONE
modification, regardless of the value of n.

return: a retval_t structure containing the best match and the
number of modifications made on the string (always 1 in this
function).
************************************************************ */

struct retval_t *gnr_loop( char *w1, char *w2, struct retval_t \
*retval )
{
  int pos, pos2, newretval, oldretval;
  char *new, *tmp1, backup;

  oldretval=99999;
  retval->word=NULL;

  for( pos=0; pos<strlen(w2)-1; pos++ )
    {
```

```
      backup=w2[pos];
      tmp1=remove_char( w2, pos );
      for( pos2=0; pos2<strlen(w2); pos2++ )
{
  if ( pos2==pos )
    continue;
  new=insert_char( tmp1, backup, pos2 );
  newretval=exact_diff( new, w1 );
  if ( (newretval<oldretval) || (newretval==0) )
    {
      oldretval=newretval;
      retval->word=malloc( strlen( new )+1 );
      strcpy( retval->word, new );
      if (!newretval) break;
    }

}

    }
  retval->val=1;
  return retval;
}

/* ************************************************************
int minimum( int *array, int elements )

Returns the index of the element with the smallest numerical
value  in the integer array array. elements is the number of
elements in the array.
************************************************************ */

int minimum( int *array, int elements )
{
  int min=0, i;

  for( i=0; i<elements; i++ )
    if ( array[i]<array[min] )
      min=i;

  return min;
}

/* ************************************************************

int main ( int argc, char **argv )

Arguments: w1 and w2, two strings. The difference of the strings
is computed as the number of operations necessary to convert w2
into w1. Possible operations are defined in the *_loop()
functions above.
```

```
This function may give non-optimal results because it doesn't
avoid local optima well enough. A full alpha/beta game algorithm
would be needed here to always get the correct result.

return: always 0. Error conditions are not checked or reported.
********************************************************** */

int main ( int argc, char **argv )
{
  char *word1, *word2, strl1, strl2;
  int diff, errors=0;
  int newdiff[5], minindex=0, bestdiff;

  /* Get the command line arguments */
  word1 = malloc( strlen( argv[1] )+1 );
  word2 = malloc( strlen( argv[2] )+1 );

  strcpy( word1, argv[1] );
  strcpy( word2, argv[2] );

  printf( "Original=%s Test=%s EXACT=%d\n", word1, word2, \
exact_diff( word1, word2 ) );

  /* Compute initial difference */
  bestdiff=diff=exact_diff( word1, word2 );

  /* Modify w2 until it matches w1 */
  while ( bestdiff )
    {
      printf( "diff=%d word2=%s\n", diff, word2 );

      rotate_right_loop( word1, word2, &retval );
      newdiff[0]=exact_diff( word1, retval.word );

      insert_loop( word1, word2, &retval );
      newdiff[1]=exact_diff( word1, retval.word  );

      remove_loop( word1, word2, &retval );
      newdiff[2]=exact_diff( word1, retval.word  );

      replace_loop( word1, word2, &retval );
      newdiff[3]=exact_diff( word1, retval.word  );

      gnr_loop( word1, word2, &retval );
      newdiff[4]=exact_diff( word1, retval.word  );

      minindex=minimum( newdiff, 5 );

      printf( "Newdiff R=%d I=%d D=%d X=%d G=%d min=%d\n", \
newdiff[0], newdiff[1], newdiff[2], newdiff[3], newdiff[4], \
minindex );
```

```
      switch ( minindex )
{
case 0:
  rotate_right_loop( word1, word2, &retval );
  bestdiff=exact_diff( word1, retval.word );
  break;
case 1:
  insert_loop( word1, word2, &retval );
  bestdiff=exact_diff( word1, retval.word );
  break;
case 2:
  remove_loop( word1, word2, &retval );
  bestdiff=exact_diff( word1, retval.word );
  break;
case 3:
  replace_loop( word1, word2, &retval );
  bestdiff=exact_diff( word1, retval.word );
  break;

case 4:
  gnr_loop( word1, word2, &retval );
  bestdiff=exact_diff( word1, retval.word );
  break;
}

      if (bestdiff<diff)     /* <= ??? */
{
  errors+=retval.val;
  diff=bestdiff;
  free( word2 );
  word2=malloc( strlen( retval.word )+1 );
  strcpy( word2, retval.word );
  if (diff==0)
    break;
}
    }

  printf( "RESULT=%d for word2=%s\n", errors, word2 );
  return 0;
}
```

4.5. Transklusion

An important requirement of Xanalogical systems is the ability to transclude, or virtually include, portions of one or more documents into another.[7]

Name: Transklusion	**Letzte Änderung:** 5. Januar 2001
Synonyme: Transparente Inklusion, Einbindung verteilter Datenpartikel	**Status:** Erfolgreich eingesetzt auf Dokumenten–Ebene, im Entwurfs–Stadium auf Partikel–Ebene
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Die Interdependenz unterrichtlicher Inhalte, Methoden und Medien erfordert nicht nur die isolierte Wiederverwendung dieser einzelnen Strukturelemente sondern auch Verfahren für die Implementierung und Koordination verteilter Verantwortungen im netzwerkbasierten Unterricht. Besondere Schwierigkeiten ergeben sich hieraus in den Bereichen Datenredundanz und Wartungsintensität sowie Schutz intellektuellen Eigentums. Hier bieten bisherige unterrichtliche Werkzeuge keine praktikablen Lösungsansätze.	
Kurzdarstellung der Lösung: Eine Lösung für dieses Problem kann durch Nutzung des Musters „Trennung von Inhalt und Methodik“ erreicht werden, wenn dies gestattet, Daten auf Dokument– und Partikelebene im Netzwerk für transparente Inklusion in Echtzeit bereitzustellen.	
Gegenanzeigen: –	
Besondere Hilfsmittel: SeL–Interpreter	
Referenzen: –	
Verwandte Muster: „Trennung von Inhalt und Methodik“	
Anmerkungen: –	

Diskussion

Unterricht soll in der Lage sein, sich im urheberrechtlichen Rahmen frei mit Inhalten zu versorgen. Dies gilt für unterschiedliche Medien ebenso wie für Inhalte unterschiedlicher Herkunft. Diese Anforderung wird von unterschiedlichen

[7] Pam, Andrew: [43]

Hypertextsystemen jedoch unterschiedlich unterstützt.

Frühe Hypertextsysteme haben mit der Verfügbarkeit und Verbreitung von Netzwerktechnologien sehr bald die Abgeschlossenheit isolierter Datenspeicher überwunden, und bereits in den 1960er Jahren wurden praktisch — im Palo Alto Research Center — und theoretisch — durch die Arbeit Theodor H. Nelsons — Konzepte für die Integration verteilter hypermedialer Datenbanken entworfen. Bei unserer alltäglichen Nutzung des WWW erscheint es heute als selbstverständlich, daß Hypertexte inhaltlich nicht auf Rechnersystemen oder auf die Angebote individueller Autoren abgeschlossen sind sondern über die Grenzen einzelner Rechner und Subnetze und über die Zuständigkeit einzelner Autoren hinaus frei verlinkt werden können und ebenso frei navigierbar sind. Für Unterricht und Lernen ergeben sich hieraus neue Möglichkeiten, da Informationsangebote nicht mehr auf die Verantwortung eines oder weniger Autoren begrenzt sind und Hypertexte *ohne Anfang und Ende* selbstbestimmt explorativ durchstöbert werden können.

Als Tim Berners–Lee mit seinem Vorschlag für ein verteiltes Hypertextsystem[8], im Jahr 1989 die Entwicklung des WWW auslöste, kam er damit Theodor H. Nelson zuvor, der bereits seit 30 Jahren an seiner eigenen Idee für ein weltweit vernetztes Hypertext–Sysetem, seinem ehrgeizigen Projekt *Xanadu*, gearbeitet hatte. Im Jahr 1991 war dieses sehr anspruchsvolle Projekt bei Version 87.1 angelangt, ist aber praktisch nie ernsthaft eingesetzt worden und hat sich nicht verbreiten können. Das populäre WWW ist im Vergleich zu *Xanadu* ein oberflächliches, pragmatisches und wenig ausgereiftes Konzept. Daher weist das WWW nicht die von Nelson angestrebten Qualittäten auf, was letztlich aber der Grund dafür ist, daß es im Gegensatz zu *Xanadu* tatsächlich irgendwann verfügbar war. Das Design eines Hypertext–Systems ist ein bösartiges Problem, und das vergleichsweise zentralisierte Entwicklungsmodell von *Xanadu* war diesem Problem nicht gewachsen.

Die technische Überlegenheit von *Xanadu* zeigt sich in Eigenschaften wie der Transklusion. In *Xanadu* haben Nutzer nicht nur in ihrer Rolle als Leser die

[8] Berners–Lee, Tim: [3]

Möglichkeit, verteilte Ressourcen zu rezipieren, sondern diese auch in ihrer Rolle als Autoren in Dokumente zu integrieren. Dabei werden Elemente entfernter Dokumente direkt und für den Leser transparent in das erstellte Dokument eingebunden. Dies erfolgt nicht über das Anlegen einer lokalen Kopie des Fragments sondern durch Auslesen der jeweiligen Originalquelle im Moment jedes Dokument–Abrufs.

Nelson hat vorhergesehen, daß in verteilten Hypertext–Systemen sowohl inhaltliche als auch rechtliche Probleme auftreten werden: Zitate werden bewußt oder unbewußt aus dem Zusammenhang gerissen oder Urheberrechte nicht eingehalten. In Online–Lernsystemen ist dies heute ein großes Problem: Nachdem WWW–Server für Lerninhalte mittlerweile sehr einfach und günstig bereitgestellt werden können, stellt sich die Entwicklung und Pflege der Inhalte als das eigentliche Problem dieser Projekte heraus. Nachdem mit großen Anstrengungen solche Lerninhalte gesammelt und aufbereitet wurden, erscheinen sie aber oftmals als zu wertvoll für eine Publikation in einem Medium, das auf dem Prinzip des freien Kopierens basiert. Die technischen Möglichkeiten, diesem Problem des WWW zu begegnen, sind gering und bestehen in aller Regel aus Zugangssperren in Form von Firewalls oder Paßwortschutz, was wiederum den Wartungsaufwand der Systeme erhöht und Lernmittel als wertvolles und zu behütendes Material in hohem Maß an seine Besitzer bindet. Die Entwicklung interinstitutioneller verteilter Lernmittel–Datenbanken wird dadurch nachhaltig erschwert.

Beispiel

Das hier dargestellte Beispiel zeigt die Transklusions–Implementierung von SeSAMe. Diese gestattet zur Zeit die Transklusion auf Dokumenten–Ebene. Die Implementierung von Transklusion auf Pratikel–Ebene ist geplant. Mit dem folgenden Kommando kann der Inhalt eines auf einem entfernten Server bereitgestellten Dokuments geladen, und zur weiteren Verarbeitung an eine SeL–Variable übergeben werden.

```
=transclusion $_bstrip($_include("http://www.uni-kassel.de/"));
```

Auf diese Weise kann beispielsweise der Fließtext eines entfernten Dokuments komplett in ein lokales Dokument transkludiert werden. Die geplante Transklusion auf Partikel–Ebene[9] wird einzelne, in logisch in SeL ausgezeichnete Dokument–Elemente transkludieren und mit lokalen Masterfiles verarbeiten können. Mit einem Aufruf nach dem Muster

```
=a <SQD SOURCE="http://remote.addr/foo.ghk">a</SQD>;
```

und einer Wertzuweisung in der referenzierten Datei auf dem entfernten Server nach dem Muster

```
=a <SQD ACCESS="all">b</SQD>;
```

kann die Zuweisung in der entfernten Datei für das lokale Angebot genutzt werden. Die zusätzlichen Attribute `SOURCE` und`ACCESS` in den SeL–Wertzuweisungen werden für die Adressierung entfernter Dateien sowie für die Definition von Zugriffsrechten benötigt. Durch eine Implementierung, die nur `SOURCE` oder `ACCESS` in einer Zuweisung erlaubt, kann die „Verkettung“ von Transklusionen unterbunden und die ausschließliche Transklusion originaler Resourcen erzwungen werden. Abbildung 4.1 illustriert das Prinzip der mit den obigen Beispiel–Kommandos diskutierten Transklusion eines Dokument–Partikels.

[9] Fischer, Thomas: [15]

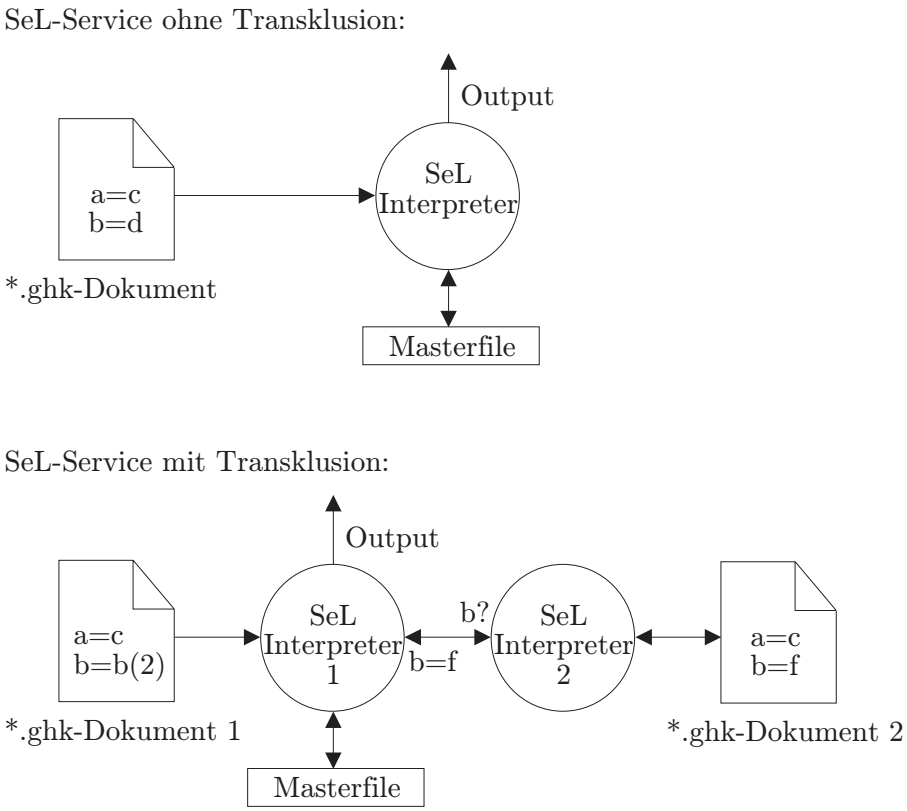


Abbildung 4.1.: SeL WWW—Service ohne und mit Transklusion

Aus unterrichtlicher Sicht ergeben sich aus diesem Prinzip besonders interessante Anwendungsmöglichkeiten für *Online Learning* in höheren Bildungsinstitutionen. *Virtuelle Universitäten* brauchen sich prinzipiell nicht notwendigerweise den an ihren „realen“ Standorten vorhandenen Resourcen unterzuordnen. So ist es durchaus denkbar, das Bildungs–Angebot einer virtuellen Universität über das an der jeweiligen realen Universität hinausgehen zu lassen, indem die dafür erforderlichen Resourcen von anderen Universitäten (etwa im Tausch gegen eigene Angebote) erworben werden. Neben der hierfür erforderlichen Klärung rechtlicher und administrativer Fragen, fehlt jedoch bislang die technische Basis für ein solches Vorgehen.

Mit der dargestellten Transklusion auf Partikel–Ebene wird es folglich technisch möglich sein, Lernmittel einer entfernten Institution mit lokalen Methoden und in einem lokalen Erscheinungsbild anzubieten. Ein Fachbereich einer anderen virtuellen Universität kann virtuell mitbenutzt und entsprechend den eigenen

gestalterischen und didaktisch/methodischen Vorstellungen in das eigene virtuelle Lehrangebot transkludiert werden.

4.6. Automatische Gruppierung für Partner– und Gruppenarbeit

Name: Automatische Gruppierung für Partner– und Gruppenarbeit	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich getestet
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Die für die Durchführung von Partner– und Gruppenunterricht erforderliche Zusammenstellung von Lernergruppen ist eine arbeitsintensive Routine–Aufgabe Lehrender. Diese wird um so schwieriger, je mehr die jeweilige gewünschte Gruppierung von didaktischen Kriterien abhängt, je größer die Anzahl der zu gruppierenden Lerner ist und je mehr verschiedene Gruppierungen pro Unterrichtseinheit erforderlich sind. Der Versuch, diese organisatorische Aufgabe schnell und einfach zu lösen, führt in der Lehrpraxis oft dazu, daß Lerner den Auftrag erhalten, sich selbständig zu gruppieren, was Lehrenden jedoch weitgehend den Einfluß auf die Einhaltung didaktisch intendierter Regeln für die Gruppenfindung entzieht. Dies erschwert die Bildung wünschenswerter Konstellationen oder schließt sie sogar aus. Somit behindert diese organisatorische Routine–Aufgabe in der Praxis immer wieder die Umsetzung didaktischer Zielsetzungen.	
Kurzdarstellung der Lösung: Dieses Muster bietet eine einfache und schnelle Lösung für die Herstellung von Lernergruppen beliebiger Stärke aus beliebigen Lernerzahlen, in Abhängigkeit frei definierbarer didaktischer Kriterien. Dabei können wahlweise homogen oder inhomogen aufgebaute Gruppen in zufälligen individuellen Zusammensetzungen erzeugt werden.	
Gegenanzeigen: –	
Besondere Hilfsmittel: Perl–Interpreter	
Referenzen: –	
Verwandte Muster: –	
Anmerkungen: –	

Diskussion

Die Einteilung von Lernergruppen ist eine unterrichtliche Routineaufgabe. Dabei sollen Gruppierungen für Partner– oder Gruppenarbeiten, ggf. in Abhängigkeit von definierten Kriterien (Lernereigenschaften etc.) hergestellt werden. Dies wird insbesondere in solchen Unterrichtssituationen zum Problem, in denen häufig unterschiedliche Gruppierungen benötigt werden oder in denen große Lernergruppen involviert sind. Die übliche Praxis, die jeweiligen Lerner

selbst mit der Gruppierung zu beauftragen, erschwert nicht nur die Erzeugung wünschenswerter Kombinationen. Im computervermittelten (Fern–)Unterricht erfordert dies einen aufwendigen elektronisch vermittelten Verhandlungs– und Koordinationsprozeß, der selbst die Dauer langsamer manueller Gruppen–Komposition durch den Lehrer bei weitem übertrifft. Ein Werkzeug zur raschen Erzeugung zufällig durchmischter Lernergruppen nach didaktischen Kriterien schafft hier nicht nur Abhilfe, sondern gestattet darüber hinaus die Durchführung bislang kaum realisierbarer unterrichtlicher Modelle, die in sehr raschen Abständen viele neue Gruppierungen innerhalb eines Kurses erfordern.

Beispiele für Gruppierungskriterien können erfaßte Leistungen (Noten), individuell gewählte Vertiefungs–Fächer, die Zugehörigkeit zu Lehrinstitutionen (etwa bei der Koordination multikultureller Kooperationen) sein. Das hier aufgeführte Programm stellt Lernergruppen nach derartigen Kriterien entweder inhomogen oder homogen zusammen. So ist es beispielsweise einerseits möglich, Gruppierungen unterschiedlich leistungsstarker Lerner oder möglichst interdisziplinär herzustellen, andererseits können ebenso einfach Lerner gleicher Leistungsstärke oder gleicher Fachgebiete in Gruppen zusammengeführt werden.

Beispiel

Dieses Beispiel gestattet frei wählbare Zeichenketten für die Identifikation von Lernern (Name, Matrikelnummer etc.) sowie für die Gruppierungs–Kriterien (gegenwärtiger Leistungsstand in Form einer Schulnote, individueller Studienschwerpunkt etc.). Es nutzt die objektorientierten Eigenschaften von Perl und kann nach dem Einbinden der unten aufgeführten Bibliothek mit `require 'combine.pl';` mit einem einzigen Aufruf aktiviert werden. Dafür sind drei Elemente erforderlich: Ein Bezeichner für die Art der Gruppierung (`h` für homogen und `i` für inhomogen) und ein numerischer Wert für die gewünschte Gruppenstärke. Ein aufrufendes Programm könnte folgendermaßen aussehen:

```
#!/usr/bin/perl

require 'combine.pl';

open(READ,"<./course.txt");
```

```
while(<READ>){
    chop($_);
    @HELP=split(/,/,$_,,2);
    $GIVEN{$HELP[1]}=$HELP[0];
}
close(READ);

# at this point %GIVEN is expected to contain
# identity/attribute-pairs. $ARGV[0] is supposed
# to be "i" or "h" for either inhomogenous or
# homogenous groups. $ARGV[1] is the group size.


# grouping command:
my %GROUPS = Group->create($ARGV[0],$ARGV[1],%GIVEN);

# print result:
while (($student,$group)=each(%GROUPS)){
print "$student->Group $group(".$GIVEN{$student}.")\n";
}

exit;
```

Die Bibliothek, die durch `require 'combine.pl';` eingebunden werden muß und den eigentlichen Algorithmus für die Gruppierung enthält, ist in der Datei '`combine.pl`' enthalten. Sie hat den folgenden Inhalt:

```
# combine.pl:   Random Learner Grouping by Attribute
# (c)2000 by himself@tfischer.de
#
#####################################################

package Group;

my($student); my($spec); my($flag);
my(@CHECK); my($stdts); my($attnr);
my($specnh); my(@ARRAY); my(@speccount);
my($max); my(@THIS); my(@THAT);
my($key); my(%MORE); my(@NEWORDER);
my($i); my($x); my($j);
my($GNTR); my(@HELPARRAY); my(@OUT);
my($grpcntr); my($group); my(@HELP);
my(%GIVEN); my($optmax); my($specno);

sub create{
my ($SELF,$TASK, $SIZE, %GIVEN)=@_;

my $stdts=0; my $attnr=0;

while (($student,$spec)=each(%GIVEN)){
```

```
    if( !$student || !$spec ){
die print "Sorry: Database damaged.\n";
    }
    $flag=0;
    for(@CHECK){
if ($_ eq $spec ){
    $flag=1;
}
    }
    if ($flag == 0){
        push(@CHECK,$spec);
$attnr++;
    }
    $stdts++;
}

if ($stdts == $attnr){
    die"Number of students equals number of attribute values.\
No grouping necessary/possible.\n";
    exit;
    }

# found $stdts students with $attnr different attributes.

while (($student,$spec)=each(%GIVEN)){
    $specnh=0;
    for(@CHECK){
if ($_ eq $spec ){
    $specno=$specnh;
}
$specnh++;
    }
    @ARRAY[($specno)+($attnr*$speccount[$specno]++)]=$student;
}

# students are now distributed in a 2d array with $attrno columns

$max=$speccount[0];
for ($i=1;$i<=$attnr;$i++){
    if($speccount[$i] > $max){
$max=$speccount[$i];
    }
}

#$square=$max*$attnr;

# maximum attr size is $max.
# there are $square (empty and/or used) fields

#scramble each column content for random compositions:
```

```
for ($i=0;$i<=$attnr-1;$i++){
    @THIS=();
    @THAT=();
    for($j=0;$j<=$speccount[$i]-1;$j++){
$THIS[$j]=$ARRAY[$j*$attnr+$i];
}
    push(@THAT, splice(@THIS, rand @THIS, 1))while@THIS;
    for($j=0;$j<=$speccount[$i]-1;$j++){
$ARRAY[$j*$attnr+$i]=$THAT[$j];
}
    }
# column contents are scrambled

# find modulo differences and new order (optimizing table)
for ($i=0;$i<=$attnr-1;$i++){
    $MORE{$i}=$speccount[$i]%$SIZE;
    }

foreach $key (sort ascsrt (keys(%MORE))){
    @NEWORDER=(@NEWORDER,$key);
    }
# found modulo differences and new order

# move low modulos up
$i=0;
for (@NEWORDER){
    for($j=0;$j<=$max-1;$j++){
$OUT[$j*$attnr+$i]=$ARRAY[$j*$attnr+$_];
}
    $i++;
}
@ARRAY=@OUT;
# low modulos moved up

# create optimized table
$grpcntr=0;
if(($stdts/$attnr)>int(stdts/$attnr)){
    $optmax=int($stdts/$attnr)+1;
    }else{
    $optmax=$stdts/$attnr;
    }
# optimized table created

# create groups:
if ($TASK eq "h"){                      # homogenous ?
    for($i=0;$i<$attnr;$i++){
for($j=0;$j<$max;$j++){
    $x=$j*$attnr+$i;
    if($ARRAY[$x] ne ""){
        @HELPARRAY=(@HELPARRAY,$ARRAY[$x]);
     }
```

```
    }
        }
    }else{ # inhomogenous ?
    for($i=0;$i<$max;$i++){
for($j=0;$j<$attnr;$j++){
    $x=$j*$max+$i;
if($ARRAY[$x] ne ""){
    @HELPARRAY=(@HELPARRAY,$ARRAY[$x]);
         }
    }
        }

    }
# groups created

# print groups:
$GNTR=1;$i=0;
for(@HELPARRAY){
    if(($i%$SIZE==0)&&(($stdts-$i)>$SIZE/2)){$GNTR++;}
    $GROUPS{$HELPARRAY[$i]}=$GNTR;
    $i++;
    }
return %GROUPS;
} # subroutine end

# sorting routine:
sub ascsrt{
    $MORE{$a}<=>$MORE{$b}||$speccount[$b]<=>$speccount[$a];
    }
```

4.7. Echtzeit–Visualisierung von Lerner–Interaktionen

Name: Echtzeit–Visualisierung von Lerner–Interaktionen	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich experimentell eingesetzt eingesetzt
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer, Online–Lehrer
Problem: Wo netzerk– und computerbasierte Interaktion an die Stelle von Seminar–, Klassenraum– oder Studio–Interaktion treten, droht ein Verlust von Informationen über Lerneraktivität. Diese ist jedoch oft sehr wichtig und wetvoll für didaktische Analysen und Reflexionen, für die Leistungsbewertung usw. Die normale Beobachtung einer Gruppe im Klassenraum gestattet Lehrenden die Wahrnehmung der gesamten Gruppensituation sowie das Verfolgen ggf. vieler paralleler Interaktions– und Lernprozesse. Online sind Beobachtungen von ähnlicher Qualität sehr schwierig. Das maschinelle Anlegen von Interaktions–Protokollen in Datenbanken ist keine zufriedenstellende Lösung, da *sehr dichte Kommunikation*, *sehr schnelle Interaktion* und ähnliche Vorkommnisse im Klassenraum natürlich wahrgenommen werden. Es ist im Klassenzimmer kaum möglich, ein beispielsweise irgendwie besonderes, letztlich aber *offensichtliches* Verhalten zu übersehen. Bei der Auswertung von Protokolldaten (etwa durch textuelle SQL–Abfragen) findet man hingegen immer nur solche Informationen, die man auch sucht und es ist sehr wahrscheinlich, beispielsweise irgendwie besonderes Verhalten zu übersehen, wenn man eine Datenbank nicht explizit danach durchsucht.	
Kurzdarstellung der Lösung: Datenvisualisierungen gestatten Darstellungen von Datenbank–Inhalten, deren visuelle Wahrnehmung sehr viel mehr Aufschluß über die Aktivität von Lernergruppen als Auswertungen beispielsweise alphanumerischer Protokolle. Wie auch beim Autorensystemdesign ergibt sich aus der Definition graphischer Oberflächen eine mangelhafte Flexibilität und Erweiterbarkeit. Benötigt wird daher eine Geometrie zur Visualisierung von Lerner–Interaktionen, die frei erweiterbar auch die Visualisierung nicht vorhergesehener Unterrichtsmethoden gestattet.	
Gegenanzeigen: –	
Besondere Hilfsmittel: Auf der Server–Seite CGI–fähiges 3D–Modelling Paket wie z.B. 3D Studio Max oder alternativ ggf. Code–Bibliotheken zur Erzeugung von VRML. Auf der Client–Seite VRML–Browser (PlugIn) wie z.B. Cosmo Player.	
Referenzen: Die Darstellung dieses Musters beruht teilweise auf Fischer, Thomas, Christiane Herr und Cristiano Ceccato: Towards Real Time Interaction Visualization in NED [17]	

Verwandte Muster: –
Anmerkungen: Im Rahmen der Entwicklung dieses Musters weisen gegenwärtige VRML–Browser auf gängigen Rechnersystemen Probleme mit der Darstellung umfangreicherer Interaktions–Datenbanken auf. Diese quantitativen Beschränkungen dürften in Zukunft mit steigender Rechenleistung gelöst werden.

Diskussion

Der traditionelle Klassenunterricht ist ein über Jahrhunderte gewachsenes Konzept. Er hat sich bis heute durch seine Stärken bewährt, gegen andere Lehrformen durchgesetzt und damit das typische Bild des modernen Schulunterrichts geprägt. Einer seiner Vorteile ist die unmittelbare physische Präsenz, mit der sich Lerner und Lehrer im Unterrichts–Szenario begegnen. Sie erlaubt die direkte Beobachtung und Analyse von Interaktion. Dieses Instrument wird zunehmend verdrängt, wo Unterricht in Software und Netzwerken und nicht mehr im Klassenzimmer oder Seminarraum stattfindet. Eine Ausnahme bilden hier jene computervermittelten Unterrichtssituationen, mit denen implizite künstliche Darstellungen von Unterrichtssituationen, etwa in Form von Avatars in virtuellen Räumen, einhergehen.

Dort, wo Interaktion nicht implizit visualisiert wird, drohen wichtige didaktische Informationen verloren zu gehen, die nicht nur das Potential haben, Unterricht auf unterschiedliche Weisen zu bereichern, sondern auch eine zentrale Grundlage für dessen Beurteilung darstellen. Hieraus resultiert ein dringender Bedarf an Methoden, die die Möglichkeiten der Beobachtung traditionellen Unterrichts auf computervermittelte Lehre anwendbar machen und auch hier jederzeit direkte Situations–Analysen gestatten. Dies ist möglich, wenn Software–Lernumgebungen automatisch Protokolle ihrer Nutzung anlegen, die von Lehrenden einfach analysiert werden können. Zu diesem Zweck bieten sich abstrakte, in Echtzeit generierte graphische Datenrepräsentationen an. Im Rahmen der Entwicklung möglichst generischer Lösungen für Autorensysteme sollten die abstrakten Geometrien, durch die Interaktion repräsentiert werden soll, ihrerseits eine möglichst generischen Charakter haben, d.h. sie sollten in der Lage sein, nicht nur einzelne, sondern möglichst viele (im Idealfall alle) didak-

tischen Interaktionen darstellen zu können.

Das Protokollieren von Lerner–Interaktion bietet eine Möglichkeit, jederzeit zu untersuchen, *was* Lerner *wann wie* tun und dabei individuelle Leistungen zu vergleichen. Es ist relativ einfach, Software so zu programmieren, daß sie Protokolle der eigenen Nutzung, sogenannte *Logfiles*, anlegt. Tatschlich ist dies etwa bei WWW–Servern eine gängige Praxis. Hier werden aus Sicherheitsgründen sowie zu statistischen Zwecken automatisch sämtliche Anfragen nach Dokumenten und auftretende Fehler protokolliert. Jedoch ist die unmittelbare Analyse solcher maschinengenerierten, oftmals sehr umfangreichen und kryptischen Protokolle für Menschen eine sehr anstrengende und zeitaufwendige Aufgabe. Dies gilt insbesondere Unterrichtssituationen, in denen fortlaufende Beobachtungen von Lernergruppen in kurzen Intervallen erfolgen sollen, um auf Beobachtungen ggf. möglichst schnell mit der Anpassung unterrichtlicher Strategien reagieren zu können. Die Identifikation von Verhaltensmustern von Individuen oder Gruppen, komplexen Interaktionsmustern oder Korrelationen können praktisch nicht zuverlässig intuitiv durch menschliches Lesen erfaßt werden.

Den zugriffs– und fehlerorientierten Logfiles, die übliche WWW–Server anlegen, sind Kategorien wie Lerner oder Lernergruppen und deren Aktivität fremd. Sie können Lehrende nur sehr unzureichend mit den Informationen versorgen, an denen sie tatsächlich interessiert sind. In einer CGI–basierten Anwendung können einzelne Programme Informationen über ihre Nutzungsweise und –zeiten in Datenbanken protokollieren. Wichtig ist hier allerdings, daß diese Datenspezifikation für diesen Zweck flexibel und generisch genug ist, alle (möglicherweise zunächst unvorhergesehenen) Interaktionsformen transportieren zu können.

Ein Vorteil der Interaktions–Protokollierung und deren visueller Auswertung gegenüber der Beobachtung des traditionellen Klassenunterrichts besteht in der automatischen Speicherung, die auch zeitversetzt erfolgen kann (dies ist im Klassenzimmer bzw. im Seminarraum nur durch technische Maßnahmen wie Videoaufnahmen möglich). Andererseits kann Software nur diejenigen Aktionen protokollieren die sie auch erwartet, weil Lehrer bzw. Programmierer sie vorhergesehen haben.

Ein potentielles Problem besteht hingegen in der möglichen Protokollierung und Auswertung von privaten Informationen, etwa persönlichem E-Mail–Verkehr oder Chat–Beiträgen. In eigenen Applikationen begnüge ich mich aus diesem Grund mit der Protokollierung von Ereignissen, nicht aber konkreten Inhalten. Dies wiederum wird von Lernenden mitunter erkannt, worauf dann mit größeren Mengen sinnloser Lerner–Kommunikation reagiert wird, die in Protokoll–Auswertungen als starkes unterrichtliches Engagement erscheinen sollen. Automatische Unterrichtsbeobachtung allein vermag Aufmerksamkeit und gesundes Mißtrauen nicht ganz zu ersetzen.

Die in Abb. 4.4 und Abb. 4.5 dargestellte Visualisierung wurde nicht in VRML sondern mit der 3D–Modellierungs–Software 3D Studio MAX hergestellt. Grund dafür ist der Umfang des entsprechenden VRML–Quellcodes, der zur Zeit nicht sinnvoll von aktuellen VRML–Browsern dargestellt werden kann. Eine Lösung dieses Probems ist wohl nur eine Frage der Zeit. Bis dahin besteht die (von mir nicht näher untersuchte) technische Möglichkeit, 3D Studio MAX via CGI aufzurufen und zur serverseitigen Herstellung von jpeg–Grafiken des 3D–Modells zu nutzen. Damit würde die Clientseite sowohl in bezug auf die Rechenleistung entlastet als auch das Erfordernis eines 3D–Browsers (bzw. PlugIns) entfallen. Andererseits wären die erzeugten Visualisierungen nicht interaktiv dreidimensional navigierbar.

Eine Lösung, die computervermittelten Unterricht beobachtbar macht, erfordert drei Teillösungen: Zunächst müssen die Interaktionsformen, die das Interface einer Lernsoftware bereitstellt, identifiziert werden und es muß sichergestellt werden, daß bei Erfolgen einer solchen Interaktion ein Protokoll-Eintrag erfolgt. Ein solches Protokoll kann durch ein Datenbank–Management–System verwaltet werden oder als einfache Textdatei angelegt werden (eine sogenannte *Flat File Datenbank*). Zweitens, dies ist eine Voraussetzung für eine brauchbare Datenbank, ist eine eindeutige, erweiterbare und leicht maschinenlesbare Spezifikation für die Protokolle erforderlich. Eindeutig heißt, daß Daten weder mehrdeutig noch redundant codiert sind. Erweiterbar heißt, daß sich eine erfolgte Datenspezifikation flexibel verhält, wenn im Verlauf der Nutzung neue Interaktionsoptionen und somit neue Datenelemente hinzugefügt werden

sollen. Maschinenlesbarkeit sollte gesichert sein, wenn Daten eindeutig ausgezeichnet werden, jedoch erfordert die maschinelle Zerlegung (das sog. *Parsing*) verschiedener Formate unterschiedlich viel Aufwand. Dies wird wichtig, wenn umfangreiche Protokolle ausgewertet werden müssen. Drittens ist eine Geometrie erforderlich, die in der Lage ist, Daten vergleichbar zu repräsentieren, sämtliche protokollierte Daten visuell umzusetzen und sich ebenfalls kooperativ gegenüber Spezifikations–Erweiterungen verhält.

Die Analyse von Unterricht erfolgt im Klassenzimmer im wesentlichen in Form von unmittelbarer, visueller und akustischer Beobachtung und Beurteilung. Der Charakter der auf diese Weise gesammelten Unterrichtsinformation entspricht weitgehend dem normaler menschlicher Umweltwahrnehmung. Dies ist bei maschinell angefertigten Protokollen sicherlich weniger der Fall. Dementsprechend schwer fällt es, derartigen Protokollen Informationen und Effekte zu entnehmen, die im Klassenzimmer unmittelbar erkennbar sind. Die didaktische Weisheit, daß ein Bild mehr sagt als tausend Worte, trifft auch hier zu.

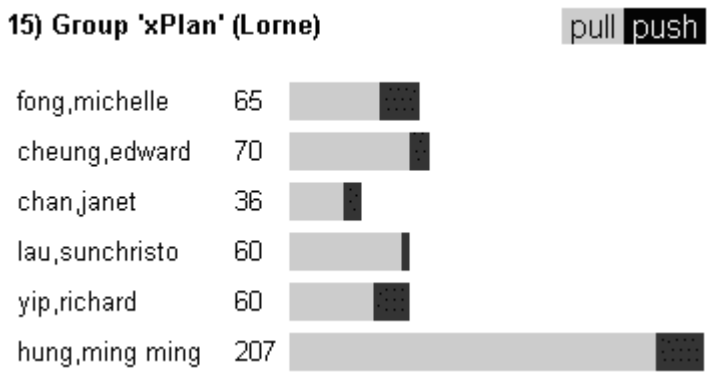


Abbildung 4.2.: Balkendiagramm–Visualisierung von Lernerinteraktion

Abb 4.2 zeigt einen Ausschnitt einer umfangreichen Visualisierung von Interaktionsdaten einer Lernsoftware (der „Virtual Design Company“ Website). Genauer handelt es sich um die Darstellung der individuell gemessenen Aktivität der Mitglieder einer Lerngruppe. Das Diagramm, aus dem dieser Ausschnitt stammt, ist ein per CGI–Script aus einer Protokoll–Datenbank erzeugtes HTML–Dokument, das schließlich zur Benotung des Kurses herangezogen

wurde. So wie diese Gruppe weisen auch alle anderen Lernergruppen dieses Kurses einen Effekt auf, der ohne Visualisierung wahrscheinlich übersehen worden wäre: In jeder Gruppe überragt die Leistung eines Lerners deutlich die Leistungen der jeweiligen Gruppen–Mitglieder. Dabei sind diese besonders aktiven Studenten nicht zwangsläufig die, die den Lehrenden als besonders leistungsstark bekannt waren. Die Gruppierung wurde weder von der Lernsoftware, noch von den Lehrenden vorgegeben. Vielmehr hat die Software eine freie, eigenständige Gruppierung der Lernenden ermöglicht. Interessanterweise handelt es sich bei fast allen der herausragenden Studenten um diejenige Person, die die jeweilige Gruppe ursprünglich gegründet hatte. Es liegt die Vermutung nahe, daß mit einer gewissen Wahrscheinlichkeit aus der Gründung einer Arbeitsgruppe ein gesteigertes Gruppenbewußtsein resultiert, das zu einer deutlich erhöhten Aktivität führt. Selbstverständlich wäre es ebenso möglich, daß die Studenten unterschiedlich gerne mit dem Computer arbeiten und diejenigen, die dies besonders gerne tun, sich einerseits als besonders aktiv erweisen und andererseits besonders schnell verschiedene Hürden überwinden und früher als andere Lernergruppen gründen. Es ist kaum möglich, Effekten in Lernsituationen eindeutige Regeln und Variablen zuzuordnen und so sollte man vorsichtig sein, auf der Basis solcher Beobachtungen Empfehlungen oder Handlungsweisen für andere, irgendwie vergleichbare Situationen auszusprechen. Es ist jedoch durchaus denkbar und begrüßenswert, wenn individuelle Lehrer aus eigenen derartigen Beobachtungen ihrer Lerner(gruppen) Schlüsse ziehen und diese dann in die Gestaltung ihres Unterrichts einfließen. Eine mögliche Reaktion auf den beschriebenen Effekt könnte sein, daß man versucht, auf die eine oder andere Weise besonders förderungswürdig erscheinenden Lernern die Verantwortung für zukünftige Gruppenbildungen zu überlassen.

Letztlich setzt eine derartige Umsetzung von Erkenntnissen aber zunächst deren Wahrnehmung voraus. Das aufgezeigte Beispiel veranschaulicht, daß visuelle Datenrepräsentationen das Potential haben, einen intuitiveren Zugang zu unerwarteten Effekten eröffnen zu können als rein textuelle Protokolle.

Beispiel

Die hier präsentierte Beispiel–Problemlösung erscheint zur Zeit aufgrund technischer Konstriktionen noch nicht ohne weiteres praktisch umsetzbar. Da es sich hier um rein quantitative Konstriktionen in der Datenverarbeitung handelt, kann man jedoch davon ausgehen, daß es nur eine Frage der Zeit ist, bis das folgende Szenario praktikabel wird:

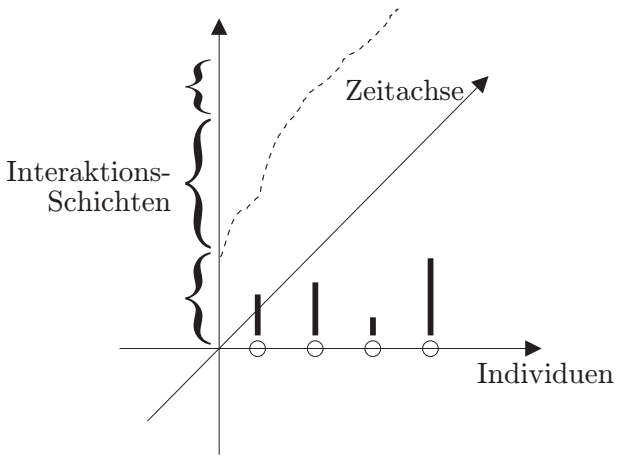


Abbildung 4.3.: Erweiterbare Geometrie zur Visualisierung von Lernerinteraktion

Ein Eintrag in einem Flat–File Log–Protokoll könnte z.B. in der Form `1072869,34,13,*!].` erfolgen. In dieser Komma–separierten Liste enthält das erste Element die Sekunde des laufenden Kurses, das zweite eine Lerner–ID–Nummer, das dritte einen Code für eine Aktivität wie z.B. das Verschicken einer E-Mail an eine Lernergruppe und das vierte Feld, das für optionale Attribute des dritten Feldes reserviert ist, enthält eine Liste der gegenwärtigen Mitglieder der Gruppe, an die die E-Mail verschickt wird. Es handelt sich in diesem Fall um eine ASCII–Interpretation einer binär codierten Submenge aller Lerner des Kurses[10].

[10] Bei dieser Codierung wird eine binäre Zahl generiert, die eine Stelle für jeden Lerner des Kurses enthält. Lerner, die in der betreffenden Submenge enthalten sind, werden mit einer 1 repräsentiert, alle anderen mit 0. Das entstehende Bitmuster kann dann (z.B. mit der PERL–Funktion *pack*) als Reihe von ASCII–Zeichen interpretiert werden. Der Vorteil dieses Verfahrens besteht

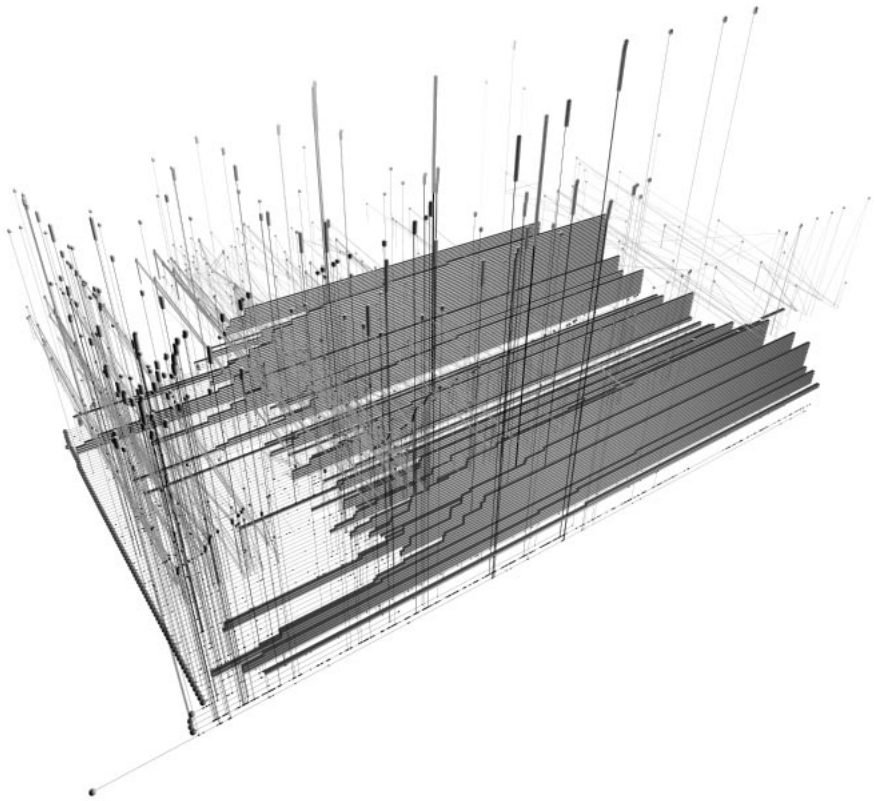


Abbildung 4.4.: Visualisierung von Lernerinteraktion, Perspektive

Zum Zweck der Unterrichtsbeobachtung erfolgt eine Visualisierung einer Interaktions–Datenbank aus Gründen der Übersichtlichkeit in der Mehrheit der Fälle nicht komplett, sondern in Teilen. Aspekte der Auswahl sind analog zur obigen Protokoll–Spezifikation *Aktivitäten*, *Zeiten* und *Personen*; oder aus didaktischer Sicht: *Welche Handlungen erfolgen?*, *Wann erfolgten Handlungen?* und *Wer handelt?* Einer Teilvisualisierung der Datenbank muß demnach eine Auswahl vorausgehen, die im einfachsten Fall manuell vom analysierenden Lehrer erfolgt, der z.B. in einem HTML–Formular ein Zeitfenster definiert sowie Interaktionsformen und Lerner bestimmt, die in der Visualisierung dargestellt werden sollen.

---

in den sehr kurzen resultierenden Zeichenketten, die eindeutig decodierbar sind, sowohl flat file als auch mittels Datenbank–Managementsystemen leicht verwaltet werden können und mit binären Operationen sehr leicht zu durchsuchen sind

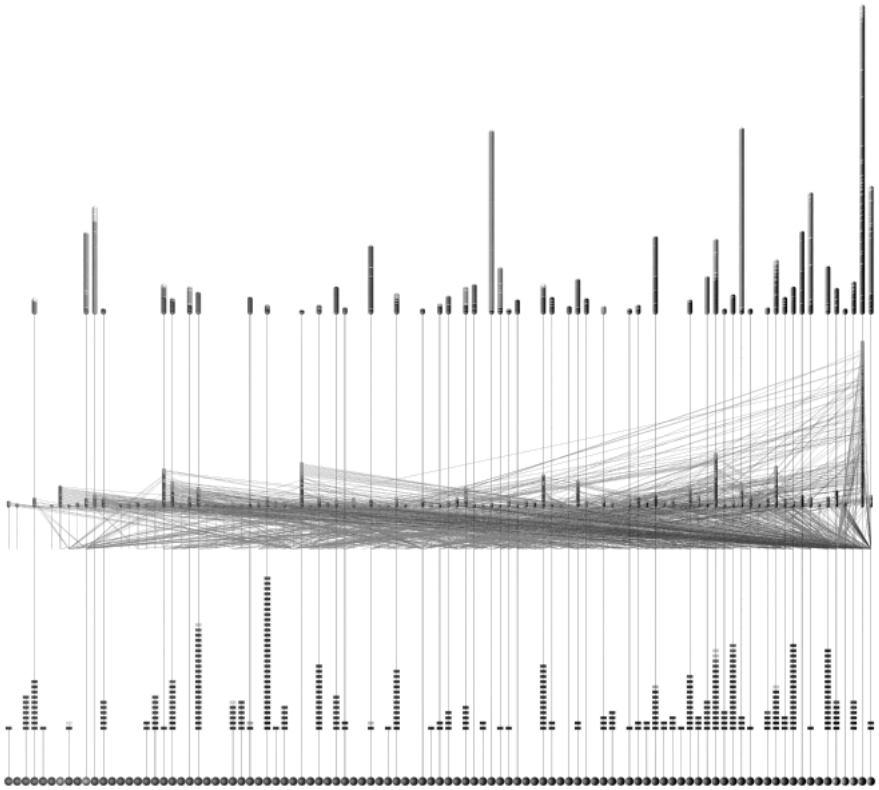


Abbildung 4.5.: Visualisierung von Lernerinteraktion – Frontalansicht

Die in Abb. 4.4 dargestellte Gesamtvisualisierung einer Kurs–Interaktion[11] veranschaulicht, wie die drei räumlichen Dimensionen genutzt werden können, um die drei genannten Beobachtungsaspekte zu erfassen: Entlang der x–Achse erstecken sich die Profile der am Kurs beteiligten Lerner und Lehrer und wachsen entlang der Zeitachse (y–Achse). Entlang der z–Achse werden die verschiedenen Interaktionsformen dargestellt.

Details werden in dieser Geometrie durch das Ausblenden derjenigen Elemente betont, die gerade nicht von Interesse sind. Andere Problemlösungen im Bereich der Datenvisualisierung[12] bedienen sich zu diesem Zweck nicht–linearer Darstellungen wie etwa sogenannter *fisheye views*, in denen die Größen dargestellter Elemente proportional zu deren jeweiliger Relevanz skaliert werden.

[11] Design Studies 4, 1999 an der School of Design, Hong Kong Polytechnic University

[12] siehe Card, Stuart K. et al. [6]

Nicht–lineare Darstellungen nehmen dem Betrachter die Möglichkeit direkter quantitativer Vergleiche, die bei der Leistungsbewertung eine zentrale Rolle spielen. Hier kann bereits die perspektivische Darstellung von 3D–Geometrien verzerrend wirken. Aus diesem Grund erfolgen die Repräsentationen von Aktivitäten bei jedem erneuten Auftreten entlang der Zeitachse mit einer Verschiebung in z–Richtung. Als Resultat erscheinen diese Aktivitäten (siehe Abb. 4.5) in der Frontal–Ansicht als linear vergleichbare Balkendiagramme.

4.8. Anzeige gleichzeitiger Nutzung von Online–Ressourcen

Name: Anzeige gleichzeitiger Nutzung von Online–Ressourcen	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich eingesetzt
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer, Online–Lehrer
Problem: Computer und die Netzwerke, über die sie miteinander in Kontakt stehen, stellen wahrscheinlich das leistungsstärkste elektronische Medium zur Massen– und Unterrichtskommunikation dar, das uns heute zur Verfügung steht. Selbstverständliche gestalterische Entscheidungen, die dem *Massenmedium WWW* zugrundeliegen, widersprechen jedoch naheliegenden Anforderungen an das *Unterrichtsmedium WWW*: Daß WWW–Server diskrete parallele Dialoge mit einzelnen Nutzern führen, ohne dabei Informationsfluß zwischen den Nutzern zu gestatten, liegt sicherlich im Interesse des Schutzes persönlicher Informationen. Unterricht zielt jedoch häufig neben der Kommunikation zwischen Lerner und Material auf auch auf den Austausch zwischen Lernern *über* das Material. Es gibt bislang jedoch keine Möglichkeit, Lernern, die zeitgleich dieselben Ressourcen nutzen, ihr gemeinsames Interesse maschinell zu signalisieren und gezielt Känale zum Austausch über diese Ressourcen zu öffenen.	
Kurzdarstellung der Lösung: Wenn Dokumente nicht statisch aus Dateien serviert werden sondern unter Zuhilfenahme von Scriptsprachen wie Perl oder SeL dynamisch (etwa aus Datenbanken) erzeugt oder ergänzt werden, ist es möglich, Hinweise über die gleichzeitige Nutzung in das jeweilige Dokument selbst einzubauen, die persönliche Informationen über die jeweiligen Lerner sowie Online–Verknüpfungen zu Kommunikations–Werkzeugen wie Chatrooms, Whiteboards oder Videokonferenz–Systeme enthalten können.	
Gegenanzeigen: –	
Besondere Hilfsmittel: WWW–Server mit SSI– bzw. CGI–Schnittstelle und eine Scriptsprache wie Perl oder SeL, ggf. Datanbank–Management–System zur Verwaltung von Anwesenheitsdaten.	
Referenzen: –	
Verwandte Muster: Dies ist ein Muster, das einer Form der Online–Gruppenbeobachtung aus Lernerprespektive dient. Für die Beobachtung einer kompletten Lernergruppe aus Lehrerperspektive empfiehlt sich das Muster „Echtzeit–Visualisierung von Lerner–Interakton“ (4.7). Die Identifikation und Zuordnung individueller Lerner–Daten kann mit dem Muster „Dynamische Erteilung von Zugriffsrechten“ (4.1) erfolgen.	
Anmerkungen: –	

Diskussion

Nutzerinnen und Nutzern WWW–gestützten Lernmittel–Datenbanken bietet sich typischerweise ein Bild aus Lerninhalten und formalen, z.B. der Navigation dienenden Elementen. Informationen über andere Lerner, die vielleicht sogar zur selben Zeit mit denselben Ressourcen arbeiten, gibt es nicht. Der einzige (wenig hilfreiche) Aufschluß über Lernmittel–Nutzung durch andere Lerner ist oft ein Zugriffs–Zähler im Seitenfuß, der bestenfalls signalisieren kann, *daß* auf ein Dokument zugrgriffen wird, nicht aber *von wem*. Spontane Kommunikation zwischen den Nutzern eines Dokuments ist unmöglich. Da diese im nicht–unterrichtlichen WWW–Betrieb typischweise unerwünscht ist, müssen zur Implementierung einer Lösung einige Hürden genommen werden.

Eine bewährte Lösung besteht in der Einbindung eines Icons, das einen Nutzer des Dokuments symbolisiert, in der jeweiligen Anzahl gleichzeitiger Nutzer. Bei Positionierung des Maus–Zeigers über diesen Icons werden Lerner–Details als sogenannte ALT–Tags oder als „onMouseOver“–Texte sichtbar gemacht. Ein Klick auf einen Lerner (Icon) führt dann beispielsweise zu einer Einladung zu einem Chat oder zum Öffnen eines E-Mail–Clients.

Aus Server–Perspektive ist das „Verweilen auf einem Dokument“ nicht feststellbar, da der Dialog zwichen Client und Server lediglich aus der Anfrage nach einem Dokumnet und dessen Übermittlung besteht. WWW–Server „sehen“ daher leider nur das Laden eines Dokuments, nicht das Verlassen. Bereits Augenblicke nach dem Servieren einer Seite kann aus Sicht des Servers unter normalen Umständen nicht mehr festgestellt werden, ob die Seite von dem jeweiligen Nutzer gelesen wird, ob sie bereits wieder verlassen wurde oder ob der Nutzer sogar seinen Browser bereits geschlossen und seinen Leseprozeß beendet hat.

Normalerweise ist aus Sicht des Servers ebenfalls nicht unterscheidbar, ob ein Dokument infolge eines unmittelbaren menschlichen Kommandos abgerufen wurde, oder ob es sich bei einem Client vielleicht um ein Programm wie z.B. das Indexierungs–Programm einer Suchmaschine handelt.

Beispiel

Die hier vorgeschlagene Beispiel–Implementierung verwendet die Nutzer–Authentifizierung nach dem Muster „Dynamische Erteilung von Zugriffs–Rechten“ zur Identifizierung von Lernern. Nach dem „Login“ in die jeweilige Lernmittel–Datenbank ist während des folgenden Dialogs zwischen Client und Server der Login–Name des Lerners bekannt, der wiederum durch eine Datenbank zur Studenten–Verwaltung in deren reale Namen aufgelöst werden können. Falls auch Lesern, die nicht der jeweiligen Lernergruppe angehören, Zugriff gewährt werden soll, kann zu diesem Zweck ein Gast–Zugang mit öffentlich dokumentiertem Passwort eingerichtet werden. Durch dieses Verfahren kann das System Zugriffe menschlicher Nutzer von maschinellen Zugriffen (etwa durch Suchmaschinen) unterscheiden.

Die Feststellung der „Anwesenheit“ eines Lesers „auf“ einem Dokument kann auf zwei unterschiedlichen Wegen erfolgen: Ein Java–Applet in der jeweiligen Seite kann eine Netzwerk–Verbindung zu einer zu dieem Zweck bereitsgestellten Server aufbeuen und aufrecht erhalten, solange das Dokument vom Browser dargestellt wird. Diese Strategie kann das Probelm des nach der Dokumnet–Übertragung erfolgenden Kontaktabbruch zwichen Server und Client durch die Eröffnung eines neuen Kommunikationskanals lösen. Dies ist jedoch mit einigen Nachteilen verbunden: Zunächst muß das jeweilige Applet und der zugehörige Server entwickelt werden. Beide müssen auf einem für diesen Zweck reservierten TCP/IP–Port kommunizieren, was je nach Kontext an ggf. installierten Firewalls scheitern kann. Um die andauernde Rezeption des Materials zu überprüfen, muß die beschriebene Kommunikation *kontinuierlich*, ggf. in einem definierten Zeitraster erfolgen, was sowohl den Datenverkehr in den jeweils beteiligten Netzwerken und die Prozessor–Belastung des Server–Rechners erhöht. Der letztlich ausschlaggebende Grund gegen die Verwendung eines Java–Applets war im Fall der bisherigen Anwendungen dieses Musters die Tatsache, daß die Unterstützung der Sprache Java auf der Client–Seite nicht unbedingt vorausgesetzt werden kann. Nicht jeder Browser versteht Java, nicht jeder Browser versteht jede Version der Sprache Java und schließlich kann der jeweilige Nutzer seinen Browser auch so konfigurieren, daß er Java–Applets nicht

ausführt, selbst wenn dies vom Browser unterstützt würde. Da die gewünschte Kommunikations–Infrastruktur kohärent für alle Mitglieder der Lernergruppe (sowie für Besucher, die im Rahmen der jeweiligen Kurse zum Besuchen des Materials und zur Kommunikation mit Studierenden eingeladen wurden) gewünscht wurde, war eine andere, rein serverbasierte Lösung erforderlich.

Diese Lösung besteht auf Hypothesen, die über die Verweildauer von Lernern „auf“ Dokumenten aufgestellt werden. Existiernede Implementieungen gehen von einem fest definierten 3–minütigen Zeitfenster aus, während dessen vermutet wird, daß ein Dokument nach seiner Übertragung an einen Client vom Lerner rezipiert wird. In den Fällen, in denen ein Dokument lediglich beim „Surfen“ aufgerufen wurde, ohne näher studiert zu werden, wäre die 3-Minuten–Zeitspanne viel zu lang. Jedoch wird der auf der Severseite verwaltete „Aufenthaltsort“ des Lersers bei jeder Anfrage neuer Dokumente überschrieben und aktualisiert. Diese Lösung kommt ohne zusätzliche Kommunikation zwischen Client und Server aus, erfordert keine Java–Unterstützung und liefert dennoch relativ verläßliche Informationen über die jeweils gegenwärtig rezipierten Inhalte. Tatsächlich falsch sind diese Informationen nur in denjenigen Zeitspannen (die kürzer als 3 Minuten sind), in denen auf der Nutzerseite der Leseprozeß durch den Menschen abgebrochen wird (was auch ein Java–Applet nicht erkennen könnte), wenn die Recherche außerhalb der Lernmitteldatenbank fortgesetzt wird oder wenn der Browser geschlossen und die Sitzung beendet wird.

Zukünftige Weiterentwicklungen dieses Musters könnten die zu Beginn definierte, triviale 3-Minuten–Hypothese durch die Beobachtung und Auswertung der Interaktion zwischen Lerner und Server mit realistischeren Vorhersagen über die wahrscheinliche Verweildauern ersetzen.

4.9. Generisches Format zur Leistungsbewertung

Name: Generisches Format zur Leistungsbewertung	**Letzte Änderung:** 5. Januar 2001
Synonyme: Prozentuale Leistungsbewertung	**Status:** Entwurf
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Die Formate, in denen Bewertungen von Lernerleistungen codiert werden, unterscheiden sich zwischen einzelnen Institutionen und Bildungssystemen teils beachtlich. Vergleichbare Bewertungen werden daher in Abhängigkeit vom jeweiligen Kontext durch sehr unterschiedliche Symbole ausgedrückt. Als sekundäre Folge unterscheiden sich auch die jeweiligen Mechanismen der Verarbeitung (wie z.B. der Durchschnittsbildung und der Rundung). Dies führt zu einer Reihe technischer Probleme, zusätzlichen Aufgaben für Lehrende (Redaktion, Kompensation) und/oder zu inkonsistenten (ungerechten) Bewertungen in Lernprozessen, die zwischen mehreren Lerngruppen mit unterschiedlichen Bewertungsformaten koordiniert werden.	
Kurzdarstellung der Lösung: Als Lösung empfiehlt sich die Abbildung der jeweiligen lokalen Bewertungsformate auf ein gemeinsames, lineares und generisches Format wie die Prozent–Skala.	
Gegenanzeigen: –	
Besondere Hilfsmittel: Zur Visualisierung prozentualer Leistungsbewertungen in HTML bietet sich die SeL–Bibliothek `percentbars.lib` an.	
Referenzen: Die Bibliothek `percentbars.lib` wurde in der gegenwärtigen Fassung von mir entwickelt und ist Bestandteil der SeL–Distribution.	
Verwandte Muster: Dieses Muster kann z.B. die Anwendung des Musters „Kollaborative und kummulative Produkt–Bewertung“ (4.13) unterstützen.	
Anmerkungen: –	

Diskussion

Manche Bewertungsformate für Lerner–Leistungen nutzen die Symbole von „A“ bis „F“, manche die von „1“ bis „15“, manche die von „1“ bis „6“ usw. Bei der Skala zwischen „1“ und „6“ steht in einigen Fällen die „1“ und in anderen Fällen die „6“ fuer die beste mögliche Leistung. In vielen Fällen sind sechs Sysmbole nicht genug, um das Spektrum möglicher Lernleistungen ausreichend aufzulösen und es werden zusätzliche Symbole wie „2-“, „2-3“ und „3+“ eingeführt. Wenn solche zusätzlichen Symbole (die keine konsistente Ergänzung des offiziellen

Symbolvorrats darstellen) aber nicht in offiziellen Zeugnissen auftauchen sollen, sondern nur (inoffiziell) während des Lernprozesses angewendet werden, entscheiden sich Lehrer auch für inoffizielle, also eigene arithmetische Interpretationen und Verarbeitungsstrategien für die zusätzlichen Symbole. Obgleich dieses Vorgehen als „gerecht“ interpretiert werden kann, solange die resultierenden Bewertungsskalen auf komplette Lernerguppen gleich angewendet werden, gestaltet sich die Koordination der Leistungsbewertung für mehrere Lerngruppen durch mehrere Lehrer ggf. über die Grenzen von Bildungssystemen hinweg (etwa im Rahmen von „interkulturellem Lernen“, „vernetztem Entwerfen“ etc.) schwierig.

Um dieses Problem zu lösen wird eine Bewertungsskala benötigt, die linear aufgebaut und ausreichend hoch aufgelöst ist, deren Symbolen eindeutige Wertigkeiten zugeordnet sind und deren Verarbeitungsstrategien eindeutig definiert (z.B. Rundung von Nachkommastellen) sind. Da eine solche Skala mit hoher Wahrscheinlichkeit deutlich von den gewohnten offiziellen Skalen zur Gesamtbewertung von Kursen, Halbjahren oder Semestern abweicht, muß sie auch in der Lage sein, Lernleistungen intuitiv und nicht–sprachlich verständlich und vergleichbar darzustellen.

Als Lösung für diese Anforderungen sind eine Reihe von Skalen denkbar. Eine besonders naheliegende (und im Beispiel unten diskutierte) Lösung ist die Prozentskala. Bei der Verwendung jeder selbst definierten Skala ist es wichtig, daß deren Abbildung auf die jeweils gültigen offiziellen Skalen rechtzeitig eindeutig festgelegt wird.

Beispiel

Ganz ähnliche Anforderungen wie die in der Diskussion angesprochenen müssen auch von Computer–Spielen gelöst werden: Wie sollen Werte völlig unterschiedlicher Skalen (z.B. „Gesundheit“, „Munition“ und „Beute“) für eine Gesamtbewertung miteinander in Beziehung gesetzt werden? Wie soll die dafür erforderliche Interoperabilität zwischen verschiedenen (z.B. einzelnen „Levels“) hergestellt und aktuelle Werte fortlaufend intuitiv und nicht–sprachlich verständlich

und vergleichbar visualisiert werden?

Dieses Muster empfiehlt die Anwendung der klassichen Lösung dieses Problems auf dem Spielesektor im Unterricht: Die Verarbeitung und Visualisierung variabler Werte durch sogenannte „Prozentbalken“. Die folgende SeL–Bibliothek enthält Funktionen, die durch einfachen Aufruf Prozentbalken in HTML generieren. Erforderlich sind drei Parameter: Der darzustellende Prozentwert, die Tendenz der Interpretation und Angaben zur visuellen Ausrichtung im HTML–Dokument (siehe Quelltext).

Diese Funktion kann in WWW–basierten Lernumgebungen genutzt werden, um sehr einfach beliebige, in Prozent ausgedrückte Größen zu visualisieren. Zum Zweck der sinnvollen Darstellung werden übergebene Werte nach der dritten Nachkommastelle kaufmännisch gerundet. Dies hat jedoch keine Rückwirkung auf die verarbeiteten Werte. Dadurch ist die Implementierung eigener interner Rundungsstrategien möglich.

```
/* SeL Percent Bar Lib - 2000 by Thomas Fischer,
himSeLf@tfischer.de

usage:

lib "percentbars.lib";
$percentbar("x","y","z")


x,y and z are mandatory parameters, whereas:

x=value to be visualized (0..100, floating point values allowed,
     but will be truncated 2 digits after floating
       point)

y=value interpretation   (0=neutral interpretation,
       p=positive interpretation,
       n=negative interpretation,
       this will result in different color schemes)

z=HTML alignment attribute (e.g. TOP,CENTER,RIGHT,...)


example:
$percentbar("42","p","LEFT")

*/
```

```
!percentbar(
lib.percentbars.pow,lib.percentbars.type,lib.percentbars.align )
  if ($_match($lib.percentbars.pow,",") != "-1") then
    =lib.percentbars.point
$_match($lib.percentbars.pow,",");
    =lib.percentbars.diff $lib.percentbars.point+3;
    =lib.percentbars.td
$_copy($lib.percentbars.pow,0,$lib.percentbars.point);
    =lib.percentbars.pow
$_copy($lib.percentbars.pow,0,$lib.percentbars.diff);
  else
    if ($_match($lib.percentbars.pow,".") != "-1") then
      =lib.percentbars.point
$_match($lib.percentbars.pow,".");
      =lib.percentbars.diff
$lib.percentbars.point+3;
      =lib.percentbars.td
$_copy($lib.percentbars.pow,0,$lib.percentbars.point);
      =lib.percentbars.pow
$_copy($lib.percentbars.pow,0,$lib.percentbars.diff);
    else
    =lib.percentbars.td $lib.percentbars.pow;
    fi
  fi
  if ( $lib.percentbars.pow > 100 ) then
  =lib.percentbars.pow 100;
  =lib.percentbars.td 100;
  fi
  if ( $lib.percentbars.pow < 0 ) then
  =lib.percentbars.pow 0;
  =lib.percentbars.td 0;
  fi

  if ($lib.percentbars.type != "n") then
    if ($lib.percentbars.type != "p" ) then
      =lib.percentbars.type "0";
    fi
  fi
  <SQD>
  <TABLE CELLPADDING=0 CELLSPACING=0</SQD>
  if ( $lib.percentbars.align != "" ) then
    <SQD> BORDER ALIGN="</SQD>
    $lib.percentbars.align
    <SQD>"</SQD>
  fi
  <SQD>><TR><TD>
  <TABLE WIDTH=100 HEIGHT=10 CELLPADDING=0 CELLSPACING=0
BORDER=0>
  <TR></SQD>
  if ($lib.percentbars.pow > 1) then
    <SQD><TD WIDTH=</SQD>
```

```
  $lib.percentbars.td
  <SQD> BGCOLOR="#</SQD>
  if ( $lib.percentbars.type == "p" ) then
  =lib.percentbars.rp $lib.percentbars.td*255/100;
  =lib.percentbars.rp 255-$lib.percentbars.rp;
  =lib.percentbars.gp $lib.percentbars.td*255/100;
  $_hex(2,$lib.percentbars.rp)
  $_hex(2,$lib.percentbars.gp)
  fi
  if ( $lib.percentbars.type == "n" ) then
  =lib.percentbars.gp $lib.percentbars.td*255/100;
  =lib.percentbars.rp $lib.percentbars.td*255/100;
  =lib.percentbars.rp 255-$lib.percentbars.rp;
  $_hex(2,$lib.percentbars.gp)
  $_hex(2,$lib.percentbars.rp)
  fi
  if ( $lib.percentbars.type == "0" ) then
    <SQD>999999</SQD>
  else
    <SQD>66</SQD>
  fi
  <SQD>" ALIGN=RIGHT><FONT SIZE=1 COLOR="#000000"></SQD>
  if ($lib.percentbars.td >= 60) then
    <SQD><B></SQD>
    $lib.percentbars.pow
    <SQD>%</B> </SQD>
  else
    <SQD> </SQD>
  fi
  <SQD></FONT></TD></SQD>
fi
if ($lib.percentbars.pow < 99) then
  <SQD><TD WIDTH=</SQD>
  100-$lib.percentbars.td
  <SQD> BGCOLOR="#</SQD>
  if ( $lib.percentbars.type == "p" ) then
    if ( $lib.percentbars.td != 0 ) then
      <SQD>FFFFFF</SQD>
    else
      <SQD>FF0066</SQD>
    fi
  fi
  if ( $lib.percentbars.type == "n" ) then
    if ( $lib.percentbars.td != 0 ) then
      <SQD>FFFFFF</SQD>
    else
      <SQD>00FF66</SQD>
    fi
  fi
  if ( $lib.percentbars.type == "0" ) then
  <SQD>FFFFFF</SQD>
```

```
    fi
    <SQD>"><FONT SIZE=1 COLOR="#000000"></SQD>
    if ($lib.percentbars.td <= 59) then
      <SQD> <B></SQD>
      $lib.percentbars.pow
      <SQD>%</B> </SQD>
    else
      <SQD> </SQD>
    fi
    <SQD></FONT></TD></SQD>
  fi
  <SQD></TR></TABLE></TD></TR></TABLE></SQD>
;
```

4.10. Upload physischer Objekte

Name: Upload physischer Objekte	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich eingesetzt
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Die Kommunikation von Inhalten mittels Computer und Netzwerken erfordert die Digitalisierung (und somit die Digitalisierbarkeit) der entsprechenden Inhalte. Während dies im Fall von Texten und Multimedia–Inhalten problemlos möglich ist, bleiben natürliche Objekte, deren besondere Qualität in ihrer physischen Originalität besteht, diesen Kommunikationsformen vorbehalten. Oftmals sind aber es aber genau diese Objekte, die Lernern „in die Hand gegeben“ werden sollen.	
Kurzdarstellung der Lösung: Eine beliebige Form von Lager–Möglichkeit	
Gegenanzeigen: Selbstverständlich ist dieses Verfahren nur innerhalb sehr begrenzter räumlicher Distanzen sinnvoll. Es scheint zumindest prinzipiell nicht ausgeschlossen, sowohl den „Upload“ als auch die Inspektion physischer Objekte (etwa durch Installation von Kameras und Postversand) auch über größere Distanzen zu unterstützen. Dies negiert jedoch die vielen Vorteile unmittelbarer physischer Zugänglichkeit.	
Besondere Hilfsmittel: Neben einem (allen Kursteilnehmern zugänglichen) Ort zur Aufbewahrung der Objekte scheint die Integration in ein Dokument–Management–System angebracht, das neben der üblichen Administration von Texten, Bildern usw. in die Lage versetzt werden muß, auch physische Objekte zu verwalten. Dies kann die Speicherung von beschreibenden Texten, Schlüsselwörtern und Angaben zum Besitzer des Objekts umfassen. Dieses Verfahren kann durch die automatische Ausgabe eines Ettiketts zur Identifizierung einzelner Objekte und für die Zuordnung einzelner Objekte zu zusätzlicher digitaler Information deutlich vereinfacht werden.	
Referenzen: Die Idee zu diesem Muster hatte Lorne Falk im Februar 2000. Sie wurde im Rahmen des Kurses DS6 erfolgreich angewendet.	
Verwandte Muster: –	
Anmerkungen: –	

Diskussion

Der in Computern enthaltene Speicher kann nichts außer digitalen Daten verwalten. Dies reicht zwar, um eine Vielzahl von Multimedia–Datenformaten für Texte, Bilder, Audio und Video zu verarbeiten. Gegenstände, die jedoch (entwe-

der im Rahmen einer konkreten Unterrichtssituation oder aber grundsätzlich) nicht in digitale Daten zerlegbar sind oder nicht zerlegt werden sollten, können nicht mit dem Computer bearbeitet werden. Das digitale Codieren und das damit implizit einhergehende Kopieren erfordert einerseits einen vom Medientyp abhängig unterschiedlich intensiven Arbeitsaufwand, andererseits vermag es das Original des jeweiligen Medienobjekts nicht vollständig mit allen Elementen und Attributen abzubilden. Soll z.B. ein Buch im computervermittelten Unterricht an Schüler übergeben werden, bietet es sich an, dieses vorher von der ersten bis zur letzten Seite visuell zu scannen und gegebenenfalls die auf den Seiten enthaltene Schrift mit einer Texterkennung in digitalen Text zu konvertieren. Dies ist ein arbeitsintensiver Vorgang, der sich dadurch lohnen kann, daß der Text fortan beliebig oft kopierbar ist und somit aus einem physischen Quell–Objekt eine beliebig hohe digitale Auflage gewonnen wird und eine beliebige Anzahl von Lernern mit eigenen Kopien ausgestattet werden kann. Wenn aber Elemente des Buchs im Unterricht thematisiert werden sollen, die — jenseits des textuellen Inhalts — nicht durch visuelle Scans kommuniziert werden können, fehlt der Kommunikationskanal für diese Inhalte. Dies ist der Fall, wenn z.B. ein altes Buch Gegenstand des Unterrichts ist: Das Material des Einbands, der Seiten, sein Gewicht, sein Geruch oder die bei der Bindung verwendete Technik sind Attribute und Eigenschaften, die sich am besten durch das originale physische Objekt selbst und nicht über den Computer transportieren lassen.

Bei ihren ersten praktischen Anwendungen von Computern und Netzwerken im Unterricht tendieren Lehrer zu ganz besonderer Vorsicht. So verlangen Lehrer im Rahmen ihrer ersten Online–Kurse z.B. oft nach Größenbeschränkungen für studentische *Uploads*, weil sie ein „Sprengen“ der jeweiligen Server–Kapazitäten fürchten. Diese Befürchtung ist zwar aus technischer Sicht durchaus berechtigt, dies sollte aber keinesfalls zu starken Reglementierungen oder zu übertriebenen didaktisch/planerischen Zugeständnissen, da mir kein einziger Fall eines „gesprengten“ Kurs–Servers bekannt ist. Ich halte es für wichtig, darauf hinzuweisen, daß bei der ersten Anwendung dieses Musters Probleme mit Diebstahl und Sachbeschädigung befürchtet wurden, was sich in diesem Fall glücklicherweise auch als übertrieben erwiesen hat. Sämtliche zur Verfügung gestellten Objekte

konnten nach Ende des Kurses unbeschädigt an ihre Eigentüber zurückgegeben werden (bei den Lehrenden entstand der generelle Eindruck, daß das Beitragen eigener Objekte zu den Lernmaterialien durch die Lerner einen sehr positiven Effekt auf die *Kursmoral* hatte). Eine *Webcam* kann sicherlich auch helfen, Diebstahl und Beschädigungen vorzubeugen, was weiterhin den Vorteil hätte, daß von den gesammelten Objekten zumindest ein Kamerabild online abrufbar wäre.

Beispiel

In Fall der *Silkroad Interactive Encyclopedia*[13] wurde eine Online–Datenbank für vier unterschiedliche Medientypen entwickelt: Für Klartext, Multimedia–Dateien (Jpeg, Quicktime, VRML, mp3, AVI, Midi usw.), Links (URLs) und schließlich für physische Objekte. Für die Lagerung der Objekte wurde ein Bücherregal reserviert, daß öffentlich in einem der Designstudios aufgestellt wurde. Die Online–Datenbank konnte von den jeweiligen Besitzern eingegebene textuelle Beschreibungen der Objekte verwalten sowie Etiketten mit Referenz–Nummern zur vereinfachten Zuordnung erzeugen.

[13] siehe Falk, Lorne, Cristiano Ceccato und Thomas Fischer: [7]

4.11. Automatische Erzeugung von Keyword–Listen

Name: Automatische Erzeugung von Keyword–Listen	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich eingesetzt (mit Texten in englischer Sprache)
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Lernerbestimmte Rezeption von Unterrichtsmaterial erfordert Mechanismen, die das Auffinden gesuchter Information gestatten. Für diesen Zweck werden typischerweise Suchmaschinen bereitgestellt. Diese durchsuchen entweder komplette Datenbestände in Echtzeit nach Suchbegriffen, oder sie durchsuchen Keyword–Listen, die zuvor aus den Lernmaterialien gewonnen werden mußten. Obgleich die letztere Strategie deutlich weniger arbeitsintensiv ist als die erste, stellt die Erzeugung der erforderlichen Keyword–Listen ein redaktionelles Problem dar. Darüberhinaus sind es meist die Schlüsselwörter von Texten, die in Text–Rekonstruktions–Aufgaben wie z.B. Lückentests abgefragt werden. Für die automatisierte Herstellung derartiger Lernkontrollen ist daher ebenfalls ein Verfahren zur Erzeugung von Keyword–Listen erforderlich.	
Kurzdarstellung der Lösung: Dieses Programm extrahiert Schlüsselwörter aus Fließtext. Die Auswahl der Schlüsselwörter basiert auf vordefinierten, fachspezifischen Listen „profaner“ Wörter mit mehr als vier Buchstaben und „nicht–profaner“ Wörter mit weniger als fünf Buchstaben.	
Gegenanzeigen: Anwendungen auf Texte in anderen Sprachen werden ggf. nicht oder nur schlecht funktionieren (siehe Diskussion).	
Besondere Hilfsmittel: Perl–Interpreter	
Referenzen: –	
Verwandte Muster: –	
Anmerkungen: Die hier vorgestellte Lösung hat sich zur Herstellung relativ langer Listen von Schlüsselwörtern bewährt, die Zusammenstellung der im Programm erforderten Listen zur Ausnahmebehandlung ist jedoch ihrerseits ein recht arbeitsintensiver redaktioneller Prozeß. Die Auswahl der Schüsselwörter erfolgt auf der Basis völlig trivialer Regeln und ohne jede semantische Auswertung. Zukünftige Versionen dieses Musters sollten elegantere Strategien verfolgen und z.B. auch Fachwörterbücher zur Auswahl der Schlüsselwörter heranziehen.	

Diskussion

Online–Metadaten, lokale Such–Indexe und Lernkontrollen auf Wortebene sind drei wichtige Gründe für die Herstellung von Keyword–Listen für Lernmittel–

Datenbanken. Die hierfür erforderliche semantische Auswertung der Fließtexte ist nicht ganz einfach automatisierbar. Die Identifikation fachterminologischer Begriffe und interdisziplinärer semantischer Differenzen kann kaum in einfachen Algoritmen beschrieben werden. Der hier vorgestellte, triviale Lösungsansatz basiert auf der Annahme, daß Begriffe mit wachsender Zeichen–Anzahl eine tendenziell höhere semantische Signifikanz besitzen, während sich kurze Wörter tendenziell weniger dazu eignen, den sie umgebenden Inhalt zu repräsentieren. Daher müssen bekannte Ausnahmen dieser Annahme definiert werden. Diese Strategie ist nicht abstrakt genug, um als generische Lösung gleichartiger Probleme in der Zukunft herangezogen zu werden, da mit jedem neuen zu verarbeitenden Text potentiell neue, noch nicht definierte Ausnahmen auftreten können. Die Notwendigkeit redaktioneller Betreuung ist daher nicht beseitigt, auch wenn der eigentliche Prozeß der Isolierung von Schlüsselwörtern mit diesem Programm sehr schnell erfolgen kann.

Sowohl die Annahme, daß die Länge eines Wortes eine Aussagekraft über seine semantische Signifikanz hat, als auch die hier zur Unterscheidung benutzte Grenze von 4 Zeichen führt bei der Anwendung auf andere Sprachen möglicherweisen zu Problemen. Dieses Muster mit den beschriebenen Annahmen wurde erfolgreich auf Fließtexte in englischer Sprache angewendet. Eine Anwendung auf Texte in chinesischer Schrift wäre sinnlos.

Beispiel

Das im folgenden Quelltext dargestellte Programm übergibt den Inhalt einer Text–Datei an eine Variable. Die darin enthaltenen Leerzeichen und Zeilenumbrüche werden als Trennzeichen für die Zerlegung in eine Liste benutzt. Aus dieser Liste werden anschließend sämtliche „profanen“ Wörter mit mehr als vier Buchstaben und alle nicht–profanen Wörter unter 5 Buchstaben entfernt. Praktisch empfehlen sich möglicherweise andere Schnittstellen zum Lernmaterial, etwa zu Datenbank–Verwaltungssystemen oder die Einschachtelung dieses Programms in eine übergeordnete Schleife, die komplette Verzeichnis–Strukturen durchsucht und verarbeitet.

```
#!/usr/bin/perl

# keyword list generator 1999/2000 by Thomas Fischer

$INPUTFILE="/tmp/file.txt";

# prophane > 4 characters
@PROFAN=("caused","beneath","between","include","around",
"various","applied","useful","ensure","formed","occurs",
 [Liste gekuerzt]
"doubles","times","inadequate","useable","approx");

#nonprophane < 5 characters
@NONPROFAN=("mesh","roof","bulb","slab","flat","fire",
"wall","base","pvc","damp","soil","pipe","site","air",
 [Liste gekuerzt]
"span","ramp","code","cost","raft","new","ties");

open(LESEN,"<$INPUTFILE");
while(<LESEN>){
  $CONTENT=$CONTENT.$_;
  }
$ORIGINAL=$CONTENT;
close(LESEN);

$CONTENT=~s/\n\n\n/ /g;
$CONTENT=~s/\n\n/ /g;
$CONTENT=~s/\n/ /g;
$CONTENT=~s/\-/ /g;
$CONTENT=~tr/[A-Z]/[a-z]/;
$CONTENT=~s/[^a-zA-Z0-9\ ]/ /g;
$CONTENT=~s/ /\n/g;

@LISTIN=split(/\n/,$CONTENT,);
$j=0;
for(@LISTIN){
  #recognize repetitions:
  $repeat=0;
  $k=0;
  $CJ=$LISTIN[$j]; chop($CJ);
  for($k=0;$k<$j;$k++){
    $CK=$LISTIN[$k]; chop($CK);
    if(($LISTIN[$j] eq $LISTIN[$k])||
       ($CJ eq $LISTIN[$k])||
       ($CK eq $LISTIN[$j])){
      $repeat=1;
    }
  }
  if((index($LISTIN[$j],"lly") eq -1) && ($repeat eq 0)){
    #recognize profane words
    $profan=0;
```

```
    $k=0;
    for(@PROFAN){
      if($LISTIN[$j] eq $PROFAN[$k]){
$profan=1;
      }
      $k++;
    }
    #recognize non-profane words
    $nprofan=1;
    $k=0;
    for(@NONPROFAN){
      if($LISTIN[$j] eq $NONPROFAN[$k]){
$nprofan=0;
      }
      $k++;
    }
    if(((length($LISTIN[$j])>4)&&
($profan eq 0))||
       ((length($LISTIN[$j])<5)&&
($nprofan eq 0))){
      $KEYWORDS=$KEYWORDS.$LISTIN[$j].",";
    }
  }
  $j++;
}
$j=0;
@LIST="";
$DETAILTITLE="";
chop($KEYWORDS);
@KEYW=split(/,/,$KEYWORDS);
@ORD=sort(@KEYW);
$KEYWORDS="";
for(@ORD){$KEYWORDS=$KEYWORDS.$_.",";}
chop($KEYWORDS);
$KEYWORDS="<keywords>".$KEYWORDS."</keywords>";
print "$KEYWORDS\n";
$KEYWORDS="";
exit;
```

4.12. Turing–Maschinen–Emulator

Name: Turing–Maschinen–Emulator	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich eingesetzt
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer
Problem: Die Turing–Maschine ist ein zentrales Konzept der Theorie der Informationsverarbeitung und der Künstlichen Intelligenz. Sie wurde als Gedankenexperiment entworfen, bis heute als solche behandelt und nie gebaut. Die Turing–Maschine wird umfassend in der Literatur diskutiert, ihrem Stellenwert kann praktischer Unterricht aber kaum gerecht werden, da von ihr kein „anfaßbares“, interaktives Lernmodell zur Verfügung steht.	
Kurzdarstellung der Lösung: Dieses Muster ist ein SeL–Programm, das beliebige Instanzen der von Alan Turing entworfenen hypothetischen Maschine als Online–Lernsimulationen herstellen kann. Die Maschinen (bzw. ihr Programmcode) werden auf der Basis individueller Parameter mit frei wählbarer Länge des Magnetbandes und beliebigen Programmstrukturen dynamisch erzeugt.	
Gegenanzeigen: –	
Besondere Hilfsmittel: SeL–Interpreter	
Referenzen: –	
Verwandte Muster: –	
Anmerkungen: –	

Diskussion

In Anlehnung an Albert Einsteins Hypothesen entwarf Turing im Jahr 1936 seine Maschine als Gedankenexperiment. Er illustrierte damit seine Hypothese, daß Maschinen theoretisch in der Lage seien, sämtliche formal beschreibbaren mentalen Prozesse auszuführen. Dieses Kriteruim erfüllt die Turing–Maschine jedoch nur, wenn sie mit einem *endlosen* Magnetband ausgestattet wird. Da ein solches Band nicht existieren kann, kann auch die Turing–Masachine nicht existieren. Dies ist zwar wahr, die jüngere Geschichte der Datenverarbeitung zeigt aber, daß sich unsere alltäglichen Rechner mit ihren endlichen Speichern als sehr nützliche Werkzeuge erweisen können.

Unsere heutigen Computer können theoretisch betrachtet nicht jeden beschreibbaren mentalen Prozeß nachbilden, da sie nur über *endliche* Mengen an Speicher verfügen. Allerdings sind sie daher auch baubar und anwendbar. Genau

genommen hat die Endlichkeit moderner Computer–Speicher sogar einen großen Vorteil: Start und Ende des Speichers bieten Bezugspunkte, durch die viele Berechnungen wesentlich vereinfacht werden können, wenn ihre Adresse innerhalb des Speichers in die Operation einbezogen wird.

Zwei charakteristische Merkmale des modernen Computers (der sogenannten von–Neumann–Architektur) sind der bedingte Sprung (dynamische Verzweigungen von Programm–Abläufen) und die äquivalente Behandlung von Programmen und Daten (beide werden im selben Speicher verwaltet). Diese Konzepte lassen sich direkt auf die Prinzipien der Turing–Maschine zurückführen. Aus diesem Grund besitzt sie neben ihrem theoretischen Stellenwert auch eine hohe praktische Relevanz.

Als neuer Lerninhalt ist das *Konzept Turing–Maschine* jedoch nicht einfach zu verinnerlichen. Daß sie *idealisiert* und *nicht baubar* ist, erschwert dieses Problem. Dieses Pattern verfolgt den pragmatischen Ansatz, Turing–Maschinen nicht mit endlosen Speichern bereitzustellen, sondern wie den modernen Computer mit endlichen Speichern auszustatten. Es gestattet jedoch die Herstellung beliebiger Instanzen der Turing–Maschine, also auch solcher mit sehr langen Bändern (einziges Limit ist die Darstellungs–Kapazität des Browsers auf der Lernerseite). Die Ausstattung mit zwar endlichen, dafür aber baubaren Speichern, ermöglicht längst keinen Apparat, der sämtliche mentalen Prozesse ausführen kann. Jedoch gestattet dieser Kompomiß die individuelle Herstellung von Modellen kleiner Turing–Maschinen, die als interaktive Lernsimulationen das Verstehen von Turings Entwurf erleichtern können.

Beispiel

Abbildung 4.6 zeigt eine Instanz der Turing–Maschine, die auf der Basis von Lerner–Eingaben durch den unten aufgeführten SeL–Quellcode erzeugt wurde. Die vom Lerner konfigurierbaren Variablen sind die Länge des mit *Daten* versehbaren Bandes (Anzahl der *Zellen*), die zusätzliche Länge des Bandes über diesen Bereich hinaus, die Anzahl der *Maschinen–Status* sowie die Definition des Verhaltens im beim Lesen von `1` und `0` jedes Maschinen–Status.

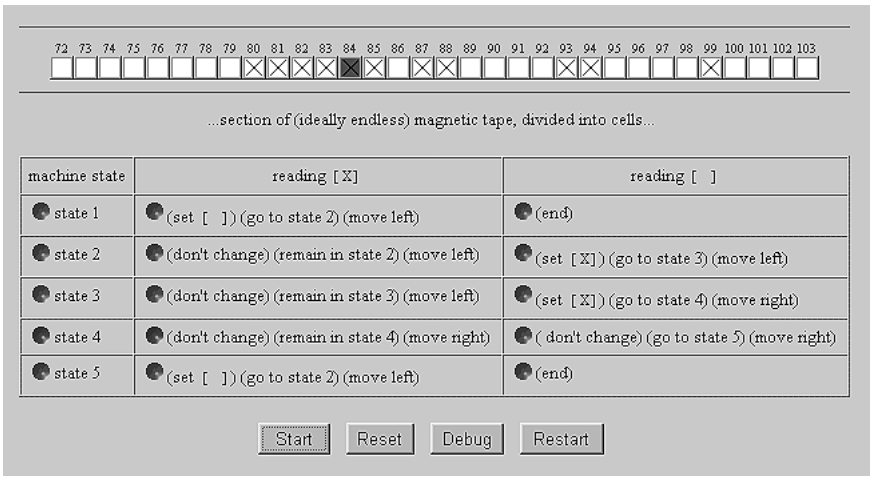


Abbildung 4.6.: Eine Instanz der Turing–Maschine

Quellcode:

```
/* Turing Machine Generator (c) 2000 by Thomas Fischer, \
himself@tfischer.de */

!runlevel1(void)
<SQD>
<HTML>
<!-- Turing Machine Generator (c) 2000 by Thomas Fischer, \
himself@tfischer.de -->
<HEAD><TITLE>Turing Machine Configuration</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function next(){
    if((isNaN(document.CONF.CELLS.value)==true)||
       (isNaN(document.CONF.STATES.value)==true)||
       (document.CONF.CELLS.value==’’)||
       (document.CONF.STATES.value==’’)||
       (document.CONF.CELLS.value<1)||
       (document.CONF.STATES.value<1)){
alert(’Please enter positive numbers!’);
}
else{
document.CONF.submit();
}
    }
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#CCCCCC" TEXT="#333333" LINK="#333333" \
ALINK="#333333" VLINK="#333333">
<CENTER>
<TABLE HEIGHT="100%"><TR><TD>
```

```
<CENTER>
<TABLE><TR><TD>
<FORM NAME="CONF" METHOD="POST">
<TABLE>
<TR><TD><FONT SIZE=2>Number of editable tape cells: </FONT>\
</TD><TD><INPUT TYPE="TEXT" SIZE=3 NAME="CELLS"></TD></TR>
<TR><TD><FONT SIZE=2>Auxiliary cells at each side: </FONT>\
</TD><TD><INPUT TYPE="TEXT" SIZE=3 NAME="AUX"></TD></TR>
<TR><TD><FONT SIZE=2>How many rules/states? </FONT></TD><TD>\
<INPUT TYPE="TEXT" SIZE=3 NAME="STATES"></TD></TR></TABLE>
<P><INPUT TYPE="HIDDEN" NAME="RUNLEVEL" VALUE="1"><CENTER>\
<INPUT TYPE="BUTTON" VALUE="Program Machine" \
onClick="next()"></CENTER>
</FORM>
</TD></TR></TABLE>
</CENTER>
</TD></TR></TABLE>
</CENTER>
</BODY>
</HTML>
</SQD>
;

!runlevel2(CELLS,STATES,AUX)
<SQD>
<HTML>
<!-- Turing Machine Generator (c) 2000 by Thomas Fischer, \
himself@tfischer.de -->
<HEAD><TITLE>Turing Machine Programming Panel</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function next(){
document.CONF.submit();
}
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#CCCCCC" TEXT="#333333" LINK="#333333" \
ALINK="#333333" VLINK="#333333">
<CENTER>
<TABLE HEIGHT="100%"><TR><TD>
<CENTER>
<TABLE><TR><TD>
<FORM NAME="CONF" METHOD="POST">
<TABLE BORDER CELLPADDING=5 CELLSPACING=0>
<TR><TD ALIGN="CENTER"><FONT SIZE=2>machine state</FONT></TD>\
<TD ALIGN="CENTER"><FONT SIZE=2>reading </FONT><TT>[X]</TT></TD>\
<TD ALIGN="CENTER"><FONT SIZE=2>reading </FONT><TT>[ ]</TT></TD>\
</TR>
</SQD>
for (i,1,$STATES) do
<SQD><TR><TD ALIGN="CENTER"><FONT SIZE=2></SQD>$i<SQD></FONT></TD>
<TD>
```

```
    <SELECT NAME="doX</SQD>$i<SQD>">
<OPTION VALUE="0">set [  ]</OPTION>
<OPTION VALUE="nset">don't change</OPTION>
<OPTION VALUE="end">end</OPTION>
    </SELECT>
    <SELECT NAME="csX</SQD>$i<SQD>"></SQD>
for(j,1,$STATES) do
<SQD><OPTION VALUE="</SQD>$j<SQD>"></SQD>
if($i==$j)then
<SQD>remain in </SQD>
else
<SQD>go to </SQD>
fi
<SQD>state </SQD>$j<SQD></OPTION></SQD>
done
    <SQD></SELECT>
    <SELECT NAME="mvX</SQD>$i<SQD>">
<OPTION VALUE="l">move left</OPTION>
<OPTION VALUE="r">move right</OPTION>
    </SELECT>
</TD>
<TD>
    <SELECT NAME="do_</SQD>$i<SQD>">
<OPTION VALUE="1">set [X]</OPTION>
<OPTION VALUE="nset">don't change</OPTION>
<OPTION VALUE="end">end</OPTION>
    </SELECT>
    <SELECT NAME="cs_</SQD>$i<SQD>"></SQD>
for(j,1,$STATES) do
<SQD><OPTION VALUE="</SQD>$j<SQD>"></SQD>
if($i==$j)then
<SQD>remain in </SQD>
else
<SQD>go to </SQD>
fi
<SQD>state </SQD>$j<SQD></OPTION></SQD>
done
    <SQD></SELECT>
    <SELECT NAME="mv_</SQD>$i<SQD>">
<OPTION VALUE="l">move left</OPTION>
<OPTION VALUE="r">move right</OPTION>
    </SELECT>
</TD></TR>
</SQD>
done
<SQD></TABLE>
<CENTER>
<FONT SIZE=2>Boot sequence automatically loads state 1 when 1\
<SUP>st</SUP></FONT> <TT>[X]</TT> <FONT SIZE=2>is found.</FONT>
</CENTER>
<INPUT TYPE="HIDDEN" NAME="CELLS" VALUE="</SQD>$CELLS<SQD>">
```

```
<INPUT TYPE="HIDDEN" NAME="AUX" VALUE="</SQD>$AUX<SQD>">
<INPUT TYPE="HIDDEN" NAME="STATES" VALUE="</SQD>$STATES<SQD>">
<INPUT TYPE="HIDDEN" NAME="RUNLEVEL" VALUE="2"><BR>
<CENTER><INPUT TYPE="BUTTON" VALUE="Create Machine" \
onClick="next()"></CENTER>
</FORM>
</TD></TR></TABLE>
</CENTER>
</TD></TR></TABLE>
</CENTER>
</BODY>
</HTML>
</SQD>
;

!machine(CELLS,STATES,AUX)
=STARTADDRESS 71;
=IMGSHIFT 0;
=LEDSTART 2*$AUX+$IMGSHIFT+$CELLS-1; /* -1 for IMG[0] */
<SQD>
<HTML>
<!-- Turing Machine Generator (c) 2000 by Thomas Fischer, \
himself@tfischer.de -->
<HEAD><TITLE>Your Turing Machine</TITLE>
<SCRIPT LANGUAGE=JavaScript>
<!-- hide from old clients
var p;
var url_base=document.location;
var rxp=/(.*)\/([^\/])/;
rxp.exec(url_base);
url_base=RegExp.$1+"/";

var imgs=new Array();

function prelude(){
    for(m=0;m<prelude.arguments.length;m++){
imgs[m]=new Image();
imgs[m].src=prelude.arguments[m];
}
    }

prelude(url_base+"pix/0bit.gif",url_base+\
"pix/0bitr.gif",url_base+"pix/1bit.gif",url_base+\
"pix/1bitr.gif",url_base+"pix/gled.gif",\
url_base+"pix/rled.gif");

function lenchk(t){
if((t>0)&&(t<=</SQD>$LEDSTART+1<SQD>)){
    return false;
    }
    else{
```

```
    window.status='machine ran out of tape';
    delay(3);
    return true;
    }
}

function delay(s){
z=-1;
now=new Date();y=now.getSeconds();y=(y+s)%60;
while(y!=z){now2=new Date();z=now2.getSeconds();}
y=0;z=0; return true;
}

function res()
{
for(q=0; q<=</SQD>$AUX+$CELLS+$AUX-1<SQD>; q++){
window.document.images[q].src = url_base+"pix/0bit.gif";
}
for(q=</SQD>$AUX+$CELLS+$AUX<SQD>;q<=</SQD>
$AUX+$CELLS+$AUX-1+$STATES*3
<SQD>; q++){
window.document.images[q].src = url_base+"pix/gled.gif";
}
}

function stp(){
for(q=0; q<=</SQD>$AUX+$CELLS+$AUX-1<SQD>; q++){
    if(window.document.images[q].src==
url_base+"pix/1bitr.gif"){
window.document.images[q].src=
url_base+"pix/1bit.gif"
}
    if(window.document.images[q].src==
url_base+"pix/0bitr.gif"){
window.document.images[q].src=
url_base+"pix/0bit.gif"
}
    }
window.status='';
return true;
}

function sele(p){
if(window.document.images[p+</SQD>
$AUX-1
<SQD>].src == url_base+"pix/1bit.gif"){
window.document.images[p+</SQD>$AUX-1<SQD>].src=
url_base+"pix/0bit.gif";
}
else{
window.document.images[p+</SQD>$AUX-1<SQD>].src=
```

```
url_base+"pix/1bit.gif";
}

function machine()
{
window.status='running...';
delay(1);
for (i=1; i<=</SQD>$AUX+$CELLS+$AUX<SQD>; i++)
{
window.document.images[i-1].src=\
url_base+"pix/0bitr.gif";
delay(1);
window.document.images[i-1].src=\
url_base+"pix/0bit.gif";
if (window.document.images[i].src==
url_base+"pix/1bit.gif"){
state1(i+1);
break;
}
}
lenchk(i);
}
</SQD>

=TABLE <SQD><TR><TD ALIGN="CENTER"><FONT SIZE=2>machine state\
</FONT></TD><TD ALIGN="CENTER"><FONT SIZE=2>reading </FONT>\
<TT>[X]</TT></TD><TD ALIGN="CENTER"><FONT SIZE=2>reading </FONT>\
<TT>[ ]</TT></TD></TR></SQD>;

for(i,1,$STATES)do
=h "doX"$i; =doX $_cgival($h);
=h "csX"$i; =csX $_cgival($h);
=h "mvX"$i; =mvX $_cgival($h);
=h "do_"$i; =do_ $_cgival($h);
=h "cs_"$i; =cs_ $_cgival($h);
=h "mv_"$i; =mv_ $_cgival($h);
=n $i-1;
=n $n*3;

=TABLE $TABLE<SQD><TR>
<TD VALIGN="TOP"><IMG SRC="pix/gled.gif" ALIGN="LEFT">
<FONT SIZE=2>state </SQD>$i<SQD></FONT></TD>
<TD VALIGN="TOP"><IMG SRC="pix/gled.gif" ALIGN="LEFT">
<FONT SIZE=2>
</SQD>;
if($doX=="end")then =TABLE $TABLE<SQD> (end) </SQD>;
else
if($doX=="0")then =TABLE $TABLE<SQD> (set</FONT> <TT>[ ]</TT>\
<FONT SIZE=2>) </SQD>;fi
if($doX=="nset")then =TABLE $TABLE<SQD> (don't change) </SQD>;fi
if($csX==$i)then
```

```
=TABLE $TABLE<SQD> (remain in state </SQD>$csX<SQD>) </SQD>;
else
=TABLE $TABLE<SQD> (go to state </SQD>$csX<SQD>) </SQD>;
fi
if($mvX=="l")then =TABLE $TABLE<SQD> (move left) </SQD>; fi
if($mvX=="r")then =TABLE $TABLE<SQD> (move right) </SQD>; fi
fi
=TABLE $TABLE<SQD></FONT></TD><TD VALIGN="TOP">
<IMG SRC="pix/gled.gif" ALIGN="LEFT"><FONT SIZE=2></SQD>;
if($do_=="end")then =TABLE $TABLE<SQD> (end) </SQD>;
else
if($do_=="1")then
=TABLE $TABLE<SQD> (set</FONT> <TT>[X]</TT><FONT SIZE=2>) </SQD>;
fi
if($do_=="nset")then
=TABLE $TABLE<SQD> ( don't change) </SQD>;
fi
if($cs_==$i)then
=TABLE $TABLE<SQD> (remain in state </SQD>$cs_<SQD>) </SQD>;
else
=TABLE $TABLE<SQD> (go to state </SQD>$cs_<SQD>) </SQD>;
fi
if($mv_=="l")then
=TABLE $TABLE<SQD> (move left) </SQD>;
fi
if($mv_=="r")then
=TABLE $TABLE<SQD> (move right) </SQD>;
fi
fi
=TABLE $TABLE<SQD></FONT></TD>
</TR></SQD>;

<SQD>function state</SQD>$i<SQD>(i){
if(lenchk(i)==false){
window.document.images[</SQD>$LEDSTART+$n+1<SQD>].src=\
url_base+"pix/rled.gif";
if(window.document.images[i-1].src==\
url_base+"pix/1bit.gif"){
window.document.images[i-1].src=\
url_base+"pix/1bitr.gif";
window.document.images[</SQD>$LEDSTART+$n+2<SQD>\
].src=url_base+"pix/rled.gif";
delay(1);
window.document.images[</SQD>$LEDSTART+$n+2<SQD>\
].src=url_base+"pix/gled.gif";</SQD>
/* del? here: */
                if($doX=="0")then <SQD>
window.document.images[i-1].src=\
url_base+"pix/0bit.gif";</SQD>
fi
                if($doX=="nset")then <SQD>
```

```
window.document.images[i-1].src=\
url_base+"pix/1bit.gif";</SQD>
fi
<SQD>
/* move here */
</SQD>
if($mvX=="l")then
"i--;"else"i++;"fi
<SQD>
window.document.images[</SQD>$LEDSTART+$n+1<SQD>\
].src=url_base+"pix/gled.gif";
/* go to/remain in state n or end */
</SQD>
if($doX=="end")then
<SQD>stp();</SQD>
else
<SQD>state</SQD>$csX<SQD>(i);</SQD>
fi
<SQD>
}
else{</SQD>
if($do_=="1")then <SQD>
window.document.images[i-1].src=\
url_base+"pix/1bitr.gif";</SQD>
else <SQD>
window.document.images[i-1].src=\
url_base+"pix/0bitr.gif";</SQD>
fi
<SQD>
window.document.images[</SQD>$LEDSTART+$n+3<SQD>\
].src=url_base+"pix/rled.gif";
delay(1);
window.document.images[</SQD>$LEDSTART+$n+3<SQD>\
].src=url_base+"pix/gled.gif";</SQD>
/* write? here: */
if($do_=="1")then <SQD>
window.document.images[i-1].src=\
url_base+"pix/1bit.gif";</SQD>
fi
                if($do_=="nset")then <SQD>
window.document.images[i-1].src=\
url_base+"pix/0bit.gif";</SQD>
fi
<SQD>
/* move here */
</SQD>
if($mv_=="l")then
"i--;"else"i++;"fi
<SQD>
window.document.images[</SQD>$LEDSTART+$n+1<SQD>\
].src=url_base+"pix/gled.gif";
```

```
/* go to/remain in state n or end */
</SQD>
if($do_=="end")then
<SQD>stp();</SQD>
else
<SQD>state</SQD>$cs_<SQD>(i);</SQD>
fi
<SQD>
}
}
}
</SQD>
=TABLE $TABLE"
";

done

<SQD>
//-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#CCCCCC" TEXT="#333333" LINK="#333333" \
ALINK="#333333" VLINK="#333333">
<CENTER>
<TABLE HEIGHT="100%"><TR><TD>
<CENTER>
<HR SIZE=1 WIDTH="100%">
<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=0>
<TR>
</SQD>

=j $LEDSTART+1;
for(i,1,$j)do
    <SQD><TD ALIGN="CENTER"><FONT SIZE=1 COLOR="#</SQD>
    if($i>$AUX) then
=h $AUX+$CELLS;
if($i>$h)then"666666"else"CC0000"fi
    else
    "666666"
    fi
<SQD>"></SQD>$STARTADDRESS+$i<SQD></FONT></TD>
</SQD>
done
<SQD></TR>
<TR>
</SQD>

for(i,1,$j)do
    <SQD><TD ALIGN="CENTER"></SQD>
    if($i>$AUX) then
=h $AUX+$CELLS;
```

```
if($i>$h)then
<SQD><IMG SRC="pix/0bit.gif"></SQD>
else
<SQD><A HREF="javascript:sele(</SQD>$i-$AUX<SQD>);" \
onMouseOver="window.status='';return true">
<IMG SRC="pix/0bit.gif" BORDER=0></A></SQD>
fi
    else
    <SQD><IMG SRC="pix/0bit.gif"></SQD>
    fi
<SQD></TD>

</SQD>
done

<SQD>
</TR>
</TABLE>
<HR SIZE=1 WIDTH="100%">
<FONT SIZE=2>...section of (ideally endless) magnetic tape,
divided into cells...</FONT>
<P>
<TABLE BORDER CELLSPACING=0 CELLPADDING=5>
</SQD>
$TABLE
<SQD></TABLE>
<FORM>
<INPUT TYPE="BUTTON" onClick="machine()" VALUE=" Start ">\
  
<INPUT TYPE="BUTTON" onClick="res()" VALUE="Reset">\
  
<INPUT TYPE="BUTTON" onClick="window.back()" VALUE="Debug">\
  
<INPUT TYPE="BUTTON" onClick="document.location.href=\
document.location.href" VALUE="Restart">
</FORM>
</CENTER>
</TD></TR></TABLE>
</CENTER>
</BODY>
</HTML>
</SQD>
;

!main(void)
$_cgiparse($_read())
if($_cgival("RUNLEVEL")=="")then
$runlevel1()
fi

if($_cgival("RUNLEVEL")=="1")then
```

```
$runlevel2($_cgival("CELLS"),$_cgival("STATES"),$_cgival("AUX"))
fi

if($_cgival("RUNLEVEL")=="2")then
$machine($_cgival("CELLS"),$_cgival("STATES"),$_cgival("AUX"))
fi
;

$main().
```

4.13. Kollaborative und kummulative Produkt-Bewertung

Name: Kollaborative und kummulative Produkt–Bewertung	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich eingesetzt
Autor(en): Urs Hirschberg	**Zielgruppe:** Autorensystem–Designer, Online–Lehrende
Problem: Die Bewertung studentischer Designprodukte erfolgt üblicherweise durch Lehrende in Form von Schulnoten. Die von Lernergruppen erzeugten Mengen an (Zwischen–)Produkten überschreitet dabei sehr leicht die den Lehrenden für die Leistungsbewertung zur Verfügung stehende Zeit. Beurteilungen von Designprodukten durch Einzelpersonen helfen Lernenden durch ihre Deutlichkeit, sind aber oft weniger repräsentativ (und somit weniger fair) als Beurteilungen durch Gruppen. Beurteilungen durch Gruppen hingegen sind repräsentativer und fairer als Einzelbeurteilungen, verlieren aber an Aussagekraft.	
Kurzdarstellung der Lösung: Jeder (lehrende und lernende) Teilnehmer eines produktorientierten Kurses darf jedes Produkt des Kurses mit einem der Werte '-', '=' oder '+' beurteilen. Die Summen der gesammelten negativen, neutralen und positiven Urteile werden nicht weiter verarbeitet, um in anderen Formaten dargestellt zu werden sondern als solche direkt dargestellt.	
Gegenanzeigen: Da die bisherigen Anwendungen dieses Musters ausschließlich auf die Bewertung von Designprodukten zielten, sollten Bewertungen von Design–Prozessen mit besonderer Vorsicht erfolgen. Die Einbeziehung von Studenten in die Bewertung produktorientierter Leistungen erscheint weit weniger problematisch, da sie ad hoc, auch ohne detailliertes Verständnis besonderer didaktischer Zielsetzungen erfolgen kann, während die Prozeßbewertung tendenziell ein tiefergehendes Wissen über jeweilige Lernziele und formale Schwerpunkte erfordert.	
Besondere Hilfsmittel: keine	
Referenzen: Online–Beispiele erfolgreicher Anwendungen dieses Musters finden sich unter: `http://space.arch.ethz.ch/ss99` und `http://space.arch.ethz.ch/ss98`	
Verwandte Muster: siehe auch „Generisches Format zur Leistungsbewertung“	
Anmerkungen: –	

Diskussion

Leistungsbewertung wird in gestalterischen Kontexten dadurch erschwert, daß Urteile eher subjektiv und damit um so fairer sind, je mehr Parteien an der Bewertung teilnehmen. Urteile und die ihnen zugrundeliegenden Gesichtspunkte weichen jedoch oftmals stark voneinander ab und erlauben nur bedingte Vergleichbarkeit. Dieser Effekt verstärkt sich um so mehr, je weniger die Beurteilenden mit der Entwurfssituation vertraut sind. Darüber hinaus führt Durchschnittsbildung bei der Bewertung durch mehrere Beurteilende zu mittelmäßigen, wenig varianten und somit wenig aussagekräftigen Gesamturteilen. Das mit diesem Muster vorgeschlagene Bewertungsmodell verzichtet völlig auf Schulnoten, auf numerische Urteile und damit letztlich auf arithmetische Mittelung. Stattdessen werden negative, neutrale und positive Urteile für jedes (Zwischen–)Produkt gesammelt und in ihren Summen präsentiert. Jedem lehrenden und lehrnenden Mitglied des Kurses steht es frei, jedes Produkt zu bewerten, hat aber für jedes nur eine einzige, revidierbare Stimme.

Beispiel

Dieses Muster wurde in unterschiedlichen CAAD–Einführungskursen an der Architekturabteilung der ETH Zürich eingesetzt. In den Kursen wurden die Grundprinzipien und Möglichkeiten des Entwerfens mit dem Computer eingeführt — im Falle von Phase(x) (`http://space.arch.ethz.ch/ss99`) anhand von abstrakten geometrischen Kompositionen, welche phasenweise von verschiedenen Autoren weiterentwickelt wurden, im Falle von fake.space (`http://space.arch.ethz.ch/ws98`) anhand von digitalen Rekonstruktionen der Wohnungen der Studierenden, welche zu Handlungssträngen verbunden wurden. Der Bewertung durch die Studierenden, welche nach dem beschriebenen Muster erfolgte, kam in diesen Prozessen keine tragende Bedeutung zu. Da im Fach keine Noten verteilt wurden, wurde sie auch nicht zur Notengebung herangezogen.

Die Kursteilnehmer konnten die Bewertung jederzeit durch einen einzelnen Klick machen oder danach revidieren, sie geschah freiwillig und fast beiläufig.

Abbildung 4.7.: Kummulativ bewertetes Design–Produkt

Schlechte Bewertungen durch die Kommilitonen waren vor diesem Hintergrund einfacher hinzunehmen, umgekehrt wurden die Bewertungen aber auch nicht durch Höflichkeit oder Rücksichtnahme verfälscht. Trotz der beschriebenen Beiläufigkeit hatten die Bewertungen eine Art demokratisch legitimierte Objektivität und waren wertvoll als Feedback für die Studenten. Das System ergibt bei höchster Einfachheit für den einzelnen Benutzer in der Summe ein differenziertes Bewertungsbild. Mit jeder Arbeit konnten die Studenten auch eine Selbstbewertung machen und auf einer Skala von 1 bis 5 angeben, wie zufrieden sie mit dem Ergebnis ihrer Arbeit waren. Auch diese Selbstbewertung wurde für jedermann sichtbar mit der abgegebenen Arbeit dargestellt und der Betrachter konnte sich seinen eigenen Reim darauf machen. Neben dem individuellen Feedback wurden die Resultate des Bewertungssystems auch als Suchfilter verwendet (Sortieren nach Beliebtheit, etc.), um auf diese Weise die große Anzahl der Arbeiten zu sichten.

Wir haben in anderen Kursen andere Bewertungsmechanismen verwendet, zum Teil differenziertere, zum Teil mehr auf Gruppenbewertung mit Diskussionen abzielende, zuletzt auch solche, bei denen die Bewertungsresultate wie eine Art Währung, ein Spielgeld, verwendet wurden. Eine Gegenüberstellung und Bewertung all dieser Ansätze ist hier nicht möglich und auch nicht sinnvoll. Es hat sich aber gezeigt, daß die Einfachheit und die leichte Nachvollziehbarkeit der Resultate, die wir bei diesem Muster erreicht haben, sehr vorteilhaft ist und auch bei den Studenten gut angekommen ist.

4.14. Produkt–Evolution durch wechselnde Autorenschaft

Name: Evolutionäre Produkt–Entwicklung durch wechselnde Autorenschaft	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erfolgreich eingesetzt
Autor(en): Cristiano Ceccato	**Zielgruppe:** Autorensystem–Designer, Online–Lehrende, Offline–Lehrende
Problem: Die Entwicklung von Design–Produkten in Gruppensituationen erfordert die Erfassung aller durch die Gruppenmitglieder produzierten einzelnen Ideen. Ansätze mit unerkanntem Potential gehen oft durch einen in der Gruppe entstehenden „Designstrom“ verloren. Computergestütztes evolutionäres Design empfiehlt sich hier als effektive Hilfe, ist jedoch an sich ein komplexer Gegenstand, der sowohl den jeweils beteiligten Designern als auch Design–Studenten vermittelt werden muß.	
Kurzdarstellung der Lösung: Ein dezentralisiertes, evolutionäres Designsystem ermöglicht die Darstellung einzelner Entwurfskomponenten als „genetisches Erbgut“, welches als „Gen-Pool“ der gesamten Gruppe zur Verfügung steht. Jeder Designer trägt mit seinen Produkten zu diesem Pool bei; das System kann dann durch die Evaluation formaler Anforderungen sowie durch (menschliche) ästhetische Bewertungen erfolgreiche Design–Ideen identifizieren und den Entwerfern wiederum als Erbgut für eine neue Entwurfs–Generation anbieten. Es entsteht ein nicht–deterministisches Designsystem, welches die beteiligten Entwerfer einerseits in ihrer Beurteilung von Designvarianten unterstützt und ihnen andererseits zu einem angereicherten Ideen–Pool verhilft. Eine Anwendung dieser Strategie im Design–Unterricht dient sowohl der Förderung erzielter Designqualitäten als auch der didaktischen Vermittlung der evolutionären Designstrategie.	
Gegenanzeigen: Das angestrebte Modell stellt ein dezentralisiertes und gleichberechtigtes, letztlich also ein besonders demokratisches Entwurfssystem dar, innerhalb dessen jede Design–Idee das Potential hat, zur Geltung zu kommen. Dies ist jedoch nicht in jeder Entwurfssituation wünschenswert und mitunter sogar kontraproduktiv. Oft müssen Designprobleme unter dem Druck realer Bedingungen gelöst werden, die eine zeitaufwendige Entwicklungsphase nicht zulassen. In diesen Fällen ist die Leitung des Entwurfsprozesses bzw. die Führung der Design–Gruppe zu bevorzugen, um auf direktestem Weg eine Entwurfslösung zu erstellen. Dieses Probelm sollte bei didaktischen Anwendungen dieses Verfahrens explizit thematisiert werden.	

Besondere Hilfsmittel: Groupware für (Meta–)Kommunikation von Designideen und ggf. Software für maschinelle Evaluation der „requirement and constraint satisfaction“.
Referenzen: Die Entwicklung dieses Musters wurde inspiriert durch John Frazers Grundlagenforschung und seine zahlreichen Anwendungen auf dem Gebiet des Evolutionären Designs und der Evolutionären Architektur[14].
Verwandte Muster: Eine maschinelle Selektion auf Basis menschlicher Urteile kann erreicht werden, indem dieses Muster mit dem Muster „kummulative und kollaborative Produkt-Bewertung“ kombiniert wird.
Anmerkungen: Das hier dargestellte evolutionäre Entwurfsmodell kann durch computer– und netzwerkbasierte Hilfsmittel unterstüzt und weiterentwickelt werden. Hierzu ist 1999 an der Hong Kong Polytechnic University ein Forschungsprojekt begonnen worden, das mittels verteilter Informationstechnologien und evolutionärer Suchmethoden das hier dargestellte System implementiert. Eine erfolgreiche Anwendung dieses Musters fand in *Group Three* auf der Workshop–basierten Konferenz `allomorphe` im Juni 1999 an der School of Architecture and Building an der Deakin University in Australien statt.

Diskussion

Die Praxis der heutigen Architektur– und Industriedesignpraxis erfordert höchst komplexe Entwurfs– und Realisationsprozesse, innerhalb derer sich Erfolg nur durch das Zusammenspiel gestalterischer, funktionaler, konstruktiver, materieller, herstellerischer und wirtschaftlicher Prozeßelemente einstellt. Entwerfen ist in diesem Kontext einen nicht–linearer Prozeß, innerhalb dessen die Abhängigkeit einzelner Variablen zueinander nicht eindeutig beschreibbar ist.

Dennoch ist es zwingend erforderlich, möglichst dem Ideal enstprechende Entwurfslösungen zu finden, die schließlich erfolgreiche Designs ergeben. In diesem Sinne ist es sehr wichtig, die verschiedenen Entwurfsideen, die sich in Gruppensituationen herausstellen, sowohl individuell als auch gegeneinander zu bewerten und im weiteren Entwurfsprozeß entsprechend dieser Beurteilung einzubinden.

Das Kernproblem hierbei ist die Bewertung anhand verschiedener Kriterien; die intuitive Auslese von Entwurfselementen durch die Designer selbst ist dabei eine genauso wichtige und wirkungsvolle Strategie wie die automatisierte Auswahl anhand konkreter Leistungsdaten.

[14] siehe Frazer, John: [19]

Beispiel

Zur (didaktischen) Implementierung dieses Musters kann die Zeit, die für die Produktentwicklung zur Verfügung steht, in ein Raster aufgeteilt werden, dessen einzelne Abschnitte so dimensioniert werden, daß sie jeweils sinnvoll die Entwicklung und Ausformulierung von Entwurfsideen erlauben. Die Designer–Gruppe (bzw. die Lernergruppe) wird in Kleingruppen oder individuelle Entwerfer aufgeteilt, die während jedes Zeitabschnitts Produkte entwickeln. Die während des ersten Abschnitts getroffenen Design–Entscheidungen sind relativ frei und sollten lediglich eine sinnvolle Beziehung zu der zu Beginn des Prozesses definierten Problemstellung aufbauen. Am Ende dieses und der folgenden Abschnitte werden die erzeugten Produkte an einem für die gesamte Gruppe zugänglichen Ort abgelegt. Die Arbeit in jedem weiteren Abschnitt darf nicht von Grund auf neu beginnen. Die Entwerfer müssen zu Beginn jedes weiteren Abschnitts von ihnen bevorzugte Inhalte (die nicht ihre eigenen sind!) aus der Produktsammlung auswählen und als „Ausgangsmaterial“ ihrer weiteren Arbeit in dem jeweiligen Abschnitt verwenden. Während des nächsten Abschnitts verlieren sie jedoch wieder die Möglichkeit, an ihrem alten Produkt zu arbeiten. Sie müssen als Ausgangsmaterial wieder ein Produkt einer anderen Gruppe (bzw. eines anderen Individuums) wählen und können nur versuchen, die Weiterexistenz ihres Produkts, das sie an die Gruppe verlieren, durch hohe Designqualität zu sichern.

Das Resultat ist ein evolutionärer Designprozeß: Bessere Ideen und Konzepte setzten sich durch, während andere, weniger erfolgreiche „aussterben“. Die kreative Arbeit der beteiligten Designer entspricht der natürlichen Variation durch Mutation und das „Survival of the Fittest“ erfolgt ähnlich dem natürlichen Vorbild durch das Anlegen von Selektionskriterien. Diese Kriterien werden in Designprozessen wie dem oben beschriebenen durch menschliche Auswahl implementiert. Dort, wo Selektionskriterien formalisierbar und maschinell feststellbar sind, kann die Auswahl auch durch Computer unterstützt werden. Um in didaktischen Kontexten ein Verständnis des evolutionären Prinzips zu vermitteln, erscheint die bewußte Auswahl durch die Entwerfer (Lerner) selbst besser geeignet als transparente maschinelle Selektion.

Eine Plattform zur didaktischen Implementierung der oben diskutierten Strategie (basierend auf menschlicher Selektion) kann im einfachsten Fall durch gemeinsame Nutzung eines FTP–Servers erfolgen, auf dem für jeden Prozeß–Abschnitt ein eigenes Verzeichnis angelegt wird, in denen die Entwerfer(–gruppen) ihre Produkte unter ihrem jeweiligen Namen ablegen. Es sind auch komfortablere Varianten denkbar, in denen der FTP–Server über die CGI–Schnittstelle eines WWW–Servers durch eine nutzerfreundlichere Oberfläche bedient wird oder in denen mit redaktioneller Betreuung und Nutzung von GNU-Tools `RCS` oder `diff` und `merge` eine Versionsverwaltung mit File–Locking bereitgestellt wird.

4.15. Interaktion als Spielbaum

Name: Interaktion als Spielbaum	**Letzte Änderung:** 5. Januar 2001
Synonyme: Interaction as Game Tree	**Status:** erfolgreich eingesetzt
Autor(en): Robert Woodbury, übersetzt und in Pattern–Format übertragen von Thomas Fischer	**Zielgruppe:** Autorensystem–Designer, Online–Lehrende
Problem: Das Austeilen traditioneller Arbeitsaufträge erzeugt Interaktionsmuster mit einer einzigen oder wenigen Reaktionen auf ein Arbeitsergebnis (typischerweise der des Lehrenden und ggf. einiger Lerner). In Unterrichtssituationen, die auf das Erlernen kommunikativer Praxis, gestalterischer Verhandlung und kreativer Beurteilung, also auf ein Maximum an Interaktion abzielt, ist dieses geringe Maß an Kommunikation ungenügend. Weiterhin stellt sich am Ende von Unterrichtseinheiten regelmaeßig das Problem, daß Archive angefertigter Produkte und erfolgter Interaktion zum Zweck der Publikation aufwendig überarbeitet und (neu) strukturiert werden müssen.	
Kurzdarstellung der Lösung: Dieses Muster wendet metaphorisch das Konzept „Galerie“ auf die Organisation unterrichtlicher Interaktion an. Kursteilnehmer (Lehrer und Lerner nehmen unterschiedliche Rollen an „Kurator“, „Entwerfer“, „Gast“) oder bekommen diese zugewiesen. Diese Rollen werden bei der dynamischen Strukturierung von Kurs–Visualisierungen als Ordungskriterium herangezogen. Sämtliche Exponate (Multimedia–Dateien) werden von allen Kursteilnehmern (auch Lernern) kommentiert. Diese Kommentare können ebenfalls in beliebigen multimedialen Formaten erfolgen. Da auf diese Weise potentiell jeder mit jedem über jedes Produkt einen Dialog führt, aus dem wieder und wieder neue Produkte hervorgehen, nimmt die Menge stattfindender Interaktion exponentiell zu.	
Gegenanzeigen: –	
Besondere Hilfsmittel: Attachment–fähiger News–Server oder äquivalentes Programm für einfache Implementierungen . Besser geeignet sind WWW/CGI–basierte Systeme, die die anfallenden Informationen nach wählbaren Kriterien (Namen der Autoren, Kuratoren, Alter der Beiträge etc.) sortieren und visualisieren.	
Referenzen: Ein Online–Beispiel einer erfolgreichen Anwendung dieses Musters findet sich unter: `http://online.adelaide.edu.au/vGallery/`	
Verwandte Muster: Dieses Muster scheint besonders für eine Kombination mit dem Muster „Kollaborative und kummulative Produktbewertung“ geeignet. Die Authentifizierung und Identifikation der Nutzer–Rollen kann mit dem Muster „Dynamische Erstattung von Zugriffsrechten“ (ggf. mit zusätzlichen Ausnahmebehandlungen für besondere Rollen) erfolgen.	
Anmerkungen: –	

Diskussion

Interaktionsmuster zwischen Lernern und Lehrern (in allen denkbaren Kombinationen) sind potentiell vielschichtiger und wesentlich komplexer als es traditionelle Unterrichtsformen gestatten. Die Vorbereitung und Distribution von Aufgaben, deren Bearbeitung durch Lerner sowie die Bewertung und die Kommunikation über Arbeitsergebnisse ist im normalen Seminar–Unterricht starken zeitlichen Einschränkungen unterworfen und im Wesentlichen ein linearer Prozeß. Wird die unterrichtliche Kommunikation (wie in diesem Muster vorgeschlagen) als Spielbaum aufgefaßt, können baumartig verzweigte Interaktionsmuster entstehen, die nicht mehr an eine lineare zeitliche Abfolge gebunden sind. Parallele Kommunikationsprozesse entstehen und der Austausch über Lerner–Produkte steigt im Vergleich zum normalen Studio–Unterricht stark an. Als Folge wächst sowohl die Quantität als auch die Qualität unterrichtlicher Dialoge und Lerner erarbeiten nicht nur Lösungsstrategien und Produkte sondern trainieren weiterhin auch ihre Fähigkeiten in den Bereichen Dialog, Verhandlung und Kritik.

Im Unterschied zur klassischen Bulletin-Board–Diskussion (wie sie etwa in Newsgroups stattfindet) steht hier im Vordergrund der Kommunikation der Kursteilnehmer nicht nur der Austausch von Textbeiträgen, mindestens ebenso wichtig ist die Kommunikation von Multimedia-Dateien. Ausgangspunkt ist eine Aufgabenstellung, die als „Objekt“ den Spielbaum initialisiert. Dieses Objekt kann aus mehreren Dateien bestehen und von jedem Kursteilnehmer gemäß der gestellten Aufgabe mit anderen/modifizierten Objekten beantwortet werden. Neben der Bereitstellung von Multimedia-Inhalten kann das ursprüngliche Objekt auch Vorgaben zur Strukturierung des Kommunikationsprozesses enthalten. Wichtig ist hierbei, daß die Teilnehmer des Kurses zuvor durch Einteilung von Gruppen und die Zuweisung von Rollen ihrerseits mit „Objekteigenschaften“ ausgestattet werden, durch die ihre Aufgaben, Rechte und Pflichten definiert werden aber auch unterschiedliche Gesichtspunkte für die Kurs–Visualisierung ermöglicht werden.

Die Darstellung der Interaktion und die Zugriffs–Möglichkeiten auf entstandene

Materialien können beliebigen Gesichtspunkten folgen. So kann der Spielbaum als tatsächlicher Baum (wie ein Newsgroup–Thread), zeitachsenorientiert, die neuesten Produkte fokussierend oder als komplettes Archiv visualisiert werden oder auch die Rollen der Lerner als Strukturierungs–Element herangezogen werden. Der nach unterschiedlichen Gesichtspunkten strukturierte Spielbaum ist zugleich visuell interpretiertes Unterrichtsprotokoll, Navigationshilfe, FTP–Speicher, Interaktionsstruktur, Kommunikationsplattform und sein eigenes Archiv, das nach Kursabschluß auch Basis für die einfache Herstellung eines CD-ROM–Archivs sein kann.

Beispiel

Die Spielbaum–Metapher wurde und wird an der School of Architecture, Landscape Planning and Urban Design an der Adelaide University wiederholt fortlaufend erfolgreich eingesetzt. Der aktuelle Spielbaum kann unter dem oben genannten URL besucht werden. Die in diesem Fall angewandte Metapher ist die Galerie: Lerner, Lehrer und Besucher erhalten im Rahmen von Entwurfsprojekten die Rollen von Entwerfern, Kuratoren, Kritikern oder Betrachtern (Lesern). Dabei können einzelne Kursteilnehmer jeweils nur mit den Möglichkeiten interagieren, die ihre Rolle vorsieht — so sind anonyme Besucher beispielsweise nicht schreibberechtigt.

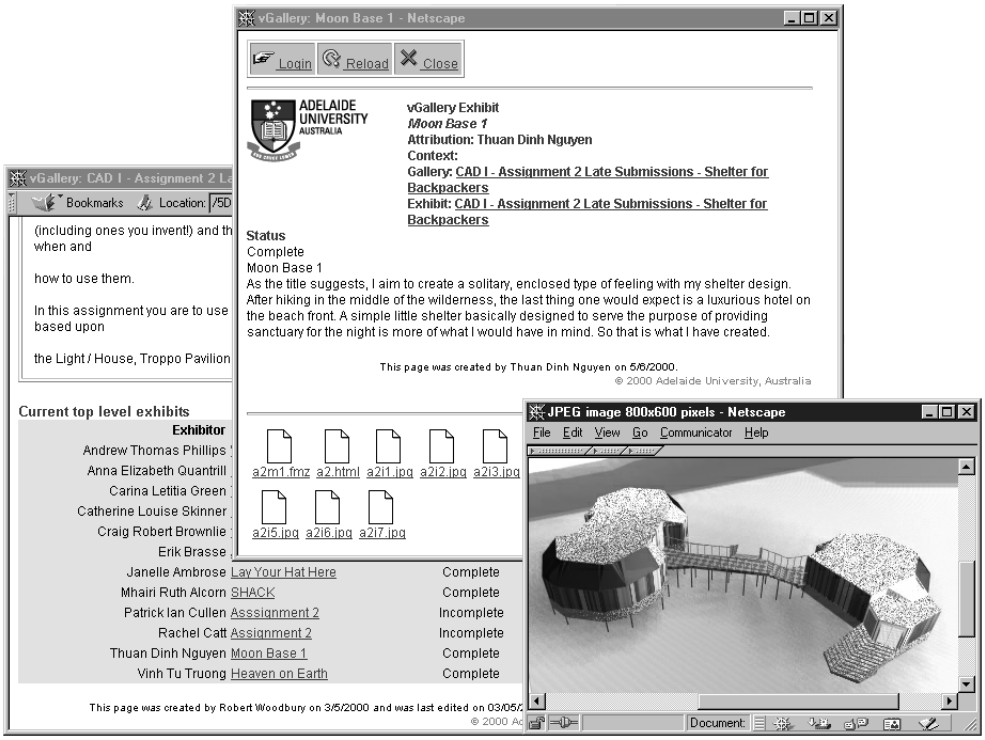


Abbildung 4.8.: Ein Exponat in einer exponentiell wachsenden Galerie

Die Verteilung der Rollen ist nicht an reale Rollen und Hierarchien gebunden: Lehrer können die Rolle eines Kritikers übernehmen oder nur die eines Betrachters erhalten; Lerner können als Entwerfer, aber auch als Kuratoren oder als Kritiker agieren. Die sich hierbei ergebenden Dialogformen fördern seit mehreren Semestern nicht nur die individuelle Aktivität und Kommunikationspraxis der Studenten, sondern durch die Übersichtlichkeit von Lernprozessen und Produkten für den Lehrenden und somit auch die Bewertung von Beiträgen.

4.16. Integration existierender Werkzeuge

Name: Integration existierender Werkzeuge	**Letzte Änderung:** 5. Januar 2001
Synonyme: –	**Status:** erste Anwendung im Entwicklungsstadium
Autor(en): Thomas Fischer	**Zielgruppe:** Autorensystem–Designer, Online–Lehrende
Problem: Autorensysteme verhalten sich unflexibel gegenüber individuellen, kontextorientierten methodischen Intentionen Lehrender. Als Konsequenz empfiehlt diese Musterprache kontextorientierten Entwurf und Implementierung von Unterrichtswerkzeugen durch Lehrende. Dies bringt neben didaktisch/methodischen Vorteilen auch Nachteile mit sich, weil es die ohnehin starke Arbeitsbelastung und Verantwortung Lehrender weiter erhöht. Weiterhin führt die Wahl, didaktische Werkzeug–Entwicklung *entweder* traditionellen Software–Entwicklern *oder* Lehrern zu überlassen zu dem Risiko, daß die letztere Strategie die Fehler der traditionellen Vorgehensweise wiederholt: Daß kommerzielle Autorensysteme die zusätzliche Integration von Individuallösungen nicht unterstützen und mitunter sogar ausschließen, darf im Umkehrschluß nicht dazu führen, daß von Lehrern entwickelte Werkzeuge die Integration kommerzieller Standardlösungen ausschließen.	
Kurzdarstellung der Lösung: Dank der technischen Eigenschaften von Hypertext und insbesondere der Architektur des WWW ist es möglich, einzelne Problemlösungen kommerzieller, webbasierter Lehrwerkzeuge zu isolieren und in individuell entwickelte Online–Lernumgebungen zu integrieren. Falls Elemente kommerzieller Werkzeuge in der Form, in der sie geliefert werden in einer Lernumgebung gewünscht werden, bietet ihre Integration in eine von Lehrern entwickelte Lernumgebung eine gute Möglichkeit zur Reduzierung des Entwicklungsaufwands.	
Gegenanzeigen: Kommerzielle Pakete verhalten sich oft wenig kooperativbei der Integration von/in Ressourcen, die nicht Teil desselben Pakets sind oder nicht vom selben Hersteller stammen. Es macht in diesen Fällen wenig Sinn, zuviel Energie in das *hacken* eines kommerziellen Systems zu investieren, wenn dieselbe Energie zur Bereitstellung einer (ggf. wiederverwendbaren und sich dabei „kooperativ“ verhaltenden) eigenen Lösung genutzt werden kann.	
Besondere Hilfsmittel: –	
Referenzen: –	
Verwandte Muster: –	
Anmerkungen: –	

Diskussion

Die Motivationen für die Integration existierender Werkzeuge können sehr unterschiedlich sein. Die in der obigen Problembeschreibung genannten didaktischen und entwicklungsökonomischen Gründe sind nur zwei davon. Es ist ebenso denkbar, daß eine bereits existierende Implementierung einer Lösung einen sehr hohen Stellenwert als Standardwerkzeug hat, dessen Nutzung am besten am Original selbst trainiert wird. Ein anderer Grund kann darin bestehen, daß nach den erforderlichen immensen Investitionen in Erwerb, Installation, Training und Instandhaltung für ein Produkt wie WebCT, dessen Anwendung sehr bald zu einem Hochschulpolitikum avanciert.

Da WebCT als praktisch einziges Produkt seiner Art und daher als Marktführer de facto zu einem Synonym für Modewörter wie „Cyber University“ und damit aus Sicht der Öffentlichkeit zu einem wichtigen Kriterium für unterrichtliche Qualität an zukunftsweisenden Bildungseinrichtungen geworden ist, können es sich Universitäten heute kaum leisten, WebCT *nicht* zu kaufen. Jeder Online–Kurs, der nach der Lizenzierung von WebCT an den jeweiligen Universitäten stattfindet ohne dieses Werkzeug zu nutzen, macht sich aus wirtschaftlicher Sicht direkt angreifbar: Steuergelder sind investiert worden und von der Investition wird kein Gebrauch gemacht (in der Regel sind es übrigens nicht die Lehrenden, die sich um den Erwerb von WebCT bemühen, sondern diejenigen, die das Lehren verwalten). Die Einschätzung, daß der Einsatz von WebCT in vielen unterrichtlichen Situationen kontraproduktiv ist und letztlich der Grundidee von (nicht nur unterrichtlichem) Entwerfen widerspricht, und an sich eine Verschwendung von Ressourcen sowie eine grundsätzliche Infragestellung unserer lehrerbasierten Dezentralisierung didaktischer Methodenwahl darstellt, wiegt dagegen ohne Einblick in die Strukturen und Potentiale von Autorensystemen nicht viel.

Beispiel

Dieses Beispiel dokumentiert die Integration eines der Kommunikations–Werkzeuge von WebCT (Whiteboard) in eine eigene Lernumgebung. Die *Sil-*

kroad Interactive Encyclopedia ist eine im Jahr 1999 für einen integrierten Design–Kurs (DS6) an der School of Design (Hong Hong Polytechnic University) hergestellte Onine–Lernumgebung. Sie dient als datenbankgestützte Plattform zur kollaborativen Sammlung von Resultaten studentischer Studien und Recherchen. Sie umfaßt unter anderem eine Volltext–Suchmaschine sowie eine dynamische zweidimensionale Echtzeit–Visualisierung der Datenbank–Inhalte als Hilfe zur Navigation in sehr umfangreichen Inhalten. Weiterhin umfaßt diese Lernumgebung ein Online–Benotungssystem, das die Benotung von mehr als 100 Lernenden durch sechs Lehrende in fünf (unterschiedlich stark gewichteten) „Meilensteinen“ mit jeweils fünf Teilnoten erlaubt, wobei Lernende jederzeit Lesezugriff auf ihre bereits erhaltenen Teilnoten erhalten und bewertungsrelevante Beiträge zur Online–Enzyklopädie einzelner Lerner für benotende Lehrende transparent in die Benotungs–Eingabemaske eingebunden und dargestellt werden.

Keines dieser einzigartigen Gestaltungsmerkmale[15] wäre mit WebCT oder einem andern der gängigen Werkzeuge für Online–Unterricht (Toolbook Instructor, Macromedia Authorware etc.) herstellbar gewesen. Dennoch soll nun, da der Kurs nach einem Jahr wieder angeboten wird, von der existierenden Installation von WebCT Gebrauch gemacht werden, was aus wirtschaftlicher Sicht völlig verständlich, aus methodischer Sicht jedoch kaum akzeptabel ist.

Frühe Planungsgespräche kamen zu dem Ergebnis, daß die Funktionalität der im Vorjahr erstellten Lernumgebung weiterhin benötigt wird, diese mittels WebCT jedoch nicht hergestellt werden kann. Andererseits verfügt WebCT über ein Whiteboard, das für die kollaborative Entwicklung visueller Inhalte für die Enzyklopädie sehr interessant ist. Es empfiehlt sich also eine Integration des in WebCT enthaltenen Whiteboards in eine neue Installation der *Interactive Encyclopedia* (siehe Entwurf in Abbildung 4.9).

Aufgrund der Tatsache, daß WebCT und die *Interactive Encyclopedia* auf der selben Hypertext–Technologie HTML basieren und dank des wenig überzeugenden Sicherheitskonzepts in WebCT ist es sehr einfach, diese Integration zu

[15] Dieses System umfaßt noch eine Reihe weiterer neuartiger Konzepte, wie z.B. das unter 4.10 beschriebene.

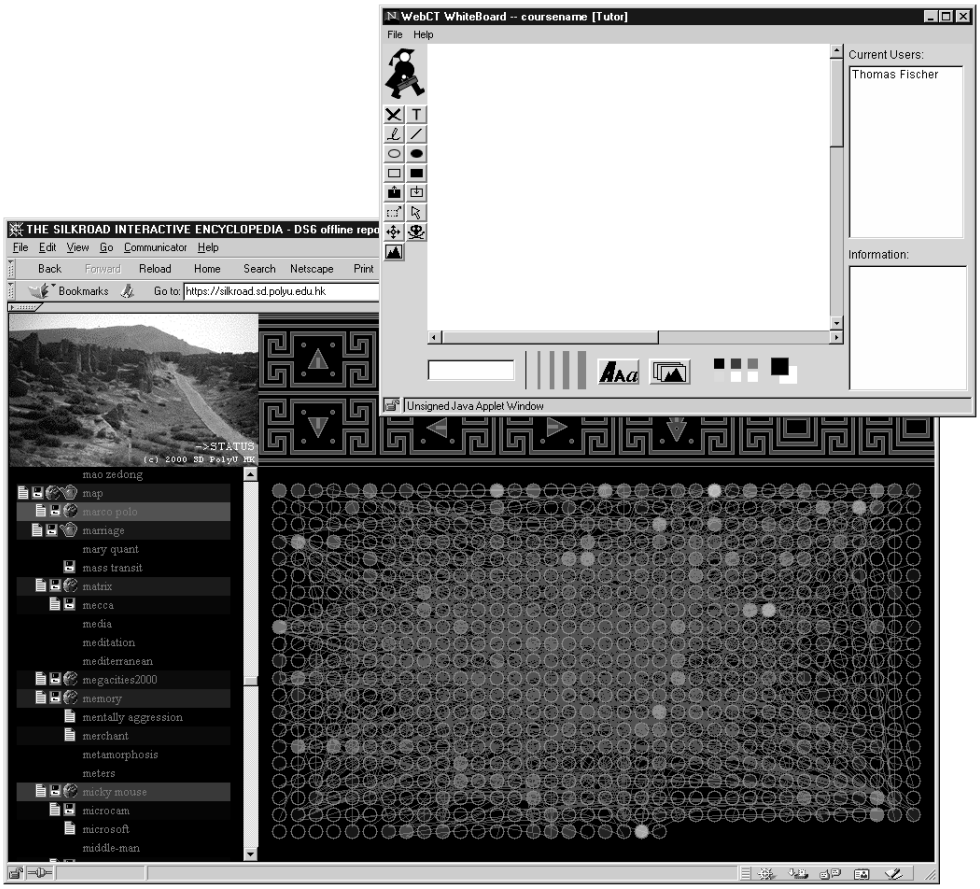


Abbildung 4.9.: Integration des WebCT–Whiteboards in eine problemorientierte Lernumgebung (*Fotomontage*)

erreichen.

```
<APPLET CODEBASE="http://127.0.0.1:8901/web-ct/code/whiteboard"
CODE="Whiteboard.class" WIDTH=200 HEIGHT=100 MAYSCRIPT>
<PARAM NAME="userID" value="Thomas Fischer">
<PARAM NAME="userName" value="Tutor">
<PARAM NAME="courseID" value="coursename">
<PARAM NAME="lang" value="0">
<PARAM NAME="server_OS" value="Unix">
<PARAM NAME="udpPort" value="4577">
<PARAM NAME="tcpPort" value="4578">
</APPLET>
```

Es ist lediglich erforderlich, über die CGI–Schnittstelle des WWW–Servers, von dem die *Interactive Encyclopedia* serviert wird, ein HTML–Dokument zu gene-

rieren, das den oben dargestellen Code enthält. Der für den enthaltenen Parameter `userID` erforderliche Name kann einer Datenbank entnommen werden, die den durch die Zugriffs–Authentifizierung bekannten Login–Namen des jeweiligen Nutzers aus der Umgebungsvariable `REMOTE_USER` liest und in dessen vollen Namen übersetzt. Das dargestellte Code–Fragment in dem generierten Dokument weist den Browser des Nutzers an, das als Java–Applet implementierte WebCT–Whiteboard zu laden und in einem neuen Fenster zu öffenen.

5. Ausblick

Ich habe in dieser Arbeit dargestellt, daß die Geschichte des computervermittelten Unterrichts, der hierfür angebotenen Werkzeuge und somit auch die Geschichte von Autorensystemen sowohl eng als auch vielschichtig mit der Geschichte, den wissenschaftlichen Strömungen und Paradigmen der vergangenen 50 Jahre verknüpft sind. Die Charakteristika von digitalen Unterrichts–Werkzeugen haben sich dabei eher an den jeweiligen technischen Möglichkeiten und wissenschaftlichen Leitbildern orientiert als an dem Ziel, den individuellen Lehrenden als kompetenten und verantwortlichen Experten in seinen Entscheidungen zu unterstützen. In diesem Zusammenhang sind beispielsweise die *Kybernetik*, *Künstliche Intelligenz* oder der *Behaviourismus* als wichtige Einflüsse zu nennen, die jeweils nach Antworten auf Probleme des Lehrens gesucht haben, ohne dabei der Rolle des Lehrers besondere Beachtung zu schenken.

Nacheinander wurden alle wichtigen elektronischen Massenmedien ohne den erhofften Erfolg als Heilmittel für den Mangel an Massenbildung in Erwägung gezogen. Der Computer macht hier bislang keine Ausnahme. Es scheint mir von besonderer Wichtigkeit, daß im Fall dieser Anwendungen von Massenmedien die Modelle didakischer Planung und Interaktion *stark zentralisiert* waren. Insbesondere in methodischer Hinsicht imitieren unsere heutigen Autorensysteme dieses zentralisierte Modell, indem sie auf der Ebene ihrer technischen Entwicklung die Strukturen unterrichtlichen Handelns vorgeben. Somit sind sie ebenso sicher vor dem Eingriff der jeweiligen Lehrer („teacherproof“) wie zentralisierte Programme im Bildungs–Radio oder –Fernsehen.

Radio und Fernsehen haben den Lehrer aus gutem Grund nicht aus dem Klas-

senzimmer verdrängt (obgleich sie sehr wohl in der Lage sind, seine Arbeit sinnvoll zu ergänzen). Daß dies auch Autorensysteme nicht schaffen werden ist bereits heute daran zu erkennen, daß Autorensysteme zwar relativ weit verbreitet und verfügbar sind, sie bei Lehrern aber auf Desinteresse und Ablehnung stoßen[1]. Seit Jahrzehnten gilt die *Nutzerfreundlichkeit* als Hürde auf dem Weg zur Nutzung von Autorensystemen. Ich behaupte jedoch auf der Basis von vielen Beobachtungen, daß sehr viele Hard– und Softwaresysteme an Hochschulen existieren und ggf. auch ungeachtet mangelnder Nutzerfreundlichkeit genutzt werden. Wenn sie benutzt werden müssen, aber nicht ausreichend nutzerfreundlich erscheinen, werden die für ihre Anwendung erforderlichen Hilfkräfte angestellt und nicht auf ihre Nutzung verzichtet, wie es im Fall von Autorensystemen aber meist der Fall ist.

Ich selbst habe eine Reihe universitärer computervermittelter Unterrichtsprojekte zunächst als Hilfskraft und auch als Lehrer begleitet und habe dabei keinen einzigen Fall erlebt, in dem Hilfskräfte für die Nutzung von Autorensystemen eingestellt wurden. Sie werden eingestellt, um eigene, an die jeweiligen konkreten didaktischen Erfordernisse angepaßte Lösungen selbst zu entwickeln. Ich betrachte dies als Indiz dafür, daß *Flexibilität* mindestens ebenso wichtig ist wie Nutzerfreundlichkeit. Flexibilität wird jedoch im Fall existierender Autorensysteme für wenig überzeugende „Nutzerfreundlichkeit“ eingetauscht. Die in diesem Zusammenhang erforderliche *visuelle Programmierung* ist für die Herstellung komplexer Software nachgewiesenermasën nur bedingt geeignet[2]. Weiterhin behindern Autorensysteme sinnvolle didaktische Entwicklungen oft durch mangelhaft dokumentierte sowie nicht interoperable Standards und Schnittstellen und ähnliche Charakteristika kommerzieller Software.

Modelle didaktischen Handelns sind, wie auch Merkmale und Funktionen von Autorensystemen *Rekonstruktionen von Unterrichtsrealität*[3]. Während diese Modelle im traditionellen Unterricht selbstverständlich nicht unreflektiert und ausschließlich auf neue Unterrichtssituationen angewendet werden, erzwingen

[1] vgl. Rode, Mary und Jim Poirot: [52]

[2] siehe Schiffer, Stefan: [54], S. 333-334

[3] vgl. Flechsig, Karl-Heinz: [18], S. 5

Autorensysteme dies bei ihrer Anwendung. Ich habe zur Lösung dieses Problems mit der vorliegenden Arbeit eine Methode entworfen, die unter besonderer Berücksichtigung der Kompetenz und der Verantwortung Lehrender die individuelle Entwicklung von Lernumgebungen und Werkzeugen für die Entwicklung von Lernumgebungen gestattet. Im Gegensatz zur üblichen Verkaufsrhetorik, mit denen Autorensysteme angepriesen werden, behaupte ich weder, daß diese Methode kinderleicht und intuitiv erlernbar ist, noch daß sie völlig ohne Programmierung möglich ist (beides ist auch im Fall von Autorensystemen nicht wahr).

Das Konzept der Mustersprache wird seit den 70er Jahren in mehreren entwerferischen und technischen Problemfeldern eingesetzt. Die hier vorgestellte Mustersprache für das Design von Autorensystemen gestattet ein hohes Maß an Flexibilität durch textuelle Programmierung, Nutzerfreundlichekeit durch die Möglichkeit der Entwicklung optionaler graphischer Schnittstellen und durch moderierte Weitergabe bewährter Lösungen sowie Produktivität durch die Wiederverwendung von Expertenwissen. Ich schlage eine Entwicklung als *Open-Source*-Projekt vor, in dem *Lehrer als Werkzeugmacher* verstanden werden und agieren und die Wahl haben, sich als Anwender, aber auch als Entwickler zu beteiligen.

Die inhaltliche Verwandtschaft der Lehre objektorientierter Programmierung (einem klassischen Anwendungsfeld für Mustersprachen) und dem computervermittelten Unterricht legt eine Anwendung des Konzepts Mustersprache auf das Autorensystem–Design nahe. Aufgrund der nötigen Offenheit von Online–Lehrwerkzeugen ist eine Entwicklung als Open-Source–Projekt ebenso naheliegend. Beides blieb bislang jedoch ebenso aus wie eine genauere Betrachtung didaktischer Mustersprachen durch die Erziehungswissenschaft. Ich hoffe, diese genannten Prozesse mit meiner Arbeit anzuregen und zu beschleunigen.

Ich habe die wissenschaftstheoretischen Gründe aufgezeigt, aus denen existierende Meta–Untersuchungen zum Unterrichten mit Computern keine nennenswerten Effekte messen[4]. Aus eben diesen Gründen kann auch ich letztlich nicht

[4]Schulmeister, Rolf: [55], S. 379 ff.

die *Wahrheit*, *Richtigkeit* oder *Effektivität* dieser Mustersprache empirisch belegen. Stattdessen kann ich jedoch auf zahlreiche interessante konkrete Anwendungen dieser Methode verweisen, von denen ich hier eine Vielzahl präsentiert und diskutiert habe.

Die bösartige Natur der Entwicklung didaktischer Problemlösungen ist der Grund dafür, daß nun keine *stopping rule* zur Verfügung steht, die den erfolgreichen Abschluß dieser Arbeit signalisieren könnte. Erkennbar ist lediglich eine Tendenz qualitativer Verbesserung und einfacherer Herstellung in der Reihe der Autorensysteme und Lernapplikationen, deren Entwicklung ich in den vergangenen Jahren begleitet habe. Doch ebenso deutlich erkennbar ist das Potential für weiteres qualitatives und quantitatives Wachstum dieser Mustersprache, denn die enthaltenen „Kochrezepte“ haben sich zwar praktisch bewährt, könnten aber inhaltlich und technisch jederzeit weiterentwickelt und verbessert werden.

Zur Zeit gibt es nur eine kleine, aber wachsende Gruppe von Lehrern, die für ihren Unterricht das WWW benutzen, die dafür erforderliche Software für jeden Kurs selbst entwickeln und anpassen sowie ihre dabei gewonnenen Erkenntnisse einander zugänglich machen. Für dieses Vorgehen fehlt bislang sowohl ein Forum als auch ein zweckorientiertes Austauschformat. Selbstverständlich eignet sich das WWW selbst ideal für die logistische Dimension dieses Vorhabens, da es eine leistungsstarke Kommunikations–Infrastruktur darstellt, zu der potentielle Nutzer bereis über Zugänge und Nutzungserfahrungen verfügen, die sich in vielen kollaborativen Entwicklungsprozessen als Diskussionsrahmen bewährt hat und die sich weiterhin ideal eignet, existierende Muster in Form downloadbarer Software–Archive zu publizieren.

Nachdem mit Abschluß dieser Arbeit ein Startvokabular für die Mustersprache vorliegt, hoffe ich, mit dessen Publikation in naher Zukunft sowie mit der Sammlung und Bereitstellung von Mustern für das Design von Autorensystemen und Lernumgebungen in einer Online–Datenbank eine Vergrößerung der Entwickler–Gemeinde und einen anhaltenden Entwicklungsprozeß anzuregen.

A. Pattern–Landkarte

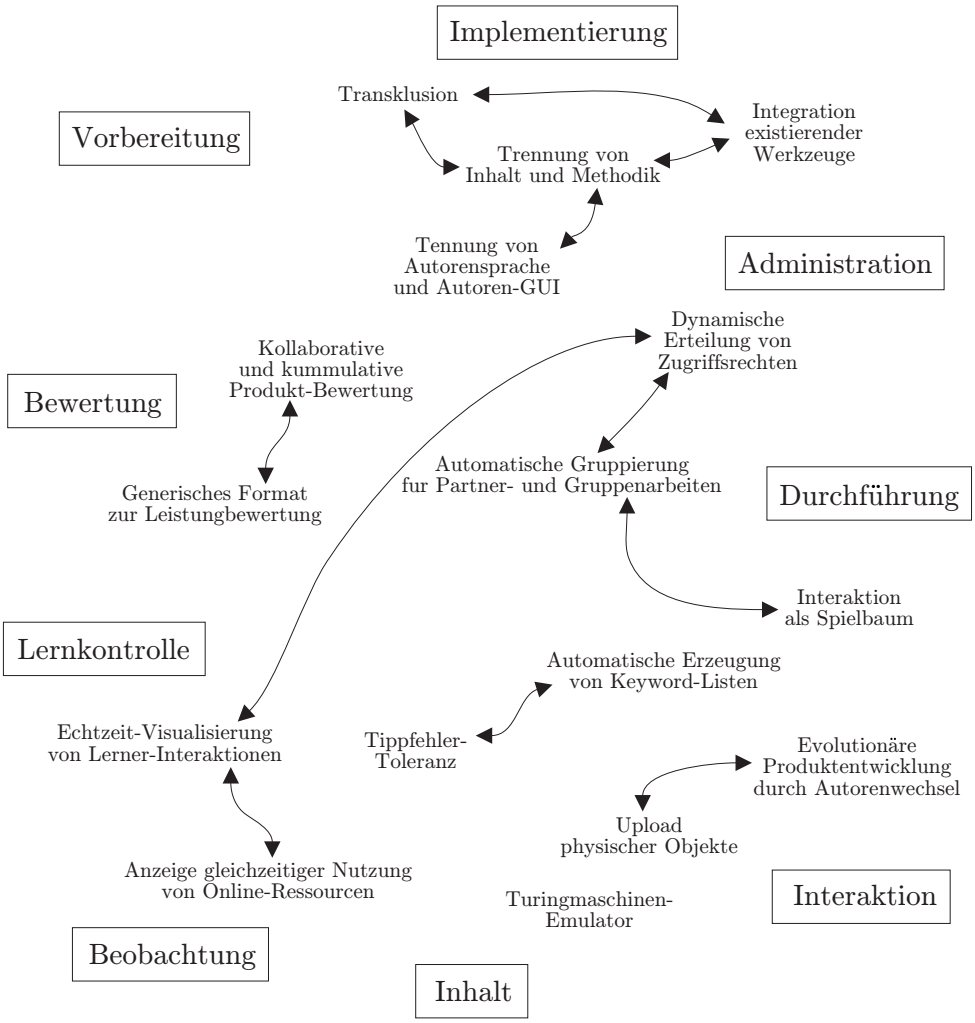


Abbildung A.1.: Pattern–Landkarte

B. Beispiel–Muster aus Christopher Alexanders Mustersprache

196 TÜREN IN DEN ECKEN

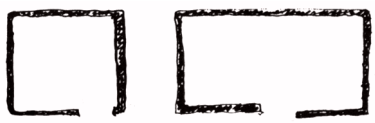
...das nachfolgende Muster hilft dabei, die genaue Lage von Türen zu bestimmen. Es kann dazu beitragen, das größere Muster VON RAUM ZU RAUM (131) zu schaffen. Es kann auch zur Ausbildung MEHRERER SITZPLÄTZE (142) beitragen, indem Ecken zum Sitzen freigelassen werden, die nicht von Türen beeinträchtigt sind; und es kann zum WECHSEL VON HELL UND DUNKEL (135) beitragen, da jede Tür, sofern sie verglast und in der Näher eines Fensters ist, eine natürliche Lustinsel bildet, welche die Menschen anzieht.

◊ ◊ ◊

Das Gelingen eines Raumes hängt zu einem großen Teil von der Lage der Türen ab. Schaffen die Türen ein Muster von Verkehrswegen, das die Orte in einem Raum zerstört, werden sich die Leute nie wohlfühlen.

Wir haben einmal den Fall eines Zimmers mit einer einzigen Tür. Im allgemeinen sollte diese Tür am besten in einer Ecke sein. Ist sie in der Mitte einer Wand, erzeugt sie nahezu immer ein Bewegungsmuster, das einen Raum in zwei Teile trennt, die Raummitte zerstört und keine einzige Fläche übrigläßt, die groß genug zur Benutzung ist. Die einzige Ausnahme von dieser Regel ist für Gewöhnlich ein langer, schmaler Raum. In diesem Fall ist es durchaus sinnvoll, ihn von der Mitte einer langen Seite zu betreten, da dadurch zwei Bereiche geschaffen werden, die beide annähernd quadratisch und deshalb auch groß genug für die Benutzung sind. Diese Art von zentraler Tür ist besonders dann sinnvoll, wenn der Raum zwei teilweise getrennte Funktionen erfüllt, die sich ganz natürlich in die Hälften aufteilen.

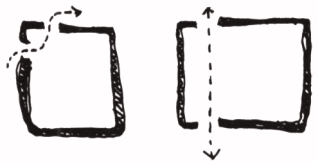
Nun zu den Räumen mit zwei oder mehr Türen: Die einzelnen Türen sollten aufgrund der oben angegebenen Gründe trotz allem in den Ecken sein. Aber



Räume mit einer Tür.

wir müssen jetzt nicht nur die Lage der einzelnen Türen, sondern auch ihr Verhältnis zueinander in Betracht ziehen. Sie sollten womöglich weniger entlang derselben Seite angelegt werden, damit der übrige Raum von den Bewegungen unbeeinträchtigt bleibt.

Ganz allgemein, wenn wir verbindende Linien zwischen den Türen ziehen, dann sollten Bereiche, die von diesen Linien nicht geschnitten werden, groß genug für die sinnvolle Benutzung sein und eine starke positive Form haben — eine dreieckige Restfläche zwischen den Verkehrswegen wird kaum je benutzt werden.



Räume mit mehr als einer Tür.

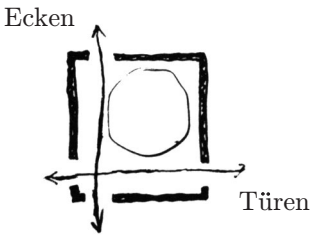
Schließlich sollte man noch beachten, daß dieses Muster nicht auf sehr große Räume zutrifft. In einem sehr großen Raum oder in einem Raum mit einem großen Tisch in der Mitte können die Türen in der Mitte sein und trotzdem einen formellen und großzügigen Eindruck vermitteln. Tatsächlich ist es in diesem Fall vielleicht sogar besser, wenn sie in der Mitte sind, damit diese Wirkung entsteht. Das funktioniert aber nur, wenn der Raum groß genug ist.

Daraus folgt:

Außer in sehr großen Räumen ist eine Tür in der Mitte der Wand nur selten sinnvoll. Anders etwa bei einem Eingangsraum, weil die Charakteristik dieses Raums im wesentlichen von der Tür bestimmt wird. Aber in den meisten Räumen, vor allem in kleinen, leg die Türen möglichst in die Ecken. Hat ein Raum zwei Türen, und die Leute gehen durch, dann leg beide Türen an einem Ende des Raums an.

◊ ◊ ◊

Wenn eine Tür einen Übergang andeutet, wie zum Beispiel in ein Schlafzimmer oder in einem anderem privaten Ort, mach sie so niedrig wie möglich —



NIEDRIGE TÜR (224); und für besonders private Stellen vertiefe den Eingang mit Schrankräumen — SCHRÄNKE ZWISCHEN DEN RÄUMEN (198). Später, wenn Du den Türrahmen machst, leg ihn so an, daß er ein Bestandteil der Wand wird, und verziere ihn nach Belieben — GERAHMTE ÖFFNUNGEN (225), ORNAMENT (249); setz Fenster in die Türen ein, außer bei sehr privaten Zimmern — SOLIDE TÜREN MIT GLAS (237)....

C. Beispiel–Muster aus Erich Gammas Mustersprache

Template Method

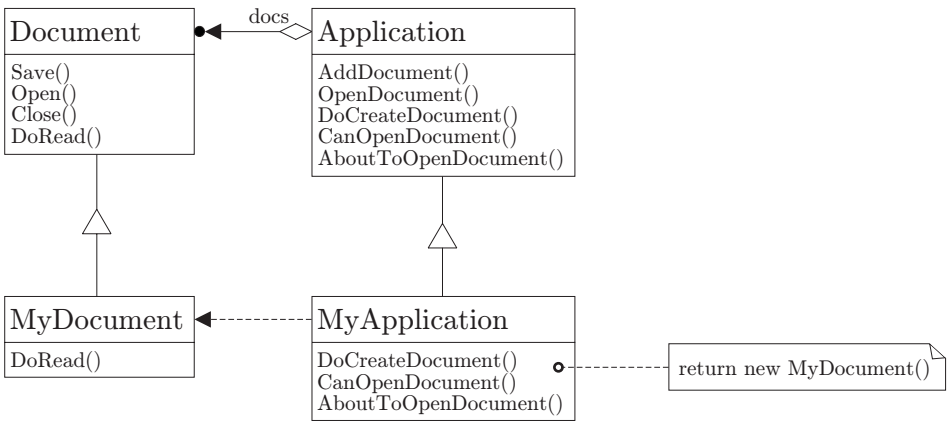
Intent

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Motivation

Consider an application framework that provides Application and Document classes. The Application class is responsible for opening existing documents stored in an external format, such as a file. A Document object represents the information in a document once it's read from the file.

Applications built with the framework can subclass Application and Document to suit specific needs. For example, a drawing application defines DrawApplication and DrawDocument subclasses; a spreadsheet application defines SpreadsheetApplication and SpreadsheetDocument subclasses.



The abstract Application class defines the algorithm for opening and reading a

document in its OpenDocument operation:

```
void Application::OpenDocument (const char* name) {
    if (!CanOpenDocument(name)) {
        // cannot handle this document
        return;
    }

    Document* doc = DoCreateDocument();

    if (doc) {
        _docs->AddDocument(doc);
        AboutToOpenDocument(doc);
        doc->Open();
        doc->DoRead();
    }
}
```

OpenDocument defines each step for opening a document. It checks if the document can be opened, creates the application-specific Document object, adds it to its set of documents, and reads the Document from a file.

We call OpenDocument a **template method**. A template method defines an algorithm in terms of abstract operations that subclasses override to provide concrete behavior. Application subclasses define the steps of the algorithm that check if the document can be opened (CanOpenDocument) and that create the Document (DoCreateDocument). Document classes define the step that reads the document (DoRead). The template method also defines an operation that lets Application subclasses know when the document is about to be opened (AboutToOpenDocument), in case they care.

By defining some of the steps of an algorithm using abstract operations, the template method fixes their ordering, but it lets Application and Document subclasses vary those steps to suit their needs.

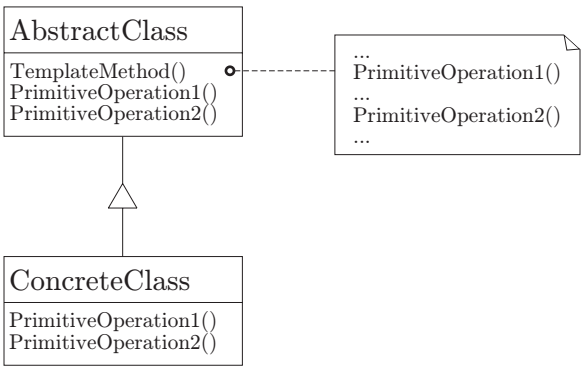
Applicability

The Template Method pattern should be used

- to implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary.
- when common behavior among subclasses should be factored and localized in a common class to avoid code duplication. This is a good example of "refactoring to generalize" as described by Opdyke and Johnson [OJ93]. You first identify the differences in the existing code and then separate the differences into new operations. Finally, you replace the differing code with a template method that calls one of these new operations.
- to control subclasses extensions. You can define a template method that calls "hook" operations (see Consequences) at specific points, thereby

permitting extensions only at those points.

Structure



Participants

- **AbstractClass** (Application)
  - defines abstract **primitive operations** that concrete subclasses define to implement steps of an algorithm.
  - implements a template method defining the skeleton of an algorithm. The template method calls primitive operations as well as operations defined in AbstractClass or those of other objects.
- **ConcreteClass** (MyApplication)
  - implements the primitive operations to carry out subclass-specific steps of the algorithm.

Collaborations

- ConcreteClass relies on AbstractClass to implement the invariant steps of the algorithm.

Consequences

Template methods are a fundamental technique for code reuse. They are particularly important in class libraries, because they are the means for factoring out common behavior in library classes.

Template methods lead to an inverted control structure that's sometimes referred to as "the Hollywood principle," that is, "Don't call us, we'll call you" [Swe85]. This refers to how a parent class calls the operations of a subclass and not the other way around.

Template methods call the following kinds of operations:

- concrete operations (either on the ConcreteClass or on client classes);
- concrete AbstractClass operations (i.e., operations that are generally useful to subclasses);
- primitive operations (i.e., abstract operations);
- factory methods (see Factory Method (107)); and
- **hook operations**, which provide default behavior that subclasses can extend if necessary. A hook operation often does nothing by default.

It's important for template methods to specify which operations are hooks (may be overridden) and which are abstract operations (must be overridden). To reuse an abstract class effectively, subclass writers must understand which operations are designed for overriding.

A subclass can *extend* a parent class operation's behavior by overriding the operation and calling the parent operation explicitly:

```
void DerivedClass::Operation () {
    // DerivedClass extended behavior
    ParentClass::Operation();
}
```

Unfortunately, it's easy to forget to call the inherited operation. We can transform such an operation into a template method to give the parent control over how subclasses extend it. The idea is to call a hook operation from a template method in the parent class. Then subclasses can then override this hook operation:

```
void ParentClass::Operation () {
    // ParentClass behavior
    HookOperation();
}
```

`HookOperation` does nothing in `ParentClass`:

```
void ParentClass::HookOperation () { }
```

Subclasses override `HookOperation` to extend its behavior:

```
void DerivedClass::HookOperation () {
     // derived class extension
}
```

Implementation

Three implementation issues are worth noting:

1. *Using C++ access control.* In C++, the primitive operations that a template method calls can be declared protected members. This ensures that they are only called by the template method. Primitive operations that must be overridden are declared pure virtual. The template method itself should not be overridden; therefore you can make the template method a nonvirtual member function.

2. *Minimizing primitive operations.* An important goal in designing template methods is to minimize the number of primitive operations that a subclass must override to flesh out the algorithm. The more operations that need overriding, the more tedious things get for clients.

3. *Naming conventions.* You can identify the operations that should be overridden by adding a prefix to their names. For example, the MacApp framework for Macintosh applications [App89] prefixes template method names with "Do-": "DoCreateDocument", "DoRead", and so forth.

Sample Code

The following C++ example shows how a parent class can enforce an invariant for its subclasses. The example comes from NeXT's AppKit [Add94]. Consider a class View that supports drawing on the screen. View enforces the invariant that its subclasses can draw into a view only after it becomes the "focus," which requires certain drawing state (for example, colors and fonts) to be set up properly.

We can use a Display template method to set up this state. View defines two concrete operations, SetFocus and ResetFocus, that set up and clean up the drawing state, respectively. View's DoDisplay hook operation performs the actual drawing. Display calls SetFocus before DoDisplay to set up the drawing state; Display calls ResetFocus afterwards to release the drawing state.

```
void View::Display () {
    SetFocus();
    DoDisplay();
    ResetFocus();
}
```

To maintain the invariant, the `View`'s clients always call `Display`, and `View` subclasses always override `DoDisplay`.

`DoDisplay` does nothing in `View`:

```
void View::DoDisplay () { }
```

Subclasses override it to add their specific drawing behavior:

```
void MyView::DoDisplay () {
    // render the view's contents
}
```

Known Uses

Template methods are so fundamental that they can be found in almost every abstract class. Wirfs-Brock et al. [WBWW90, WBJ90] provide a good overview and discussion of template methods.

Related Patterns

Factory Methods (107) are often called by template methods. In the Motivation example, the factory method DoCreateDocument is called by the template method OpenDocument.

Strategy (315): Template methods use inheritance to vary part of an algorithm. Strategies use delegation to vary the entire algorithm.

D. Beispiel–Muster aus Susan Lilly Mustersprache

PDR (Plan–Do–Reflect) Pattern

Intent:

Structure team working sessions.

Motivation:

Student teams waste time by not planning the work that they will do in a time–bound activity session. They also tend to focus so much on the work products they are creating that they are unaware of the process of their own learning.

Applicability:

Use the PDR when student teams are trying to apply what they have learned by doing an activity — an exercise, project or case study.

Structure:

Plan: Document a Plan for the session. A Plan can be as informal as notes in the whiteboard, or canbe captured on a computer. Plan should identify how long to spend on each activity, and who will take various team roles (leader, scribe, etc.)
Do: Carry out the Plan.
Reflect: Explicitly focus on what was learned in the session, what worked, what didn't work, etc. Reflection can be done individually and/or by a group; it can be informal discussion or formal response to written question(s).

Consequences:

• Ensures that the student teams create a Plan for carrying out the assignment; this supports better time management, as well as a feling of student empowerment.
• Encourages the students to be explicit about what they learned (or didn't learn) by incorporating a reflection activity into the work session.
• Takes longer that "just doing the assignment" because of the planning and reflection activities. However, the students seem to get mote learning out of the

technical assignment, especcially if the Reflection is not structured.

Implementation:

• The total time to schedule for the PDR session should include expected time to complete the technical assignment, plus time for the Planning and Reflection. Sample 90-minute PDR: Plan (10 min.), Do (60 min.), Reflect (20 min.).
• A structured reflection session is usually more effective than open discussion ("OK, what did we learn?"). Either the team or a faculty mentor can supply the structure. A page with open-ended questions and space for writing answers works well. Students answer the questions individually, and then may share their answers with the group and discuss. Sample questions : "What is the most interesting thing you learned in this session?" "What is the coolest thing your team came up with today?" "If you could start the session over, what is one thing you would do the same? One thing you would do differently?"

Related Pattern:

The PDR can be used to structure the Activity component(s) of the Extended Unit pattern.

E. Beispiel–Rezept nach Grell und Grell

Das Rezept des Informierenden Unterrichtseinstiegs

Das Rezept des Informierenden Unterrichtseinstiegs erscheint uns selbst weder besonders originell noch ungewöhnlich oder neuartig. Tatsächlich wird dieses Rezept sinngemäß im Alltagsleben ständig mit Erfolg angewendet. Aber jedesmal, wenn wir dieses Rezept bei Studenten oder Lehrern propagieren, gibt es eine ziemliche Aufregung. Offenbar müssen viele einen großen Teil ihrer vertrauten Ansichten über den Haufen werfen oder wenigstens in Frage stellen, bevor sie das Rezept des Informierenden Unterrichtseinstiegs als eine Hypothese akzeptieren können, die man einmal im Unterricht überprüfen sollte. Dieses Mißtrauen ist verständlich, weil das Rezept des Informierenden Unterrichtseinstiegs sich gegen die Ideen über Motivation wendet, die durch die Lehrerbildung verbreitet werden.

Wir geben zu, daß „Motivation“ ein hervorragender Prüfungsstoff ist. Die verschiedenen Theorien lassen sich gut nacherzählen, man kann die Formel von

Heckhausen auswendig lernen, erläutern und eloquent beteuern, daß man gegen sekundäre Motivation ist. Schlimm am Motivationskomplex sind nicht die Theorien als solche, sondern die Tatsache, daß sie praktisch nicht helfen, sondern schaden und daß sie trotzdem in der Form von Moralforderungen („Du sollst die Schüler richtig motivieren") quicklebendig bleiben und Lehrern, die ihren Beruf ernstnehmen, das Herz schwermachen. Und am schlimmsten ist, daß das Gebot, daß man die Schüler zu motivieren habe, sich in der Praxis mit üblen Gewohnheiten verheiratet. Denn diese üblen Gewohnheiten bekommen dadurch den vornehmen, wissenschaftlich klingenden Namen „Motivation" und sind für den Rest der Ewigkeit tabu.

Exkurs: Warum eignen sich Motivationstheorien nicht für die Konstruktion von Handlungsrezepten?

Eigentlich müßten wir an dieser Stelle einen Exkurs von 300 Seiten schreiben, um dem Thema gerecht zu werden, denn es gibt so viele Motivationstheorien und Varianten von Motivationstheorien und Verfeinerungen von Varianten von Motivationstheorien, daß fast alles, was man darüber sagt, gleichzeitig stimmt und nicht stimmt. Aber genau dies ist ein Grund, warum diese Theorien für die Praxis so unfruchtbar sind. Wenn alles richtig und falsch ist, dann ist es schwierig, klare Aussagen aufzufinden und daraus Handlungsrezepte zu entwickeln.

Der erste große Fehler des Motivationsbegriffs ist, daß das Wort zur Zauberformel geworden ist, die beinahe alles bedeuten kann. Die meisten pädagogischen Psychologen haben das noch nicht gemerkt, aber Wissenschaftler aus anderen Disziplinen unterhalten sich schon darüber, wie man den Begriff „Motivation" am geschicktesten abschaffen kann (z.B. die Ethologen, vgl. Heymer, 1977, S. 82).

Ein zweiter Fehler des Motivationskonzepts ist die unselige Unterscheidung zwischen primärer und sekundärer Motivation. Diese Begriffsbildung transportiert den verwirrenden Aberglauben, daß man Motivationszustände bei Schülern entweder von außen einschalten kann oder aber daß afe der einzelne Schüler von sich aus ‚innerlich' einschaltet — und daß man dieses innerliche Einschalten angeblich durch geschickte „Motivation" von außen anschalten kann. Dies belastet Lehrer mit der unlösbaren Aufgabe: Du sollst die Schüler intrinsisch motivieren. [..]

Annahmen, die dem Rezept „Informierender Unterrichtseinstieg" zugrunde liegen

Die erste Annahme ist: Wir können andere Menschen nicht motivieren, sondern jeder Mensch motiviert sich selbst. Wir können z.B. Ihr Interesse oder Ihren Lernwillen nicht von außen anschalten. [..]

Die zweite Annahme: Menschen sind eher bereit, ihre Motivation einzuschalten, wenn sie wissen, wofür. [..]

Auch [Schüler] können in der Regel ihre Motivation erst dann einschalten, wenn sie wissen, um was es genau geht und welchen Sinn es hat, die Motivation

einzuschalten.

Wenn wir mit Erwachsenen interagieren, finden wir es ganz selbstverständlich, daß sie zuerst Informationen fordern, bevor sie sich motivieren. Aber bei Schülern haben wir den Verdacht, daß sie ihre Motivation böswillig ausschalten werden, wenn sie erst genau über das Unterrichtsvorhaben informiert sind. Wir trauen ihnen nicht zu, daß sie rationalen Argumenten zugänglich sein könnten, sondern wir halten sie für „Kinder", mit denen man „kindgemäß" umgehen muß — und wir zwingen sie dadurch in die Rolle eines unmündigen Wesens, das seinen irrationalen Launen folgt und nicht begreifen kann, was zu seinem eigenen Besten ist. So verzichten wir freiwillig auf die Kräfte, die entstehen, wenn die Schüler ihre Motivation einschalten und willkürlich lernen. [..]

Das Rezept „Informierender Unterrichtseinstieg" und was es vorhersagt

Das Rezept, das wir Ihnen vorschlagen, verlangt, daß Sie Ihre Vorurteile über die Kindlichkeit von Schülern vergessen und Schüler genauso behandeln wie Sie einen Freund oder Bekannten behandeln würden.

Für den Unterricht heißt das: Versuchen Sie nicht, nach einer prunkvollen Motivierungsidee zu suchen, die die Schüler zum Lernen verführt, sondern nehmen Sie die Schüler als vernünftige Wesen ernst und sagen Sie ihnen am Stundenbeginn so einfach und so klar und so interessant, wie Sie es nur formulieren können, was in der Stunde passieren soll und warum. Erzählen Sie den Schülern alles, wenigstens alles Wichtige, über den Plan der kommenden Stunde, den Sie in Ihrem Kopf haben. Schreiben Sie die wichtigsten Punkte an die Tafel, auf eine Folie oder auf das Arbeitspapier, damit die Schüler eine Übersicht über Ihren Plan bekommen und dort immer wieder „nachschlagen" können. Auf diese Weise machen Sie für die Schüler sichtbar, was sonst immer nur in den Köpfen der Lehrer verborgen ist und von niemandem eingesehen werden kann. Sie legen die Karten auf den Tisch und verzichten auf jede Geheimnistuerei. Es macht nichts, wenn Sie Ihren Plan für die Stunde so darstellen, daß Schüler dabei lecker werden oder sich freuen. Sie brauchen den Informierenden Unterrichtseinstieg nicht langweilig oder trocken zu gestalten. Die Schüler können ruhig gefühlsmäßig angesprochen werden. Aber das wichtigste Ziel ist doch: den Schülern den kommenden Unterricht durchsichtig zu machen.

Unsere Voraussage ist: wenn Sie am Stundenbeginn einen Informierenden Unterrichtseinstieg vorausschicken, dann werden *mehr* Schüler bereit sein, sich selbst zu motivieren, als bei den üblichen Unterrichtseinstiegen, die nach dem Prinzip konstruiert sind, Informationen zurückzuhalten.

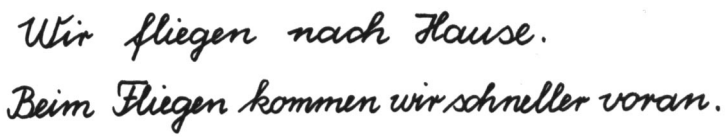
Wichtig ist beim Informierenden Unterrichtseinstieg, daß die Schüler um ihr Einverständnis mit dem Unterrichtsplan gebeten werden, wie das bei Menschen üblich ist. Oft ist es auch sinnvoll, Auswahlmöglichkeiten zu nennen, statt einen perfekten Plan zu beschreiben, der nur so und nicht anders vollstreckt werden darf. Einige Schüler werden sich freuen, daß sie mitentscheiden oder mitüberlegen dürfen. Andere können dabei lernen, daß man sich auch einmal selbst für

etwas entscheiden muß und sich nicht immer alles vorsetzen lassen kann.

Wie sieht ein lnformierender Unterrichtseinstieg aus?

Als gelernter Motivationsverehrer würden Sie eine Rechtschreib- stunde im 3. Schuljahr vielleicht folgendermaßen beginnen:

„Wir haben jetzt Deutsch und machen Rechtschreibung. Holt bitte die Rechtschreibhefte heraus. Ich schreibe euch jetzt etwas an die Tafel. Schaut es euch genau an.„ Sie schreiben:

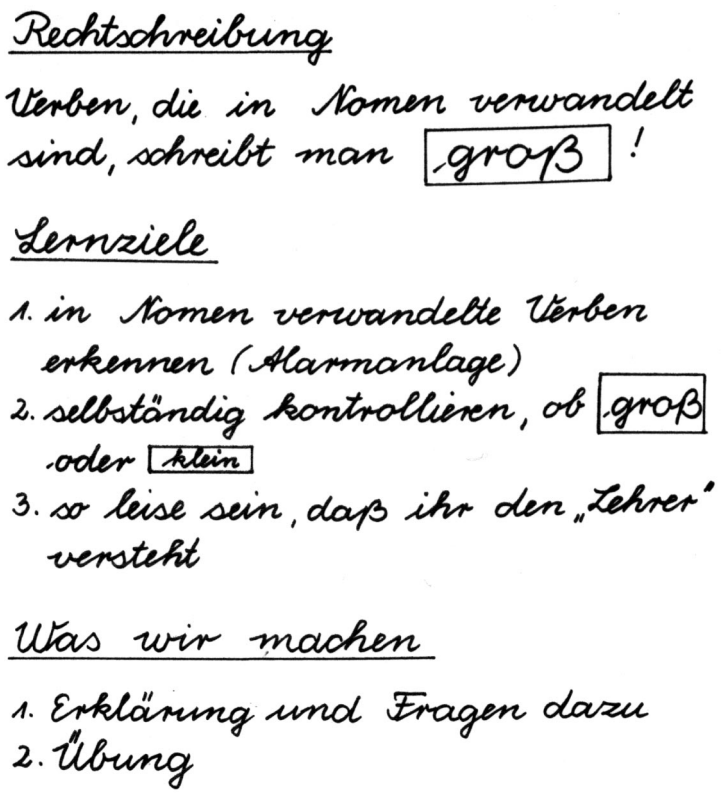


Dann drehen Sie sich zur Klasse um, schauen die Schüler mit großen Augen erwartungsvoll an und fragen: „Was fällt euch daran auf? (Pause) Wer hat etwas gemerkt? (Pause) Achtet mal darauf, wie die Wörter bei diesen beiden Sätzen geschrieben werden.“ Und so weiter, je nachdem wie lange es dauert, bis ein Schüler begriffen hat, daß Sie von ihm hören wollen, wie erstaunt er darüber ist, daß „FLIEGEN“ einmal groß und einmal klein geschrieben wird.

Das ist der traditionelle Erarbeitungs–Motivierungsversuch-Unterrichtsbeginn — und den müssen Sie sich verkneifen, wenn Sie einen Informierenden Unterrichtseinstieg machen wollen. Dann würde die gleiche Stunde etwa so beginnen:

Lehrer: „Wir machen in dieser Deutschstunde Rechtschreibung. Und zwar sollt ihr heute lernen, daß Verben in Nomen verwandelt werden können und daß sie dann groß geschrieben werden müssen. Das ist wichtig, weil man sehr viele Rechtschreibfehler macht, wenn man in Nomen verwandelte Verben nicht von echten Verben unterscheiden kann.
Ich hab mir die Stunde so gedacht:
l. Ich erkläre euch, woran man erkennt, daß ein Verb in ein Nomen verwandelt wurde und daß man es also groß schreiben muß. Und dann sollt ihr 2. die Gelegenheit haben, das Erkennen solcher in Hauptwörter verwandelter Verben genau zu üben. Ihr baut dabei sozusagen in eurem Gehirn eine kleine Alarmanlage ein: sobald euch ein Verb in die Quere kommt, klingelt eure Alarmanlage, und ihr wißt: Hier muß ich überlegen: Groß oder klein? Wenn diese Warnanlage erst richtig funktioniert, braucht ihr euch nicht mehr viel darum zu kümmern, denn sie arbeitet dann automatisch, und ihr braucht nicht jedesmal lange zu überlegen, wie das Wort geschrieben werden muß. Dieses Üben soll in einem Diktat geschehen. Dabei ist immer einer von euch abwechselnd der Lehrer. Ihr übt, ganz allein zu kontrollieren, ob das Verb groß oder klein geschrieben wird. Und gleichzeitig könnt ihr dabei versuchen, beim Diktat so leise zu sein, daß ihr den Lehrer verstehen könnt.
Ich glaube, daß euch das Spaß machen wird und daß ihr schnell erkennen werdet, worauf es bei der Groß– oder Kleinschreibung von Verben ankommt.
Ja, wollt ihr dazu etwas fragen?
Seid ihr damit einverstanden, daß wir es so machen?“

Dieser Informierende Unterrichtseinstieg dauert etwa 2 Minuten. Die Erläuterungen werden übersichtlicher für die Schüler, wenn Sie die wichtigsten Punkte an die Tafel geschrieben haben und beim Sprechen immer auf die entsprechende Stelle zeigen:



[..]

Argumente gegen das Rezept „Informierender Unterrichtseinstieg", die wir häufig hören

Von Lehrern, die das Rezept noch niemals probiert haben, kommen gewöhnlich tausend Argumente gegen den Informierenden Unterrichtseinstieg. Viele können sich ein Lächeln nicht verkneifen und winken mit der Pose des erfahrenen Praktikers lässig ab. [..]

In solchen Argumenten kommen die „ästhetischen Kriterien" zum Ausdruck, nach denen viele Lehrer Unterricht beurteilen und planen. Es sind die Kriterien, die zum Muster des Erarbeitungsunterrichts gehören. Außerdem hört man darin die Angst, die Schüler könnten ihre Mitarbeit verweigern, wenn man ihnen die Ziele und Arbeitsschritte zu früh mitteilt. Ein Lehrer sagte uns in so einer Diskussion: „Es ist doch legitim, wenn man den Schülern noch nicht gleich am Anfang sagt, daß sie hinterher schreiben müssen. Wenn ein Lehrer gleich sagt: ‚Und dann schreiben wir auch noch', sind die Schüler gleich desillusioniert." Wir halten es für besser, wenn Schüler am Anfang desillusioniert sind, als wenn

sie sich hinterher hereingelegt fühlen.

Viele Lehrer haben am Informierenden Unterrichtseinstieg auszusetzen, daß das Verfahren zu sachlich (zu kalt, zu mechanisch, nicht natürlich, nicht kindgemäß) sei. Das ist teilweise richtig. Informierende Unterrichtseinstiege sind sachlicher als Motivationstricks, genauso „unnatürlich“ wie sie, und sie sollen gar nicht ‚kindgemäß' sein, weil unsere Vorstellungen von Kindgemäßheit falsch sind. Aber es stimmt nicht, daß durch Informierende Unterrichtseinstiege „alles, was noch Spaß macht, aus dem Unterricht entfernt“ wird, wie ein Lehrer befürchtete.

Es ist nicht so, daß nur Geheimnisse Spaß machen. Man kann sogar mehr Spaß haben, wenn man weiß, was man tut.

Oft hörten wir auch das Argument „In manche Stunden gehört diese Art des Einstiegs einfach nicht hinein.“ Wir konnten aber nicht in Erfahrung bringen, um welche Art von Stunden es sich dabei handelt. Darum ist dieser Einwand für uns nicht stichhaltig.

Die meisten Lehrer glauben, daß der Informierende Einstieg sich vor allem in den höheren Klassen eigne und für die „Kleinen“ reichlich unpassend sei. Wir sind nicht dieser Ansicht, denn wir halten es für destruktiv, wenn Schüler in der Grundschule jahrelang auf Märchenstunden dressiert werden.

Manche Lehrer bezeichnen es als Nachteil des Informierenden Unterrichtseinstiegs, daß das Verfahren den Schülern eine gewisse Kontrolle über den Lehrer ermöglicht. So könnten die Schüler es etwa bemängeln, wenn nicht alles geschafft wird, was angekündigt wurde. Wir halten dies für einen Vorteil dieses Rezepts.

Sehr oft wird auch behauptet, daß die Schüler zu stark gelenkt würden, wenn Lehrer ihren Plan und die Ziele bekanntgeben. Dadurch seien die einzelnen Schritte vorher festgelegt, man könne nun nicht mehr auf die spontanen Einfälle der Schüler und auf die Besonderheiten ihrer Arbeitsergebnisse reagieren, und es wäre unmöglich, die stets unvorhersehbaren Zufälligkeiten des Unterrichtsprozesses angemessen zu berücksichtigen.

Dazu ist zu sagen, daß die meisten Lehrer in den allermeisten Unterrichtsstunden einen relativ starren — wenn auch oft verschwommenen — Plan durchzusetzen versuchen, an den sie die Schüler nicht heranlassen. Im Normalfall wird dieser Plan auf Schleichwegen verfolgt. Die Schüler werden indirekt stark gegängelt und bekommen nicht die Chance, eigene Ideen zum Vorgehen zu äußern, weil alles in der Hand des Lehrers liegt und allein sein Geheimnis bleibt. Gibt ein Lehrer dagegen seine Planung den Schülern bekannt, dann versucht er zwar, die Schüler und ihr Lernverhalten zu lenken, und dazu ist er ja auch Lehrer, aber gleichzeitig bietet er den Schülern Kontrollmöglichkeiten, denn sie können jetzt ja zu seinem Plan Stellung nehmen. Wie wir eben gesehen haben, kritisieren Lehrer am Informierenden Unterrichtseinstieg oft gerade dies: daß den Schülern die Möglichkeit eröffnet wird, kritisch Stellung zu beziehen, denn sie fürchten, die Schüler könnten dies zur Torpedierung des Unterrichts benutzen. Durch

einen Informierenden Unterrichtseinstieg werden die Schüler jedoch geradezu darauf gestoßen, daß sie mitsteuern und den Lehrerplan modifizieren können.

Interessant ist auch dieser Einwand: „Wenn ich den Schülern gleich am Anfang genau sage, was sie lernen sollen usw., dann brauche ich ja gar nicht mehr zu unterrichten, weil ich nach 10 Minuten schon mit dem Thema fertig bin." Offenbar ist es — besonders für junge Lehrer — manchmal ein Problem, die 45 Minuten der Unterrichtsstunde irgendwie vollzukriegen. Jedenfalls scheint das als ein heimliches Motiv hinter manchem umständlichen methodischen Vorgehen zu stehen. Man steuert die Lernziele nicht direkt an, sondern schleicht sich irgendwie auf Umwegen heran, um so das peinliche Erlebnis zu vermeiden, verfrüht mit dem Stoff fertig zu werden. Diese Angst hängt auch damit zusammen, daß dem Anfänger ständig eingetrichtert wird, jede Unterrichtsstunde müßte eine thematisch und methodisch abgerundete und in sich geschlossene „künstlerische Einheit" ohne „Stilbruch" sein.

Am interessantesten ist ein Argument, das beinahe in jeder Diskussion über den Informierenden Unterrichtseinstieg von einem Teilnehmer zur Verteidigung der Motivationsideologie vorgetragen wird. „Warum sind Sie eigentlich so gegen Motivieren? Wenn Sie die Schüler informieren, dann ist das doch im Grunde auch nichts anderes als eine Motivation." Dieser Gedanke zeigt deutlich, daß der Begriff „Motivation" eine Leerformel ist. [..]

Vorteile des Informierenden Unterrichtseinstiegs und Erfahrungen

Ein Informierender Einstieg kostet nichts, man braucht weder viel Zeit dafür, noch muß man geniale Ideen produzieren wie beim traditionellen Motivationskonzept, und — ein solcher Einstieg richtet keinen Schaden an. Man muß einfach nur wissen, was man will und bereit und fähig sein, dies den Schülern klarzumachen.

Hier liegt andererseits auch eine kleine Schwierigkeit: viele Lehrer wissen selbst nicht immer so genau, was sie eigentlich wollen. Aber gerade das ist ein großer Vorteil des Informationseinstiegs. Er erzieht nämlich Lehrer dazu genauer zu überlegen, was sie wollen. Wer seinen Stunden häufiger einen Informierenden Einstieg vorausschickt, der zwingt sich selbst, eine präzisere Vorstellung von seinem Unterricht zu entwickeln, gewöhnt sich daran, Unterricht klar zu strukturieren und ist auf diese Weise besser vorbereitet. Man gewöhnt sich daran, den Unterricht aus der Perspektive der Schüler zu sehen, indem man den Plan für sie verständlich darstellt. Wenn man das einige Zeit gemacht hat, lernt man auch, dem Unterricht selbst dann eine eindeutige Struktur zu geben, wenn man nicht alle Einzelheiten vorher durchdacht hat. Denn durch die Gewohnheit, Ziele, Begründungen und Arbeitsschritte offenzulegen, erwirbt man ein Repertoire strukturierender Maßnahmen und Vorgehensweisen, die man nach einiger Zeit flexibel und improvisierend einsetzen kann.

Ein Informierender Unterrichtseinstieg ist nichts weiter als eine erläuterte Tagesordnung. Eine solche Tagesordnung erhöht die Wahrscheinlichkeit, daß viele der Tagenden sich bemühen werden, beim Thema zu bleiben. Natürlich kann sie

nicht vollkommen verhindern, daß einige trotzdem vom Thema abweichen. Das ist aber kein Grund, auf eine Tagesordnung gleich ganz und gar zu verzichten. [..]

Haben die Schüler sich nur einer massiven Lenkung durch den Lehrer angepaßt? Nach unseren Erfahrungen sind Schüler sehr kritisch, wenn sie eine schlechte, diffuse Tagesordnung vorgesetzt bekommen. Andererseits sind die meisten Schüler bereit, mitzumachen, wenn sie eine Tagesordnung bekommen, die ihnen vernünftig erscheint. Eine solche Tagesordnung ist ein Mittel der Verständigung und nicht ein Heiligtum, das vor jeder Abänderung um jeden Preis geschützt werden muß. Lehrer, die von sich glauben, nur sie allein könnten aufgrund ihrer pädagogischen Qualifikation und Verantwortung den Unterrichtsablauf steuern und Schüler hätten sich da herauszuhalten, sollten lieber bei den alten Motivationsverfahren bleiben und den Informierenden Einstieg gar nicht erst versuchen.

Wenn Sie also zu Beginn der Stunde den Schülern die Ziele und Arbeitsschritte mitteilen, sollten Sie bereit sein, unter Umständen mit den Schülern Themen wie diese zu diskutieren:

– Wie findet ihr diesen Plan?
– Habt ihr Lust dazu?
– Wie könnte man diese Lernziele am besten erreichen? Welche Ideen habt ihr?
– Warum ist es wichtig, das zu lernen?

Schüler, die die Tagesordnung nicht kennen, können nicht über solche Fragen diskutieren.

Lehrer, die den Informierenden Unterrichtseinstieg probiert haben, sind überrascht, wie positiv die Schüler darauf reagieren. Die vermuteten Schwierigkeiten (daß die Schüler den Plan pauschal ablehnen und unwillig daran herummäkeln würden) bleiben aus. Statt dessen nehmen die Schüler oft intelligent Stellung, engagieren sich im Sinne der Lernziele und verhalten sich wesentlich disziplinierter als gewohnt. [..]

Auch Unterrichtsbeobachter haben etwas davon. Ein Referendar erzählt:

„Wenn ich mit dem Seminar eine Hospitationsstunde besichtige, stecke ich die Unterrichtsvorbereitung gleich in meine Tasche, denn ich will sie erst hinterher lesen. [..] Erst dann schaue ich in [die jeweilige] Vorbereitung und lese mir die dort formulierten Lernziele durch. Entweder ist meine Schätzung richtig, oder — was oft der Fall ist — sie ist falsch. Ich frage mich: Wenn ich schon nicht merke, was der Lehrer eigentlich will, wie muß es dann erst den Schülern gehen?“

Das Mittel des Informierenden Unterrichtseinstiegs gibt dem Unterricht Zieltransparenz. Die Ziele werden für Schüler und Beobachter sichtbar und der Lehrer befreit sich aus seiner einsamen und gottähnlichen Rolle, indem er sich in die Karten schauen läßt.

Indikation und Kontraindikation für das Rezept

Wann soll man das Rezept anwenden und unter welchen Bedingungen sollte man es vermeiden?

Das Grundprinzip des Informierenden Unterrichtseinstiegs ist: Der Lehrer versucht, den Schülern seine eigenen Absichten soweit zu verdeutlichen, wie es für deren Lernen nützlich ist.

Wenn Sie diese Formel zur Beurteilung heranziehen, dann werden Sie nicht in Versuchung kommen, den Schülern überflüssige Ausführungen über Ihre Absichten zu machen.

In vielen Sportstunden ist es überflüssig, ausführlich die Ziele zu beschreiben oder eine Übersicht über den geplanten Stundenverlauf zu geben, weil die Schüler sehen können was sie lernen sollen und weil sie das Muster des Vorgehens bereits aus vielen früheren Stunden kennen und verstehen. Auch wenn Sie ein Diktat diktieren wollen, benötigen die Schüler kaum noch Erläuterungen.

Ein Informierender Unterrichtseinstieg ist nicht angezeigt, wenn es keine neuartigen oder speziellen Absichten zu vermelden gibt, weil man etwas zum achtundvierzigsten Male wiederholt.

Ein Informierender Einstieg ist dagegen immer dann angezeigt, wenn Sie das Gefühl beschleicht, Sie müßten den Schülern in der guten alten Motivationsabsicht irgend etwas zunächst noch verheimlichen. In solchen Fällen müssen Sie Ihren Plan sehr kritisch durchdenken und sich die Frage stellen, ob Sie durch das Verheimlichen wirklich das Lernen der Schüler fördern und beschleunigen oder ob sie dadurch nur den Unterrichtsverlauf unnötig in die Länge ziehen. Denken Sie daran, daß direktes Vorgehen immer besser ist als indirektes Anschleichen und daß Schüler wesentlich mehr davon haben, wenn sie etwas selbständig ausführen und üben können, als wenn sie erst durch ein langwieriges und undurchschaubares Verfahren geschleust werden, an dessen Schluß irgendeine lächerliche Erkenntnis steht, die dann doch nur von zwei oder drei Schülern begriffen wird. [..]

Informierende Unterrichtseinstiege sind auch nötig, wenn der Plan der Stunde vom Lehrer noch nicht genau festgelegt ist, weil er sich erst im Verlauf des Unterrichts entwickeln soll. [..]

Auch [Erläuterungen zu aus didaktischen Gründen nicht festgelegten Plänen] sind Informierende Unterrichtseinstiege, allerdings nur dann, wenn der Lehrer, der so spricht, wirklich keine anderen Absichten in seinem Koffer versteckt hat, als er den Schülern zur Kenntnis gibt. Hat er dagegen schon einen genauen Plan, den er den Schülern nur noch nicht mitteilen will, weil er glaubt, er könne diesen Plan am besten durchsetzen, wenn er ihn gar nicht erst vorzeigt, dann sind diese Ausführungen betrügerische Unterrichtseinstiege. Und die sind niemals angezeigt.

Zusammenfassend: Das Rezept des Informierenden Unterrichtseinstieges kann immer angewendet werden, wenn die Schüler etwas Neues lernen sollen. Man kann auf dieses Rezept verzichten wenn die Schüler sowieso wissen, was, warum und wie etwas gemacht werden soll.

Hinweise für die Gestaltung Informierender Einstiege und Warnung

vor möglichen Fehlern

Zuerst zu den Fehlern, die Sie vermeiden sollten.

Da das Mitteilen der Absichten, Lernziele und Arbeitsschritte samt Begründungen den Schülern beim Lernen helfen und sie zum Mitarbeiten veranlassen soll, dürfen Sie den Informierenden Unterrichtseinstieg nicht als Selbstzweck sehen. Es ist unwichtig, ob Ihr Einstieg ein perfektes Exemplar seiner Art ist. Wichtig ist nur, daß er seine Arbeit macht, nämlich den Schülern Ihre Lehrabsichten ausreichend deutlich zu machen. Streben Sie deswegen nicht Vollständig- keit an, sondern Verständlichkeit und Übersichtlichkeit. Fassen Sie alles zusammen, was sich zusammenfassen läßt, und reduzieren Sie Ihre Ausführungen auf das Wichtigste. Rezitieren Sie den Schülern nicht alle sechsundzwanzig operationalisierten Lernziele, die Sie zur Feier der Unterrichtsbesichtigung durch das Seminar formuliert haben, sondern formulieren Sie die Ziele so, daß sie von Schülern verstanden werden. [..]

Zählen Sie nicht alle vierunddreißig Unterrichtsphasen auf, die Sie dem Studienleiter zuliebe in Ihre schriftliche Vorbereitung geschrieben haben, sondern fassen Sie die Einzelschritte für die Schüler zu drei bis fünf sinnvollen Hauptschritten zusammen. (Wenn Sie gelernt haben werden, das Erarbeitungsmuster [(ein weiteres, von Grell und Grell beschriebenes Rezept)] zu ignorieren, nimmt die Anzahl der vorgeplanten Einzelschritte sowieso bald ab.)

Wenden Sie das sogenannte Kofferprinzip an. Stellen Sie sich vor, jeder Hauptschritt der Unterrichtsstunde sei in einem Koffer verpackt, der eine Überschrift trägt, damit man die Koffer nicht verwechselt. Nennen Sie beim Informierenden Unterrichtseinstieg nur die Namen der einzelnen Koffer, damit die Schüler ungefähr Bescheid wissen, was darin ist und erklären Sie die Einzelheiten später, wenn der jeweilige Koffer aufgemacht wird. Auf diese Weise können Sie die für das Lernen wichtigen Informationen nach und nach herausrücken, ohne daß den Schülern die Übersicht über den Gesamtplan verlorengeht.

Geben Sie den Schülern nach Möglichkeit auch Ihre persönlichen Begründungen für bestimmte Einzelschritte an. [..]

Denken Sie daran, daß es unheimlich viele verschiedene Arten gibt, die Schüler über das zu informieren, was Sie geplant haben. Wenn Sie Ihre Phantasie nicht eintrocknen lassen, ist kein Informierender Unterrichtseinstieg wie der andere.

Vergessen Sie nicht, daß die Schüler zu Ihrem Plan Stellung nehmen dürfen und daß sie sogar das Recht haben, ihn abzulehnen. Fordern Sie die Schüler auf, ihre Meinung zu sagen, Fragen zu stellen, Ideen beizusteuern. Argumentieren Sie mit ihnen, wenn sie den Sinn Ihres Plans nicht erkennen, und ändern Sie Ihren Plan, wenn die Schüler bessere Ideen haben als Sie. Gewöhnlich akzeptieren die meisten Schüler Ihren Plan oder eine Variante davon, wenn Sie ihn ausreichend genau erklärt haben. Meist dauert dies nur wenige Minuten.

So können Sie das Rezept trainieren:

Wir stellen uns vor, daß Sie sich ab und zu mit Kollegen oder Freunden treffen.

Vielleicht, um über die Probleme an der Schule oder im Seminar oder sonstwo zum soundsovielten Male die altbekannten traurigen neuesten Nachrichten auszutauschen, vielleicht auch mit dem Vorsatz, auf keinen Fall über die Schule zu reden. Wie dem auch sei, Sie könnten den Vorschlag machen, einmal das Rezept des Informierenden Unterrichtseinstiegs zu trainieren. Vielleicht treiben die anderen an, vielleicht bringt es ihnen sogar Spaß, und möglicherweise entschließen Sie sich sogar, öfter mal etwas ähnlich Sinnvolles zusammen zu machen. Wir kennen Lehrergruppen, die sowas tatsächlich tun und es gar nicht mehr lassen wollen. Dieses Arbeitsblatt ist ein Vorschlag, an dem Sie sich orientieren könnten. [..]

Checkliste „Informierender Unterrichtseinstieg“

Nicht vergessen:
Ein Informierender Unterrichtseinstieg hat die Funktion, den Schülern die Ziele und den Plan der Stunde transparent zu machen. Die Schüler bekommen so die Chance, Stellung zu nehmen und sich bewußt mit der Lernarbeit zu identifizieren oder sich zu distanzieren.

1. *Wie lange dauert die Information am Stundenbeginn?*
   Beginn: Ende: Dauer:

2. *Der Lehrer nennt den Schülern die Lernziele und erläutert sie anschaulich.*

[..]

F. Stellungnahme von Philip Wong (originalsprachlicher Wortlaut)

Background

Design Study 4 is a course offered in the undergraduate program at the School of Design, Hong Kong Polytechnic University. The title of the course is: *Information, Interaction and Global Context*. With the objective of providing a learning environment so that students will be able to gain online experience in communication, negotiation and collaboration, I started to investigate into available options. The following is an attempt to summarize and evaluate the process of establishing this online environment. I also intent to illustrate why a custom designed platform is preferred over an off-the-shelf turnkey system.

The system has to perform the following functions:

1. An e-mail system capable of managing groups and mailing lists. Since students will work in groups, and because we encourage them to negotiate, it is expected that memberships of these groups will change as the project progress. As I do not intend to sit in front of the computer all day long monitoring these changes, a system that enables students to perform group management without compromising security is needed. This system will act as the main channel of information about the course.

2. Whether a design process is successful depends, to some extend, on the spontaneous interaction between designers and flow of ideas. To compliment on the inadequacy of e-mail in this respect, video-conferencing becomes the tool of choice. Unfortunately, it is very expensive to implement on the network in a large scale (100+ students), so it is not feasible. I finally resorted to chatrooms in conjunction with web pages to provide an environment that supports close to real time textual and visual communication. Textual is preferred over verbal communication for the sake of conveniences in logging the communication. These logs will be evaluated for assessment purposes. Students will also be forced to organize and structure their thoughts better before they send a message, hence

improving on efficiency.

3. The system has to provide a central storage for data and information generated by communications between students. This storage has two purposes. The first one is to provide an archive for all students' work. The archive will be web based (including uploading and downloading of files) with a graphical user interface to facilitate retrieval of information. The second purpose of the storage system is to act as a convergence point of ideas so that everyone in the system will have a clear picture of what others are doing. This will encourage collaboration and minimize workload redundancy.

4. The main platform for presentation is the web site. Since all of our students work with multimedia files, the web server has to support the most update mime types with no limitation in file size. Students are allowed to install server-side scripts and set up databases to improve on their web site's functionality should they choose to. Interface design is an important aspect of the students' presentation so working with pre-designed templates as offered by commercial packages is not acceptable.

5. The system will also be used to keep track of students' communications and present qualitative information about these communications. E.g. In a phase where students are expect to form groups, tutors will have to be able to obtain quantitative information from the system about individual student's communication activities such as the frequency of communication, the form of communication and the number of different parties a particular student has communicated with. The quantitative data can be used, among other elements, to deduce the quality of the communication process.

6. As an assessment tool, the system will provide tutors an interface to a database that records the students' performance. In the case of continuous assessment, students will have access to their latest grades and/or evaluation of performance. To avoid misunderstandings, students will also have a chance to reflect their opinions on their grades should they find it unacceptable. This will inevitably assist students in evaluating their progress and improve on performance.

Evaluation

The Hong Kong Polytechnic University uses WebCT as the platform for online teaching. It is managed by ITS (Information Technology Services) which is an independent department governing network strategies and hardware infrastructure.

The good side being that the system is managed by professionals providing support and reliability. However, IT professionals sometimes tend to be very

fond the concept: "being in control". I am sure there are legitimate reasons for being so protective of their system, but if their insistence has come to the point where teaching and learning demands are compromised (as in this case), then it is inevitable that another system has to be developed.

Being a commercial package, WebCT offers a lot of functions and performance desired. It has a comprehensive account management and authentication system; offers online examination authoring and grading; communication tools such as web hosting, message boards and group e-mail; and archival/retrieval of teaching materials. In every sense, it is capable of carrying out online teaching.

I finally decided to opt for custom building our own platform based on the following reasons:

1. WebCT does not provide chatrooms, which would be essential to the course. The main reason probably being chatrooms tend to overload servers very easily. With a custom built system, we could easily add servers to accommodate the increase in load. When the course is finished, these extra servers can be turned back into desktop machines for everyday use. In the case of a system failure, instead of having a campus wide breakdown, only one course is affected.

2. Even with the provided functions, there is not enough flexibility on the operational level. E.g. Students will not be able to use server-side scripts and databases and there is a quota on the disk space that each student can use, etc. This can be considered not as a flaw in the design of WebCT, but rather a management inefficiency. The whole setup (software + hardware + management) reflects an inadequate understanding by its designer on the demand of a "constructive" learning environment.

3. WebCT is designed for teachers not familiar with IT and web authoring, with good intentions. Templates are provided to minimize hassle in the process of authoring courseware, and students will have a consistent environment to work in when their learning progress to other courses. However, WebCT does not provide an alternative to templates; hence inhibiting customization of the courseware interface; which, in a school that teaches design, is unacceptable. In retrospect, we discovered that the freedom of creating our own interface contributed to the student's establishment of a sense of ownership, which encouraged students to take a more active role in the course.

4. Being able to manage our own system also means that we have much more freedom in setting up the system in a way we prefer. Tutors will have access to vital information such as server log files (would have otherwise been considered confidential) which provide data for evaluating the quality of communications. Students will have more participation in constructing their own learning environment.

5. The advance of the Free Software Movement has made it possible to build a high quality system at relatively no cost. There is no doubt that the usage of these freely available software will become the mainstream in software development. Complying with these standards at an early stage will save a lot of resources and efforts later on.

Conclusion

DS4 is close to the end of being run for the second time. Many of the original requirements have been fulfilled while others are still waiting to be implemented. By running our own system, we have been able to learn a lot more about how they should be set up and configured. A lot of resources have been invested into the system and new ideas have been inspired by students' participation. We encountered problems which would have otherwise been avoided should I chose not to built our own system. New modes of learning will eventually come out of these experimentations. Customization, integration and decentralization will be the guiding force of system development. It is in this spirit and direction that our "online learning environment" will be developed.

G. Glossar

Architektur Als Architektur bezeichnet man die Tätigkeit des Entwerfens von (insbesondere bewohnbaren) Gebäuden sowie dabei verwendete Methoden, dabei entstehende Produkte und deren Stile. Im weiteren Sinn versteht man unter Architektur das Resultat eines systematischen Entwurfs und dessen Struktur. Man spricht daher auch im Zusammenhang mit Computern oder Netzwerken von Architekturen und meint damit die dem jeweiligen Aufbau oder Typ zugrundeliegenden Prinzipien, Gesetzmäßigkeiten und Charakteristika.

Assembler Ein Übersetzungsprogramm (Systemwerkzeug), das benutzt wird, um Programme, die in einer 'Assembler–Sprache' geschrieben wurden, in ↗Maschinencode zu übersetzen. Sowohl Assembler–Sprachen und Assembler als auch der produzierte Maschinencode sind an eine jeweilige Plattform (Prozessor–Architektur) gebunden.

ASCII (*Abk. für engl. 'American standard code for information interchange'*) Weit verbreiteter Standard zur Zeichensatz–Codierung aus dem Jahr 1963, der alphanumerischen Zeichen sowie einer Reihe von Sonderzeichen eindeutigen Bitmustern zuordnet und somit zwischen verschiedenen Plattformen Kompatibilität auf Zeichenebene herstellt. Die eigentliche ASCII–Spezifikation benutzt 7 Bit und verwaltet damit 128 verschiedene Bitmuster bzw. Zeichen. Es gibt international eine Reihe von Ergänzungen (ISO 8859) des ASCII–Standards zur Codierung nationaler Sonderzeichen (etwa deutscher Sonderzeichen). Über lateinische ASCII–Tabellen mit nationalen Sonderzeichen hinaus gibt es eigene Tabellen für nicht–

lateinische Zeichensätze wie z.B. hebräisch, kyrillisch, arabisch oder griechisch. Sprachen mit besonders hohem Zeichenumfang wie z.B. chinesisch werden durch die Verwendung von 16 Bit pro Zeichen codiert, was bis heute ungelöste Kompatibilitätsprobleme verursacht.

Autoren–Modus (*bzw. engl. authoring mode*) Der Modus, in dem einem Lehrenden bzw. Hypermedia–Produzenten sein Produkt während der Arbeit vom Autorensystem präsentiert wird. Dabei sind die für die Materialkomposition und –bearbeitung erforderlichen Werkzeuge zugänglich, und alle Inhalte des Produkts können dargestellt und manipuliert werden. Es ist wichtig, daß ein Lehrer bei der Herstellung von Unterrichtsmaterialien die Perspektive wechseln und sich in den ↗Lernermodus begeben kann. Zu diesem Zweck verfügen einige Autorensysteme über eine Vorschau–Funktion oder sie gestatten dem Lehrer das Anlegen eines eigenen Lerner–Accounts, von dem aus die Materialien im Lernermodus einsehbar sind.

Betriebssystem (*bzw. engl. Operating System, OS*) Sammlung von Software, die gemeinsam sowohl die Systemressourcen als auch die Prozesse, die diese Ressourcen nutzen, auf einem Computer kontrollieren. Das Betriebssystem ist eine plattformspezifische Softwareschicht zwischen der Computer–Hardware und den Nutzerprogrammen. Bekannte Betriebssysteme sind AIX von IBM, Solaris von Sun Microsystems, MacOS 9, Microsoft Windows 2000 und das ↗Open Source–Produkt Linux.

Browser (*engl. to browse: blättern, stöbern*) Programm, das Anwendern das Sichten von Datenbanken, Online–Dokumentationen oder Hypertext und die Navigation darin ermöglicht. Seit Anfang der 1990er Jahre ist Browser praktisch gleichbedeutend mit Web–Browser bzw. WWW–Browser. Diese dienen der Darstellung und Navigation von WWW–Dokumenten; sie sind ↗HTTP–↗Clients. Ein populärer WWW–Browser ist der Netscape Navigator. WWW–Browser sind in vielen Fällen als ↗Laufzeitumgebungen für Online–Lernumgebungen erforderich.

Cast (*engl. Theater– oder Film–Besetzung*) Der Begriff der Besetzung spielt in denjenigen Autorensystemen eine wichtige Rolle, deren Nutzer-

schnittstellen (insbesondere im ↗Autorenmodus) auf der Theater– oder der Film–Metapher basieren. Hierbei werden entstehende Hypermedia–Produktionen mit der Terminologie und Logik von Theater– oder Film–Produktionen behandelt, um die Verständlichkeit und die Erlernbarkeit des Autorensystems zu verbessern. Hypermedia–Elemente, –Dokumente, deren Attribute oder Verhaltensweisen werden im Rahmen dieses Vorgehens als Schauspieler behandelt, die in ihrer Gesamtheit als Besetzung verstanden werden und sich gemäß definierter „Regie–Anweisungen“ verhalten.

CGI (*Abk. für engl. 'Common Gateway Interface'*) Neben dem Servieren statischer HTML–Dokumente können WWW–Server auch den Output ausführbarer Programme an Client–Programme wie z.B. Netscape übergeben und damit dynamisch erzeugte Inhalte anbieten. Die (serverunabhängige) Schnittstelle, über die diese externen Programme aufgerufen werden, heißt CGI. CGI–Programme werden typischerweise in ↗PERL, als Shellscript oder in C programmiert. Die CGI–Schnittstelle sorgt im wesentlichen dafür, daß CGI–Programme Informationen übergeben bekommen, die Nutzer in WWW–Formulare eingeben, bevor sie sie über einen WWW–Server an ein CGI–Programm verschicken.

Client (*engl. etwa für 'Empfänger von Diensten'*) Ein System (Hardware oder Software), dem in einem Netzwerk bestimmte Dienste von sogenannten ↗Servern bereitgestellt werden (Backup–, Drucker–, E-Mail–, Dateiserver). Der Netscape Navigator ist ein Beispiel für einen WWW–Client.

Compiler Ein Übersetzungsprogramm (Systemwerkzeug), das benutzt wird, um Programme, die in einer problemorientierten, sogenannten 'Hochsprache' geschrieben wurden, in ↗Maschinencode (in seltenen Fällen zunächst in eine Assemblersprache) zu übersetzen. Der hochsprachliche Input eines Compilers, der ↗Quelltext, ist eine relativ plattformunabhängige (Prozessortyp–unabhängige) Beschreibung eines Programmablaufs (Algorithmus) und ist nicht direkt ausführbar. Der Compiler selbst sowie sein Output sind plattformabhängig. Erst der maschinensprachliche Output ist

(ggf. nach der Verlinkung mit Systembibliotheken) ein auf der jeweiligen Zielplattform ausführbares Programm. Compiler, die auf einer Plattform Output erzeugen, der auf einer anderen Plattform ausführbar ist, nennt man Cross–Compiler. Ein Beispiel für eine Compilersprache ist die Sprache C.

Content–type (*engl. etwa für 'digitaler Medientyp'*) ↗Metadaten–Element, das als Vorspann einer Datei z.B. bei deren Transfer über ein Netzwerk (etwa via ↗HTTP) deren Medientyp spezifiziert. Dies ermöglicht es dem Zielrechner, einen geeigneten ↗Client bzw. ↗Browser zur Darstellung oder Bearbeitung der Datei aufzurufen.

Convergence (*engl. für Konvergenz*) Konzept, das oft im Zusammenhang mit ↗Interoperationalität diskutiert wird. Es bezeichnet in der Informationstechnologie die semantische Äquivalenz von Daten oder ↗Metadaten in unterschiedlichen ↗Wissensdomänen. Dies ist in aller Regel jedoch keine a priori gegebene Annehmlichkeit, sondern ein (durch zentrale Spezifikation oder permanente redaktionelle Arbeit) schwer erreichbares Ziel, da sich Bedeutungen von Nutzer zu Nutzer, von Wissensdomäne zu Wissensdomäne sowie entlang der Zeitachse kontinuierlich verändern können. Wenn ohne die Autorität einer zentralen Spezifikation zwischen ↗Wissensdomänen kommuniziert werden soll, erweist sich die ↗Konvergenz–Hypothese oft sehr bald als falsch.

Curriculum (*lat. für Ablauf, Zeitabschnitt*) bezeichnet im engeren Sinn Erziehungsinhalte, die im Unterricht vermittelt werden bzw. vermittelt werden sollen. Im weiteren Sinn schließt der Begriff aber auch Methoden der Vermittlung von Inhalten sowie deren Evaluation ein. Ferner unterscheidet man Curricula nach der Verbindlichkeit ihrer Vorgaben: *Geschlossene Curricula* enthalten ausschließlich festgelegte Lehrinhalte und –methoden. Der Unterricht und seine Evaluation erfolgen nach einem vordefinierten Zeitplan. In *halb offenen Curricula* sind die Lernziele festgelegt, allerdings sind Materialien und Methoden nicht verbindlich und können z.B. an Lerner–Interessen angepaßt werden. In *offenen Curricula* sind weder

Inhalte, Methoden und Zeitpläne noch konkrete Lernziele vorgegeben. Beispiele hierfür sind Diskussionsgruppen und Lernzirkel in der Erwachsenenbildung.

Datenbank Eine Sammlung von Daten, die in einem Computersystem verwaltet wird. Im einfachsten Fall ist dies eine Datei, in der die Daten als Klartext gespeichert werden (man spricht in diesem Fall von einer *Flat–File–Datenbank*). Bei der Verwaltung von Datenbanken entstehen komplexe Probleme wie etwa die Verwaltung und Einhaltung von Zugriffsrechten, die Vermeidung von Datenverlust durch gegenseitiges Überschreiben von Daten durch mehrere Nutzer bei gleichzeitigem Manipulieren der Daten (das sogenannte *Locking*), die Auswertung von Anfragen an die Datenbank, die Datensicherung und die Zugriffsprotokollierung. Diese Probleme werden durch die Verwendung sogenannter Datenbank–Management–Systeme wie z.B. MySQL (↗Open Source), Oracle8*i* von ORACLE oder DB2 von IBM gelöst. Anfragen nach Daten in Datenbanken, die durch ein Datenbank–Management–System verwaltet werden, erfolgen typischerweise in Dialekten der Sprache SQL („seequell“, engl. für 'Structured Query Language').)

Debugging (*engl. „Entkäfern“, Fehlerbereinigung*) Die Identifizierung und Beseitigung von Fehlern, sogenannten *Bugs*, aus Hard– oder (häufiger) Software. Der Begriff geht auf die Prä–Silikon–Ära zurück, als bei der Suche nach Ursachen für Programmfehler scherzhaft Käfer in (halb)mechanischen Hardware–Elementen vermutet wurden.

Dokumentbeschreibungssprache Sprache zur Beschreibung von Inhalt und/oder Struktur von Dokumenten. Beispiele sind HTML, ↗XML und ihr gemeinsamer Vorgänger SGML.

Download Der Transfer von Daten von einem zentralen Rechner in einem Netzwerk zu einem Client–Rechner (z.B. ↗LAN oder Internet).

EDUCAUSE Eine internationale, non–profit Organisation, die sich der Entwicklung höherer Bildung durch Technologie–Einsatz widmet. Sie ist das Resultat eines Zusammenschlusses von CAUSE und Educom im Jahr 1998.

Die beiden Büros, die EDUCAUSE zur Zeit unterhält, liegen in Boulder, Colorado und in Washington, D.C. Aktuelle Informationen über Programme, Initiativen und das Dienstleistungsangebot von EDUCAUSE sind unter http://www.educause.edu zu finden.

Entwurfsproblem Aus einer Entwurfsaufgabe resultierendes Planungs– und Bearbeitungsproblem. Da Entwurfsaufgaben einzigartig sind, enthalten Entwurfsprodukte immer innovative Elemente, die nicht durch rein rationales Handeln entwickelt werden können. In dieser Hinsicht unterscheiden sich Entwurfsprobleme von Nicht–Entwurfsproblemen (auch bösartige und zahme Probleme genannt). Der Entwurf von Unterricht ist ein gutes Beispiel für ein Entwurfsproblem, das durch den höchst individuellen Charakter jeder einzelnen Unterrichtssituation immer einzigartig ist.

FAQ (*Abk. für engl. 'frequently asked questions'*) Sammlung von Fragen, die häufig zu einem Thema gestellt werden, inklusive entsprechenden Antworten. FAQs haben ihren Ursprung im Usenet, sind mittlerweile aber weiter verbreitet und z.B. eine Form der Software–Dokumentation. Siehe auch ↗HOWTO.

Feedback (*engl. Rückkopplung bzw. Rückmeldung*) Die Übertragung von Ausgangsinformationen eines Systems zu einem Eingang desselben Systems, z.B. zum Zweck der Anpassung von Ist–Zuständen an Soll-Zustände. Feedback ist in diesem engeren Sinn ein zentraler Begriff in der ↗Kybernetik und der Steuerungstechnik. In der Didaktik bezeichnet man mit Feedback jede Form von Rückmeldung auf Fragen, Arbeitsergebnisse, Wortbeiträge oder Verhaltensweisen. Die Auswertung von Feedback zu eigenen Leistungen ist hilfreiches Mittel zur Selbsteinschätzung und wird in vielen Situationen aktiv gesucht, etwa in abschließenden Diskussionen. Im Design spricht man ebenfalls von Feedback. Gemeint sind hier Rückmeldungen auf gestalterische Handlungen oder Produkte, die etwa zum Zweck der Überprüfung von Methoden– oder Materialwahl gesammelt und ausgewertet werden. Feedbackprozesse schließen einen sogenannten *Regelkreis*, der fortwährende Überwachungen und Anpassungen und damit die dyna-

mische Optimierung von Prozessen in Echtzeit ermöglicht.

Flowchart (*engl. für Flußdiagramm*) Einfache graphische Repräsentation der Struktur (des Ablaufs) eines Programms, ursprünglich entwickelt etwa im Jahr 1944 von John von Neumann und Herman Goldstine. Flowcharts bestehen aus einer Reihe von unterschiedlichen zweidimensionalen Formen, die unterschiedliche Aktionen und Entscheidungen und Ereignisse repräsentieren und durch Kanten verbunden werden, die den Programmfluß beschreiben. Sie dienen hauptsächlich dem Softwareentwurf und wurden lange Zeit (insbesondere in der Lehre strukturierter Programmierung) eingesetzt, haben aber an Bedeutung verloren. Durch ihre graphische Natur stellen Flowcharts eine unmittelbare Verbindung zwischen Softwareentwicklung und ↗GUIs her, was sie zu einem wichtigen Paradigma für die ↗visuelle Programmierung und somit zu einem Designparadigma für einige Autorensysteme macht. Für den Etwurf größerer Projekte eignen sich Flowcharts weniger, weil sie einerseits dazu tendieren den Umfang sinnvoller Visualisierungen zu überschreiten und andererseits die Strukturierung von Daten ignorieren. Damit ignorieren sie im Bereich der Lernsoftwaregestaltung z.B. die interne Formatierung von Lernmaterialien und begünstigen deren Vernachlässigung.

FTP (*Abk. für engl. 'File Transfer Protocol'*) Protokoll zur plattformunabhängigen Übertragung von Dateien über ein Netzwerk.

GUI (*Abk. für engl. Graphical User Interface*) Graphische Nutzer–Schnittstelle. Software–Oberflächen, durch die Computer z.B. durch Fenster mit der Maus bedient werden. Ein andauernder Forschungsprozeß im Bereich der Ergonomie hat zur Entwicklung einer Reihe von visuell mehr oder weniger standardisierten Eingabe–Elementen wie z.B. Pull–Down–Menüs, Checkboxen, Radiobuttons, Textbuttons etc. geführt. Die Benutzung von Computern durch ein GUI ist weniger lernintensiv als durch textuelle Schnittstellen (Kommando–Zeilen) und sie gestattet eine leichte Integration von Multimedia–Elementen in die Darstellung. Daher werden sie allgemein als nutzerfreundliche Software beschrieben. GUIs dienen nicht mehr nur als

Schnittstelle zu PCs sondern in zunehmendem Maß auch zu Mobiltelefonen, Verkaufsautomaten usw.

Hawthorne–Effekt Ein Effekt, der nach einer Untersuchung in Hawthorne im US-Staat Illionis in den 1930er Jahren benannt wurde. Er besagt, daß die bloße Beobachtung individuellen Verhaltens dieses beeinflussen kann oder allgemeiner: Alles, was neu ist, funktioniert. Es gibt eine Reihe von Ursachen für den Hawthorne–Effekt, wie z.B. den ↗Novitätseffekt. Eine klinische Anwendung des Hawthorne–Effekts ist z.B. die Nutzung des Placebo–Effekts. Der Hawthorne–Effekt verzerrt u.a. Studien über die Wirksamkeit von Unterrichtsmethoden und –medien.

HTTP (*Abk. für engl. 'Hypertext Transfer Protocol'*) Platformunabhängiges Protokoll zur Übertragung von Hypertext–Dokumenten (ggf. über Netzwerke). HTTP sowie die in dessen Kontext eingesetzte ↗Dokumentbeschreibungssprache HTML wurden um 1989 von Tim Berners Lee entwickelt und markieren als praktische Produkte neben seinem Proposal die Geburtsstunde des World Wide Web. HTTP stellt sicher, daß HTML–Dokumente vom WWW–Server zum Client übertragen werden und dieser nach dem Anklicken enthaltener Hyperlinks dynamisch eine Verbindung mit dem jeweiligen Zielserver aufbaut und das angelinkte Dokument anfordert.

HowTo (*engl. etwa für 'wie geht das?'*) Eine Dokumentationsform, die sich im Kontext des ↗Open Source–↗Betriebssystems Linux entwickelt hat. HowTos sind frei verfügbare, von Nutzern für Nutzer in einem mehr oder weniger standardisierten Hypertextformat verfaßte Dokumentationen zu sich häufig stellenden Fragen und praktischen Problemen mit Kontextinformationen, schrittweisen Anleitungen und Beispielen. Die dokumentierten Muster–Lösungen sind voneinander unabhängig und es gibt kein übergeordnetes System zu Integration oder Zusammenwirken der Inhalte einzelner HowTos. Hierin besteht wohl der wesentliche Unterschied zwischen HowTos und Mustersprachen.

Implementierung Prozeß und Tätigkeit der Herstellung von Software.

Interfacedesign (*engl. für 'Schnittstellendesign'*) Der Entwurf / die Gestaltung von Schnittstellen. In der Regel wird mit diesem Begriff auf die Mensch–Maschine–Schnittstelle, also die Oberfläche von Computersystemen und insbesondere von Software referiert.

Interoperabilität Kompatibilität in bezug auf Datenaustausch. Die Fähigkeit von Systemen, Daten ohne besonderen Aufwand gemeinsam nutzen zu können. Interoperabilität soll durch die Spezifikation von Kommunikations–Standards wie z.B. ↗XML erleichtert werden.

Interpreter Programm, das ↗Quelltext Zeile für Zeile mit einem ↗Parser analysiert und dann in Maschinencode übersetzt und sofort ausführt, anstatt wie Compiler Maschinencode für die spätere Ausführung in Dateien zu schreiben. Wegen der gleichzeitigen zeilenweisen Übersetzung ist die Ausführung von Interpretersprachen typischerweise wesentlich langsamer als die von compiliertem Maschinencode. Beispiele für Interpretersprachen sind die Sprachen ↗PERL und ↗SeL.

Konvergenz–Hypothese Annahme, daß bezeichnende Symbole in unterschiedlichenden ↗Wissens–Domänen konvergent (semantisch identisch) sind und somit eindeutige Interpretationen gestatten.

Kritischer Rationalismus Wissenschaftsverständnis, das die Positionen des ↗Naiven Empirismus und des ↗Logischen Empirismus durch das Prinzip der kritischen Prüfung von Aussagen ersetzt. Da sich auch aus vielen identischen Beobachtungen nicht deren Allgemeingültigkeit ableiten läßt, kann keine Theorie als absolut sicher betrachtet werden. Es existiert demnach kein absolutes Wissen sondern nur besser oder schlechter begründetes Wissen. Mit Sicherheit können Aussagen nur widerlegt werden.

Kybernetik (*engl. Cybernetics, ursprünglich vom griechischen Wort für Steuermann*) In den 1940er Jahren entstandene Wissenschaft von der Steuerung und Regelung von Systemen. Sie beschäftigt sich mit der Analyse, Beschreibung und Modellierung von Prozessen, die in sogenannten Regelkreisen durch ↗Feedback Soll–Zustände erreichen oder einhalten sollen.

Da auch das Lernen als gesteuerter Feedbackprozeß verstanden werden kann, hatte die Kybernetik insbesondere in der Mitte des 20. Jahrhunderts auf die Lernpsychologie und die Entwicklung von Lehrmitteln großen Einfluß. Im Vordergrund stand dabei die Fähigkeit von Unterrichtsmaterialien, umgehendes Feedback für Lerneraktivität zu liefern.

LAN (*Abk. für engl. 'local area network'*) Lokales Computernetzwerk, etwa eines Unternehmens oder einer Institution (z.B. einer Schule oder eines Universitätscampus), das z.B. durch Ethernet–Verbindungen, lokalen Funk und/oder durch Laserkommunikation zwischen Gebäuden realisiert wird.

Laufzeitumgebung (*für engl. 'runtime environment'*) Softwareumgebung, in der Software–Produkte ausgeführt werden können. Für die Rezeption von computerbasierten Lernmitteln stehen für einige Datenformate isolierte, vom Autorensystem physisch unabhängige Laufzeitumgebungen zur Verfügung. Sie erlauben die Nutzung von Lernmaterialien im ↗Lernermodus.

Lernermodus (*für engl. 'learner mode'*) Der Modus, in dem einem Lerner eine computerbasierte Lernumgebung vom Autorensystem bzw. dessen Laufzeitumgebung präsentiert wird. Im Gegensatz zum ↗Autorenmodus ist nur eine definierte Menge von Aktionen möglich und eine begrenzte Menge von Inhalten zu unterschiedlichen Zeitpunkten sichtbar. Während ein Lehrer im Autorenmodus z.B. die korrekten Antworten auf Testfragen jederzeit einsehen und manipulieren kann, muß die Lernumgebung diese Informationen weitgehend vor dem Lerner verbergen.

Logischer Empirismus Wissenschaftsverständnis, das die Grundannahmen des ↗Naiven Empirismus dahingehend erweitert, daß es Sätze der modernen formalen Logik als analytisches Wissen zuläßt. Es bleibt das Problem, daß von einer endlichen Menge an Beobachtungen allgemeingültige Gesetze bzw. Aussagen abgeleitet werden sollen. Darauf reagiert der ↗Kritische Rationalismus.

Mainframe (*engl. zentraler 'Großrechner'*) Die Arbeit auf einem Mainframe erfolgt in der Regel über ein Netzwerk von einem ↗Terminal aus. Seit der

Verfügbarkeit von Personal Computern (PC) kann praktisch auf jedem Schreibtisch so viel billige Rechenleistung bereitgestellt werden, daß Mainframes ihre einstige Bedeutung weitgehend verloren haben. Sowohl zentrale (mainframebasierte) als auch dezentrale (PC–basierte) Verfügbarkeit von Rechenkapazitäten haben spezifische Vor– und Nachteile. Daher ist keine der beiden Architekturen generell „besser“ oder „schlechter“, und in der EDV dauern entsprechende Debatten und widersprüchliche Erfolgsvorhersagen bis heute an. An Schulen und Universitäten werden zentrale Großrechner heute etwa als Applikations– oder Backupserver oder zur Bearbeitung sehr rechenintensiver Aufgaben etwa in den Bereichen Simulation und Visualisierung eingesetzt. Hier steht das Bedienen von Arbeitsplatz–Terminals nicht im Vordergrund.

Maschinencode Direkt durch einen Prozessor ausführbarer, für (normale) Menschen nicht lesbarer Programmcode. Der (plattformabhängige) Output von ↗Assemblern und vieler ↗Compiler.

Metadaten (*engl. metadata*) Metadaten sind Daten über Daten. Ihre Elemente sind Container–Wert–Paare wie z.B. „author: Thomas Fischer“, „keywords: authoring systems, online teaching, design“. Sie sind das elektronische Äquivalent zu Systematiken von Bibliotheken und ein wichtiger Schritt in Richtung ↗Interoperabilität. Eine wichtige Metadaten–Spezifikation für das World Wide Web ist das „Dublin Core Set“. Eine Metadaten–Spezifikation für digitale Lernmaterialien ist das „IMS Metadata XML schema“ von ↗EDUAUSE.

Naiver Empirismus (auch Empirismus) Wissenschaftsverständnis (1. Hälfte des 20. Jahrhunderts), das davon ausgeht, daß alle Erkenntnis aus der Erfahrung kommt. Dem liegen die folgenden beiden zentralen Annahmen zugrunde: Es existieren objektive Tatsachen, die sich durch Beobachtung und Experiment erfassen lassen. Auf der Basis einzelner Beobachtungen lassen sich allgemeingültige (Natur–)Gesetze formulieren. Diese Position wurde später durch den ↗Logischen Empirismus und den ↗Kritischen Rationalismus abgelöst.

Novitätseffekt Bei frühen Kontakten mit Inhalten, Methoden oder Medien kurzfristig auftretender Motivationseffekt. Kontakte mit „Neuartigem“ sind im Bereich der Unterrichttechnologie recht wahrscheinlich, da hier regelmäßig neue Verfahren und Medien entwickelt werden. Das gesteigerte Interesse an diesen Verfahren und Medien führt oft zu einer Verbesserung der Lernleistung, die aber bald abklingt. Der Novitätseffekt kann von Lehrenden als Instrument eingesetzt werden, um kurzfristig das Lernerinteresse zu erhöhen. Er ist oft ein Störfaktor in Untersuchungen der Unterrichtsforschung. Hier lassen sich Effekte der Lehrwirksamkeit neuer Verfahren zumeist nicht vom Novitätseffet trennen (siehe auch ↗Hawthorne–Effekt).

Open Source (*engl. 'öffentlicher ↗Quelltext'*) Gemeinsamer Grundsatz einer weltweit verteilten Entwicklergemeinde, die über das Internet kommuniziert und freie Software mit frei verfügbaren ↗Quelltexten herstellt. Dabei bezieht sich *frei* nicht nur darauf, daß Open Source–Software gratis ist, sondern auch auf deren Freiheit gegenüber anderen negativen Auswirkungen kommerzieller Software (z.B. im Hinblick auf Kompatibilität). Es gibt eine Reihe sogenannter „Open Source–Lizenzen“, durch die einerseits die freie Verfügbarkeit und andererseits der Fortbestand freier Software gesichert werden soll, z.B. die „BSD License“ und die „GNU General Public License“ (GPL). Mit der Veröffentlichung von Open Source Software geht die Aufforderung zur Nutzung und Weiterentwicklung der Software einher, wobei durch die genannten Lizenzen die Erwähnung der ursprünglichen Autoren gefordert wird. Die im praktischen Teil dieser Arbeit eingesetzten Produkte ↗PERL und ↗Apache sowie die Software, mit denen dieser Text hergestellt wurde, Linux, LaTeXund Emacs, sind Open Source–Projekte. Das *Entwicklungsmodell* Open-Source ist so wie auch ↗Unterrichts–Modelle eine „Rekonstruktion von Wirklichkeit“. Daher ist es hilfreich als retrospektive Beschreibung von Prozessen und Produkten, als prospektives Planungkonzept liefert es allerdings weder verbindliche, konkrete Handlungsanweisungen noch eine Garantie für Erfolg.

Operationalisierung (auch „Problemzähmung“) In didaktischen Kontexten die

Übersetzung und Aufgliederung von (Lern–) Zielen in beobachtbare Tätigkeiten, konkrete Verhaltensweisen oder meßbare Leistungen zum Zweck der Überprüfung von (Lern–) Leistungen.

Parsing Syntaktische Analyse. In diesem Prozeß muß das Programm, das das Parsing durchführt (der sog. *Parser*), entscheiden, ob eine Sequenz von Input–Symbolen der Grammatik einer definierten Sprache bzw. Spezifikation entspricht und daraufhin ggf. eine Zerlegung des Inputs in seine Atome durchführen, um ggf. eine anschließende inhaltliche Auswertung zu ermöglichen.

PERL (*Abk. für engl. 'Practical Extraction and Replacement Language'*) Seit 1986 von Larry Wall entwickelte ↗Interpreter–↗Scriptsprache für die System– und Netzwerkadministration. PERL hat sich besonders durch seine Eignung als ↗CGI–Sprache weit verbreitet. PERL ist ein ↗Open Source–Produkt, für das für unterschiedlichste Anwendungsbereiche frei verfügbare Bibliotheken (sogenannte *Module*) bereitstehen, mit denen bestimmte Problemlösungen wesentlich leichter zu erreichen sind.

Quelltext (*auch ‚Quellcode' oder engl. 'source code'*) Die Form, in der ein Programm für Menschen (mehr oder weinger) lesbar ist. Der Input von ↗Compilern und ↗Interpretern wird als Quellcode bezeichnet. Liegt ein Programm im Quellcode vor, ist es vergleichsweise leicht zu analysieren und zu modifizieren. Wurde es mit einem Compiler in Maschinencode übersetzt, ist es nicht mehr ohne weiteres lesbar. Wird ein Programm nicht im Quellcode, sondern lediglich als compilierter Maschinencode weitergegeben, ensteht ein automatischer Schutz vor ungewünschter Analyse und Modifikation, was Vorteile in sicherheitskritischen oder kommerziellen Kontexten, aber Nachteile in bezug auf individuelle Erweiterungen und problemorientierte Anpaßbarkeit mit sich bringt. Die ↗Open Source–Bewegung stellt eine Reaktion auf die daraus resultierenden Probleme dar.

Rezept Synonym für Muster, Musterlösung oder Pattern. Der Begriff wird in der Didaktik verwendet und entspricht dort praktisch dem Pattern–Begriff bei Alexander oder Gamma. Zunächst hatte der Rezept–Begriff

eine negative Bedeutung, da mit ihm lediglich didaktische Zweckrationalisierung auf Kosten individueller Unterrichtsgestaltung verbunden wurde. In jüngerer Zeit wandelt sich dieses Verständnis insbesondere durch die Diskussion des Rezept–Begriffs von Grell und Grell, die die grundsätzliche Notwendigkeit von Unterrichtsrezepten und deren didaktisches Potential, (verantwortungs–)bewußter Einsatz vorausgesetzt, aufzeigen.

Scriptsprachen (*engl. 'scripting languages'*) sind Programmiersprachen, mit denen Software–Anwendungen programmiert werden können. So können dynamische Elemente in Webseiten in Scriptsprachen programmiert werden, die von Browsern unterstützt werden (z.B. JavaScript). Scriptsprachen werden typischerweise von ↗Interpretern verarbeitet. Scriptsprachen von Autorensystemen sind z.B. HyperTalk, AppleScript und Lingo. Auch die Autorensprache SeL sowie die Sprachen PERL, Python und Tcl können als Scriptsprachen bezeichnet werden.

SeL (*Abk. für engl. 'The SeSAMe Language'*) An der Gesamthochschule Universität Kassel entwickelte Autorensprache für das Online–Autorensystem SeSAMe. Das syntaktische Modell von SeL integriert eine erweiterbare logische ↗Dokumentbeschreibungssprache mit einer prozeduralen Sprache zur Zeichenverarbeitung. Der SeL–↗Interpreter wird in WWW–Anwendungen als Erweiterung des WWW–↗Servers betrieben.

Server (*engl. für 'Servierer'*) Ein System (Hardware oder Software), das (etwa in einem Netzwerk) bestimmte Dienste für ↗Clients bereitstellt (Backup–, Drucker–, Dateiserver). Der Apache Server ist ein Beispiel für einen ↗Open Source WWW–Server.

Shell–Account Zugangsberechtigung zu einer *Shell*, also dem textuellen Kommando–Interface zu einem ↗Betriebssystem. Für den Zugang zu einer Shell ist auf Multiuser–Systemen eine sogenannte Login–Prozedur erforderlich, bei der der Nutzer seine Nutzerkennung und ein Paßwort angeben muß, um Zugang zu erhalten.

Stufenmodelle Prospektive Handlungsmodelle, die Prozesse (z.B. Entwurfs– oder Unterrichtsprozesse) als Sequenz einzelner Stufen, Stadien oder

Tätigkeiten beschreiben. Strengere Stufenmodelle wie das sogenannte „Wasserfallmodell“ in der Informatik beschreiben endliche Anzahlen von Stufen und bestehen auf die Einhaltung einer definierten Reihenfolge dieser Stufen (was im Kontext von Entwurfsaufgaben problematisch ist). Es gibt auch weniger verbindliche Stufenmodelle wie z.B. die Aussage „Entwerfen ist das Herstellen und das Verwerfen von Varianz“[1].

Terminal Nutzer–Konsole, im einfachsten Fall bestehend aus Monitor und Tastatur, über die Rechenleistung (z.B eines zentralen ↗Mainframes) am Arbeitsplatz verfügbar gemacht wird. Die Verbindung erfolgt über ein Netzwerk oder eine direkte serielle Verbindung (auch via Telefonleitung). Man unterscheidet sogenannte *intelligente Terminals*, die über eingebaute Fähigkeiten zum Speichern und Manipulieren von Daten verfügen und *dumme Terminals* ohne entsprechende Möglichkeiten. Terminals haben seit der Verfügbarkeit von PCs an Bedeutung verloren, jedoch gibt es heute wieder Ansätze zum Einsatz von Mainframe–Termninal–Architekturen. Ein neuerer Begriff für Terminal ist *thin client*.

Transklusion (oder engl. 'transclusion') Dieser Begriff wurde (wie auch der Begriff „Hypertext“) von Theodor H. Nelson geprägt. Er setzt sich zusammen aus „transparent“ und „inclusion“ und meint die Echtzeit–Einbindung verteilter Hypertextfragmente in kohärente Dokumente. Transklusion ist eine zentrale Eigenschaft vom Nelsons *Xanadu*–System, das auf diese Weise z.B. das Kopieren von Informationen überflüssig macht und damit ursprünglich helfen sollte, Urheberrechte und „intellektuelles Eigentum“ zu schützen. XANADU konnte sich nicht gegen das WWW durchsetzen.

Turing–Maschine Von Alan Turing im Jahr 1936 entworfene hypothetische Maschine bestehend aus einem magnetischen Schreib– Lesekopf und einem idealerweise unendlich langen Magnetband. Das Magnetband ist in Zellen

[1] Dieses Sufenmodell hat Prof. Hans Dehlinger in seiner Vorlesung „Einfürung in die Grundlagen und Methoden des Planens und Entwerfens I und II“ 1998/99 am Fachbereich Architektur der Universität Gesamthochschule Kassel diskutiert — die Originalquelle ist mir unbekannt

unterteilt, die als Speicherzellen dienen und eines von zwei Symbolen (z.B. 0 und 1) repräsentieren können. Der Schreib–Lesekopf kann den Inhalt jeder Zelle sowohl auslesen als auch ändern. Die Maschine kann programmiert werden, indem für unterschiedliche Zustände (gelesene Symbole) unterschiedliche Verhaltensweisen (Bewegen des Magnetbandes entlang des Schreib– Lesekopfes, Änderung von Symbolen) definiert werden. Die dabei erforderlichen *bedingten Sprünge* wurden in der Paxis elektronischer Rechenmaschinen erst etwa ein Jahrzehnt später durch John von Neumann (mit der sogenannten von–Neumann-Architektur, auf der unsere heutigen Computer basieren) realisiert. Die Symbole auf dem Magnetband können als *Daten* und die definierten Verhaltensweisen als *Programm* bezeichnet werden. Mit der *universellen Turingmaschine* entwirft Turing einen weiter entwickelten Apparat, bei dem die Verhaltensweisen (das Programm) der Maschine auf einem zweiten Magnetband, ebenfalls in einem dualen Symbolsystem codiert, verarbeitet werden. Die Behandlung von Programm und Daten ist somit symmetrisch, wodurch deren Austauschbarkeit illustriert wird. Die universelle Turingmaschine ist in der Lage, beliebige beschreibbare mentale Prozesse nachzubilden und das Verhalten jeder anderen Maschine zu imitieren.

Turing–Test Wie auch die ↗Turing–Maschine ein Gedankenexperiment des britischen Mathematikers Alan Turing (im Bereich der Künstlichen Intelligenz). Es handelt sich hierbei um ein etwa 1944 entworfenes Imitationsspiel, von dem es eine Reihe späterer Varianten gibt. Hierbei geht es im Wesentlichen darum, in einem (schriftlichen) Dialog mit einem verborgenen Menschen und/oder einer verborgenen Maschine zu erraten, ob es sich um eine Maschine oder um einen Menschen (bzw. eine Frau oder einen Mann) handelt, wobei diese verborgenen Dialogpartner unterschiedliche Strategien in bezug auf die Ehrlichkeit ihrer Antworten anwenden können. Eine erfolgreiche Irreführung eines menschlichen Urteilenden durch eine Maschine könnte laut Turing als Zeichen oder Maßstab für Intelligenz dienen. Turing diskutiert diesen Test im Zusammenhang mit der Frage, ob Maschinen prinzipiell in der Lage sind denken zu können (lange be-

vor elektronische Rechner oder der Begriff „Künstliche Intelligenz“ überhaupt existieren. Der Test wird später zu einem Bewertungsmaßstab für viele Wettbewerbe im Bereich der Künstlichen Intelligenz. Hier zeigt sich jedoch, daß Maschinen insbesondere dann *überzeugende Menschen* sind, wenn sie die nicht perfekt handeln (und ihre Antworten z.B. Verzögerungen und Tippfehler enthalten). Praktisch stellen Turing–Tests folglich eher Tests für Imitations–Fähigkeit als für Intelligenz dar. Obgleich es auf den ersten Blick erscheint, als liefere Turing mit seinem Test einen formalen Bewertungsmaßstab für die bösartige Frage nach der Herstellbarkeit Künstlicher Intelligenz, bleibt hier anzumerken, daß jedes konkrete Ergebnis eines Turing–Tests die Einschätzung eines Individuums in einer einzigen Situation und keine echte Problemzähmung darstellt.

Unterrichts–Durchführung Die praktische Anwendung des ↗Unterrichts–Entwurfs, in der Lernende sich mit den jeweiligen Unterrichts–Inhalten auseinandersetzen. Im traditionellen (frontalen) Klassenunterricht erfolgt die Unterrichts–Durchführung als direkte Kommunikation zwischen Lehrenden und Lernenden. Diese Konstellation wird durch neue Unterrichtsmedien wie Video und insbesondere computerbasierte Lerndialoge zunehmend aufgebrochen und die Aktivität Lehrender in der Unterrichts–Durchführung sinkt tendenziell gegenüber den steigenden Verantwortungen, die sich im Rahmen der ↗Unterrichts–Planung ergeben.

Unterrichts–Entwurf Auch Unterrichtsgestaltung. Das Ergebnis der Unterrichts–Vorbereitung: Vom Lehrenden angefertigter geistiger und/oder verschriftlichter Plan für den Verlauf einer Unterrichtsveranstaltung. Hierbei werden Methoden, Hilfsmittel, Interaktionsformen sowie deren zeitliche Strukturierung und letztlich deren Beitrag zum Erreichen definierter Lernziele beschrieben. Zwar werden (insbesondere in der Lehrer–Ausbildung) gängige Formate verschriftlichter Unterrichts–Entwürfe angewendet, und bestimmte Inhalte und Ziele legen das Aufgreifen bestimmter ↗Unterrichts–Modelle nahe, objektive Aussagen nach dem Muster „Der Unterrichts–Entwurf X ist *richtig* oder *falsch*“ sind jedoch sehr problematisch. Akzeptabel sind persönliche Urteile nach dem Muster „Ich

halte Unterrichtsentwurf X *besser* als Unterrichtsentwurf Y weil Z“. Siehe ↗Entwurfsproblem und ↗Turing–Test.

Unterrichts–Modell (auch didaktisches Modell) Typisierte bzw. kategorisierte Beschreibung eines unterrichtlichen Konzepts. Beispiele für didaktische Modelle sind Frontalunterricht, Kleingruppen–Lerngespräch, Lernausstellung oder Werkstattseminar. Diese Modelle stellen laut Flechsig [18] *Rekonstruktionen von Unterrichtswirklichkeit* dar und haben als solche im wesentlichen rückblickende Qualitäten. Für prospektives unterrichtliches Planen liefern sie weder verbindliche, konkrete Handlungsanweisungen noch Garantien für Erfolg.

Unterrichts–Planung Bedingt gleichbedeutend mit ↗Unterrichtsentwurf. Dieser Begriff wird jedoch tendenziell eher im Zusammenhang mit offeneren ↗Curricula, etwa in der Erwachsenenbildung, verwendet. Hier impliziert Unterrichtsplanung weniger bindende Entwürfe und höheren Einfluß Lernender aus Methoden und Inhalte des Unterrichts.

Upload Der Transfer von Daten von einem Client–Rechner zu einem zentralen Rechner in einem Netzwerk (z.B. ↗LAN oder Internet).

Utopie Eine unerfüllbare und daher letztlich wirklichkeitsfremde Vorstellung von einer besseren Zukunft. Utopien sind Gedankenexperimente und Zukunftsentwürfe und somit Antrieb und Schlüssel für Innovation in gesellschaftlichen Systemen. Der Begriff stammt aus der Aufklärung und bezeichtete damals eine ideale politische und moralische zukünftige Situation. Als wichtige Voraussetzung für Veränderung, die schließlich zu einer Utopie führen soll, wird die Erziehung zur Verantwortlichkeit gesehen. In den vergangenen Jahrzehnten rückte zunehmend auch die Technologie und der verantwortungsvolle Umgang mit Technologie ins Blickfeld. Utopien sind Bestandteil bedeutender philosophischer Erziehungskonzepte und ebenso ein wichtiger Antrieb für viele Designprozesse. Das Gelingen von Unterricht oder von irgendeinem anderen Design ist nicht vorhersehbar, aber typischerweise moralisch reflektiert. Daher können Utopien als Antrieb für jede unterrichtliche und entwerferische Handlung verstanden

werden.

Visuelle Programmierung Die Verwendung einer visuellen Programmiersprache. Der Begriff *visuelle Programmierung* ist in der Informatik nicht sehr weit verbreitet und der Begriff der *visuellen Programmiersprache* blieb bislang weitgehend undefiniert. Schiffer[2] zitiert eine Reihe von Definitionen für die visuelle Programmierung, die die Verwendung visueller graphischer Elemente im Programmierprozeß als zentrales Charakteristikum nennen und damit die visuelle Programmierung von der textuellen Programmierung abgrenzen. Als visuelle Elemente werden Piktogramme, graphische Symbole, Zeichnungen und Gesten genannt. Da ↗Interpreter und ↗Compiler für textuelle Programmiersprachen ihren Input lediglich als eindimensionalen Datenstrom erhalten, können mit visueller Programmierung alle 2– und 3–dimensionalen Formen der Programmierung bezeichnet werden. Die Bereitstellung *visueller Entwicklungsumgebungen* für an sich textuelle Programmiersprachen gestattet die visuelle Programmierung in textuellen Sprachen.

VRML (*Abk. für engl. 'Virtual Reality Modelling Language'*) ↗Dokumentbeschreibungssprache zur Programmierung virtueller drei–dimensionaler Welten.

WAP (Abk. für engl. 'Wireless Application Protocol') Auf ↗XML basierendes Protokoll für die Übertragung von Hypertexten und kleinen Grafiken von WWW–Servern zu drahtlosen Kleingeräten, in der Regel Mobiltelefonen.

Weltausschnitt Mit Weltauschnitt bezeichne ich einen Teil der natürlichen Welt, der für ein Problem und seine Lösung relevant ist (oder als relevant dafür eingeschätzt wird). Die darin enthaltenen Elemente und deren Beziehungen untereinander können in Software nachmodelliert werden, woruch der Weltausschnitt simuliert und darin zu lösende Probleme mit Hife des Compuers gelöst ggf. werden können. Bei diesem Prozeß drohen Fehler nicht nur bei der algorithmischen Modellierung den Weltausschnitts son-

[2] Schiffer, Stefan: [54], S. 37 ff.

dern bereits bei dessen Identifikation und Abgenzung von nicht–relevanten Teilen der natürlichen Welt.

Wissens–Domäne Dieser Begriff bezeichnet in der Künstlichen Intelligenz die Menge der Elemente eines ↗Weltausschnitts sowie die Fähigkeit, diese zu interpretieren. Eine Wissens–Domaäne verfügt in bezug auf einen Weltausschnitt über einen semantischen Konsenz, der idealerweise die korrekte Intepretation von Symbolen gestattet.

WYSIWYG (*Abk. für engl. 'what you see is what you get'*) Ein im Jahr 1969 von Flip Wilson geprägter Begriff, der im Zusammenhang mit dem Design von ↗GUIs die Eigenschaft von Software bezeichtet, Grafiken und Text annähernd exakt so darzustellen, wie sie als Ausdruck auf Papier erscheinen. Diese Eigenschaft wird insbesondere bei Textverarbeitungen und Systemen für das Desktop–Publishing als besonders nutzerfreundlich angesehen.

XML (*Abk. für engl. 'eXtendible Markup Language'*) ↗Dokumentbeschreibungssprache, die auf SGML (*standard generalized markup language*) basiert und nach bzw. neben HTML 4.0 als neuer Standard für das World Wide Web propagiert wird. In XML–Dateien werden variable Elemente von Dokumenten in verschachtelten Hierarchien ausgezeichnet. Mit Hilfe zusätzlich zu programmierender *Stylesheets*, die ebenfalls an den ↗Browser übertragen werden, werden die visuellen Attribute von XML–Daten auf der ↗Client–Seite interpretiert. XML wird auch für Messaging–Dienste empfohlen. Es zeichnen sich Mängel ab in bezug auf Kompatibilität und breite Unterstützung durch verschiedene Client–Hersteller sowie in konzeptioneller Hinsicht (↗Konvergenz–Hypothese). Für Online–Autoren hat XML bislang kaum spürbare Vorteile gebracht, und der Erfolg von XML bleibt abzuwarten.

Literaturverzeichnis

[1] Alexander, Christopher, Sara Ishikawa und Murray Silverstein: *Eine Muster-Sprache.* Löcker Verlag, Wien, 1995.

[2] Astleitner, Hermann, Jörg Sams und Josef Thonhauser: *Womit werden wir in Zukunft lernen? Schulbuch und CD–ROM als Unterrichtsmedien — Ein kritischer Vergleich.* Pädagogischer Verlag, Wien, 1998.

[3] Berners-Lee, Tim: *Information Management: A Proposal.* URL: http://www13.w3.org/History/1989/proposal.html, 1989. am 14.07.2000.

[4] Bodendorf, Freimut: *Computer in der fachlichen und universitären Ausbildung.* R. Oldenbourg Verlag GmbH, München, 1990.

[5] Braitenberg, Valentino: *Vehicles. Experiments in Synthetic Psychology.* MIT Press, Cambridge, Massachusetts, 1984.

[6] Card, Stuart K., Jock D. Mackinlay und Ben Shneiderman (Hrsg.): *Readings in Information Visualization. Using Vision to Think.* Morgan Kaufmann Publishers, San Francisco, California, 1999.

[7] Ceccato, Cristiano, Lorne Falk und Thomas Fischer: *The Silk Road: An Interactive Online Encyclopedia as a Foundation for Networked Education in Design.* SIGRADI2000 (Conference Proceedings), 2000.

[8] Conklin, E. Jeffrey und William Weil: *Wicked Problems: Naming the Pain in Organizations.* Guidon, Raymonde: Designing the Design Process. In: Human-Computer-Interaction, 5:305–344, 1990.

[9] Dery, Mark (Hrsg.): *Flame Wars. The Discourse of Cyberculture.* Duke University Press, London, 1994.

[10] Eastlake, Donald E.: *A.I. Lab Memo No. 238.* „ITS Status Report“ (Massachusetts Institute of Technology); In: Levy, Stephen: Hackers. Heroes of the Computer Revolution. (Dell Publishing New York, 1984), April 1972.

[11] Eckel, Bruce: *Thinking in Java. The Definitive Introduction to Object–Oriented Programming in the Language of the World Wide Web.* Prentice–Hall Inc., New Jersey, 1998.

[12] Falk, Lorne, Cristiano Ceccato, Catherine Hu, Philip Wong und Thomas Fischer: *Towards a Networked Education in Design. A first Manifestation through the 'Virtual Design Company' Studio.* CAADRIA2000 (Conference Proceedings), Seiten 157–167, 2000.

[13] Fischer, Thomas: *Hypertext und Hypertextmedien. Strukturen und didaktisches Potential.* Examen zur Erlangung des Ersten Staatsexamens für das Lehramt an Gymnasien, Fachbereich Erziehungs– und Humanwissenschaften, Universität Gesamthochschule Kassel, 1996.

[14] Fischer, Thomas: *Dynamic Style Processing with SeSAMe.* EUNIS'99: Information Technology shaping European Universities (Conference Proceedings), Seiten 67–72, 1999.

[15] Fischer, Thomas: *Vorschlag zur Implementierung von Transklusionen in SeL/SeSAMe.* nicht veröffentlichtes Diskussionspapier, GhK, März 2000.

[16] Fischer, Thomas, Mark Burry und Robert Woodbury: *Object–Oriented Modelling Using XML in Computer–Aided Architectural and Educational CAD.* CAADRIA2000 (Conference Proceedings), Seiten 145–155, 2000.

[17] Fischer, Thomas, Christiane Herr und Cristiano Ceccato: *Towards Real Time Interaction Visualization in NED.* Catholic University, Washington D.C., 2000.

[18] Flechsig, Karl-Heinz: *Kleines Handbuch didaktischer Modelle.* Zentrum für didaktische Studien e.V., Göttingen, dritte, überarbeitete Auflage, 1991.

[19] Frazer, John: *An Evolutionary Architecture.* Architectural Association, London, 1995.

[20] Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley Longman, Inc., Reading, Massachusetts, 1995.

[21] Gelernter, David: *The Aesthetics of Computing.* The Orion Publishing Group, Orion House, London, 1998.

[22] Greenspun, Philip: *Datenbankgestützte Web-Sites.* Carl Hanser Verlag, München und Wien, 1998.

[23] Grell, Monika und Jochen Grell: *Unterrichtsrezepte.* Beltz Verlag, Weinheim und Basel, 1991.

[24] Horowitz, Ellis: *An Integrated System for Creating Educational Software.* Perspectives in Computing, 8(1):35–42, Spring 1988.

[25] Issing, Ludwig J. und Paul Klimsa (Hrsg.): *Information und Lernen mit Multimedia.* Psychologie Verlags Union, Weinheim, 1995.

[26] Issing, Ludwig J. und K. Tober: *Autorensysteme für die Entwicklung computerunterstützter Lernprogramme.* Arbeitsbereich Medienforschung, Institut für Psychologie, Freie Universität Berlin, 1988.

[27] Jank, Werner und Hilbert Meyer: *Didaktische Modelle. Grundlegung und Kritik.* Universität Oldenburg, Oldenburg, 1990.

[28] König, Eckard und Peter Zedler: *Theorien der Erziehungswissenschaft. Einführung in Grundlagen, Methoden und praktische Konsequenzen.* Belz / Deutscher Studienverlag, Weinheim, 1998.

[29] Laurel, Brenda (Hrsg.): *The Art of Human–Computer Interface Design.* Addison-Wesley Publishing Company Inc., Reading, Massachusetts, 1990.

[30] Laurel, Brenda: *Computers as Theatre.* Addison-Wesley Publishing Company Inc., Reading, Massachusetts, 1993.

[31] Lilly, Susan: *Patterns for Pedagogy.* Object Magazine, 1:95–96, 1996.

[32] Mager, Robert F.: *Goal Analysis.* Pitman Learning, Belmont, California, zweite Auflage Auflage, 1984.

[33] Maruyama, Hiroshi, Kent Tamura und Naohiko Uramoto: *XML and Java. Developing Web Applications.* Addison Wesley, Reading, Massachusetts, 1999.

[34] Matthias, Andreas und Thomas Fischer: *SeL: Benutzerhandbuch, Programmiererhandbuch, Referenz.* URL: http://www.uni-kassel.de/design2/Handbook/DE/contents.html. am 11.07.2000.

[35] Mead, Syd: *Sentinel. Steel Couture–Syd Mead–Futurist.* Dragon's Dream, Hendrik-Ido-Ambacht, Netherlands, erste Auflage Auflage, 1979.

[36] Meyer, Hilbert: *Leitfaden zur Unterrichtsvorbereitung.* Cornelsen Verlag, Frankfurt am Main, zehnte Auflage Auflage, 1991.

[37] Minsky, Marvin: *Why People Think Computers Can't.* AI Magazine, 3(4), 1982.

[38] Mitchell, William J.: *Three Paradigms for Computer–Aided Design.* Automation in Construction, 3:239–245, 1994.

[39] Nielsen, Jakob: *Hypertext & and Hypermedia.* Academic Press, Cambridge, Massachusetts, 1990.

[40] Nolan, Andrew J.: *Learning from Success.* IEEE Software, 16(1):97–105, January/February 1999.

[41] Olszewski, Pawel: *Multimedia Authoring Systems FAQ Version 2.23.* URL: http://www.tiac.net/users/jasiglar/MMASFAQ.HTML. am 05.01.2000.

[42] O'Reilly: *kurz & gut: Open Source.* O'Reilly & Associates, Köln, 1999.

[43] Pam, Andrew: *Methods for Implementing Transclusion of Text into HTML.* URL: http://xanadu.com.cu/xanadu/transclude.html. am 12.07.2000.

[44] Pam, Andrew: *The GNU Project.* URL: http://www.gnu.org/gnu/thegnuproject.html. am 30.12.2000.

[45] Papert, Seymour: *Die Revolution des Lernens.* Heise Verlag, Hannover, 1994.

[46] Postman, Neil: *The End of Education. Redefining the Value of School.* Vintage Books, New York, erste Auflage, 1995.

[47] Protzen, Jean-Pierre: *The Poverty of the Pattern Language.* Design Methods and Theories, 12(3/4):191–194, 1978.

[48] Ramo, Simon: *A New Technique of Education.* Engineering and Science, XXI:17–22, Oktober 1957.

[49] Raymond, Eric S.: *The New Hacker's Dictionary.* MIT Press, dritte Auflage, 1998.

[50] Raymond, Eric S.: *The Cathedral and the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly & Associates, Sebastopol, 1999.

[51] Rittel, Horst W.J. und Melvin M. Webber: *Dilemmas in a General Theory of Planning.* Institute of Urban & Regional Development. University of California, Berkeley, 1973.

[52] Rode, Mary und Jim Poirot: *Authoring Systems — Are they used?* Journal of Research on Computing in Education, Winter:191–198, 1989.

[53] Salichs, M.A. und L. Moreno: *Navigation of Mobile Robots: Open Questions.* Robotica, 18:227–234, 2000.

[54] Schiffer, Stefan: *Visuelle Programmierung. Grundlagen und Einsatzmöglichkeiten.* Addison-Wesley Verlag, Bonn u.a., 1998.

[55] Schulmeister, Rolf: *Grundlagen hypermedialer Lernsysteme: Theorie-Didaktik-Design.* Addison-Wesley Publishing Company Inc., Reading, Massachusetts, 1996.

[56] Stephenson, Neil: *In the Beginning was the Command Line.* Avon Books, New York, 1999.

[57] Turing, Alan: *Computing Machinery and Intelligence.* Mind, 59, 1950.

[58] Walter, Hellmuth: *Einführung in die Unterrichtsforschung.* Wissenschaftliche Buchgesellschaft, Darmstadt, 1977.

[59] Walter, Hellmuth und Inge Edelmann: *Pragmatische Unterrichtsplanung. Theorie und Praxis psychologischer Unterrichtsvorbereitung.* Georg Westermann Verlag, Braunschweig, 1979.

[60] Witzenbacher, Kurt: *Praxis der Unterrichtsplanung.* Oldenburg Verlag, München, 1994.