Quo vadis, Informatik?
Aspekte einer objektorientierten Entwurfslehre

Klaus Quibeldey-Cirkel*

Hat die junge Informatik ihren Charme schon verloren, oder ist die Rezession schuld daran, daß die Neueinschreibungen für Informatik-Studiengänge stark rückläufig sind? Folgt der Faszination nun der Frust? Keineswegs — neue Themen faszinieren heute. Der Reiz geht vom *Objekt* aus! Die universitäre Lehre tritt auf den Plan: Langfristiges Ziel ist eine objektorientierte Entwurfslehre. Sie vereint den wissenschaftlichen Generalisten mit dem professionellen Spezialisten.

„Algorithms Considered Harmful"

Die Softwarekrise der 60er Jahre war hausgemacht: Edsger Dijkstra führte sie auf die damals beliebte, heute berüchtigte Sprunganweisung zurück — „Goto Statement Considered Harmful". Auch die Softwarekrise der 90er Jahre, im Informatik-Duden bereits als Krise der Wiederverwendung vermerkt, ist hausgemacht: Als „schädlich" für den Entwurf komplexer Systeme gilt das *algorithmische Denken* an sich. Informatik ist heute weit mehr als die Lehre vom Programmieren. Die Wirthsche Gleichung „Programmieren = Entwerfen von Algorithmen und Datenstrukturen" verkürzt die universitären Lehrinhalte auf unzulässige Weise. Auch an den allgemeinbildenden Schulen wird die Umorientierung gefordert: „Algorithmen — wozu?", fragt Rüdeger Baumann in LOG IN, der Zeitschrift für Informatik-Didaktik [Bau94].

Woher stammen die Anforderungen an ein Programm? Vom Auftraggeber, dem Kunden. Dessen Sprachwelt ist aber nur bedingt die Sprachwelt des Programmierers. Der besitzt in der Regel das „linguistische Monopol" auf die Programmspezifikation. Die restriktive Wirkung der Sprache haben bereits Ludwig Wittgenstein und Benjamin Whorf betont: „Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt" [Wit73] und „Languages shape the way we think, and determine what we can think about" [Who74]. Die Fachsprache der Anforderungen in die Sprache der Informatik zu übersetzen, kommunikativ mit dem Kunden, das ist die eigentliche Ingenieurleistung. Der Entwurf wissenstechnischer Systeme braucht andere Wissensformen als Algorithmen, Programmablauf- und Datenflußpläne. Algorithmisches Denken schließt den Kunden vom Entwurfsprozeß aus!

*Dr.-Ing. Klaus Quibeldey-Cirkel ist wissenschaftlicher Mitarbeiter in der Fachgruppe Technische Informatik der Universität Siegen. Er ist Verfasser des Teubner-Lehrbuchs „Das Objekt-Paradigma in der Informatik".
E-mail: quibeldey@sieis.informatik.uni-siegen.de

Neues Denken: Informatik ohne Algorithmen?

Komplexe Systeme entwerfen heißt Kompromisse suchen. Es gibt kein globales Optimum, dem man sich algorithmisch nähern könnte. Heuristiken, exploratorisches Prototyping oder der geführte Entwurf per Entwurfsmuster (pattern languages, vgl. [Cop95]) charakterisieren die Suche nach einer *zufriedenstellenden* Lösung [Sim82]. Interessant an den neuen didaktischen Ansätzen ist die Tatsache, daß sie weitestgehend ohne den Algorithmus-Begriff auskommen. Man suche ihn einmal in den modernen Lehrbüchern der Softwaretechnik oder des Datenbankentwurfs. Er scheint entbehrlich zu sein.

Revolutionäres Denken läuft in der Wissenschaft auf einen „Paradigmenwechsel" hinaus. Paradigmenwechsel sind selten; man sollte mit dem Begriff daher umsichtig sein. In der Definition nach Thomas Kuhn ist ein Paradigmenwechsel durch drei Phasen gekennzeichnet [Kuh89]:

1. „Normale Wissenschaft": Rätsellösen im Theoriengebäude des alten Paradigmas
2. Ad-hoc-Maßnahmen: Auftreten einer Krise und Schadensbegrenzung durch Vertreter des alten Paradigmas
3. „Außerordentliche Wissenschaft": Krisenbewältigung durch Vertreter des neuen Paradigmas

Das Objekt-Paradigma — verstanden als Gegenströmung zum algorithmischen Denken — befindet sich derzeit im Übergang zur dritten Phase. Die typischen Reaktionen sind allerorts noch spürbar:

- „Kognitive Dissonanz": Die Vertreter des alten Paradigmas fürchten um ihren wissen-

schaftlichen Ruf, um die Bedeutung ihres Wissens und Methodenbestands. Sie fühlen zwar, daß etwas Wichtiges im Gange ist, unterdrükken es aber und vermeiden die bewußte Auseinandersetzung mit den Widersprüchen in ihren alten Überzeugungen.

- „Prinzip Hoffnung“: Frustriert durch die methodischen Defizite des alten Paradigmas, werden innovative Ideen euphorisch aufgenommen, ohne deren Gültigkeit zu hinterfragen.
- „Déjà-vu-Erlebnis“: Die dritte Reaktion ist die häufigste: die Fakten des neuen Paradigmas hätten auch zur Zeit des alten erkannt werden können, sie waren damals nur „verborgen“. Das neue Paradigma schafft Déjà-vu-Erlebnisse: Der Neuigkeitswert wird angezweifelt.

Für die akademische Lehre bedeutend ist die weniger bekannte Definition eines Paradigmas nach Kuhn: die *disziplinäre Matrix*. Übertragen auf das Objekt-Paradigma ist hiermit die wissenschaftliche Gemeinschaft der Entwerfer gemeint — die „scientific community“. Sie findet ihren Ausdruck auf den Fachkonferenzen wie OOPSLA oder OOP '95. Wer schon mal daran teilgenommen hat, wird dieses „Wir-Gefühl“ kennen. Will man nun die disziplinäre Matrix als das Verbindende aller objektorientierten Themen in die Lehre einbringen, bedarf es eines *didaktischen Leitmotivs*: Was ist das Gemeinsame aller objektorientierten Entwürfe — seien es Analyse- oder Designmodelle, Programme, Datenbanken oder Wissensrepräsentationen?

Leitmotiv: „Tripel des Objekts“

Die Didaktik komplexer Entwürfe fordert eine begriffliche und zugleich technische *Grundeinheit des Entwerfens*. Ich nenne sie das „Tripel des Objekts“. Gefordert ist die Kapselung von *Struktur*, *Verhalten* und *Beschränkung*. Bild 1 macht den Grundgedanken anschaulich. Bisher reduziert man das Objekt-Paradigma auf den *Dualismus* von Struktur & Verhalten. Wohlgemerkt: Natürlich gehört es zum Wesen der Objektorientierung, Datenstrukturen und Datenzugriffe zu kapseln. Was konzeptionell aber fehlt, sind die Mittel, um Objektbeschränkungen *phasendurchgängig* zu spezifizieren und werkzeugtechnisch zu prüfen. Die Prüfung anhand von Vor- und Nachbedingungen, Regeln und Constraints wird nur rudimentär und auch nur für einzelne Entwicklungsphasen unterstützt. Bertrand Meyers Sprachschöpfung Eiffel ist die rühmliche Ausnahme für die Programmierung. Die meisten Methodenschöpfer indes begnügen sich mit der „Kästchenmalerei“ ohne semantische Prüfung (Constraints als grafisches Anhängsel).

Bild 1: Das Tripel des Objekts

Das „Tripel des Objekts“ als Leitmotiv einer objektorientierten Entwurfslehre liegt nahe: Denn, wenn eine Klasse den Abstrakten Datentyp (ADT) implementiert, wo bleiben dann die *beschränkenden* „Axiome“? Die „Signatur“ eines ADT kommt zweifellos rüber: „Sorten“ werden zu Klassenvariablen und „Operationen“ halt zu Klassenoperationen — aber die Axiome? Sie sichern erst die Korrektheit und Vollständigkeit, dienen also der *semantischen* Spezifikation. Obwohl die dreiteilige ADT-Charakteristik in jedem Lehrbuch zu finden ist, geht sie beim Übergang zur Klasse verloren!

Softwaretechnische Parallelen zum didaktischen Leitmotiv liegen zum einen im Client-Server-*Vertrag*. Er findet seinen Ausdruck sowohl in der objektorientierten Programmierung („Zusicherungen“ in Eiffel) als auch in der Analyse und im Design. Hier besonders durch *CRC-Karten* (Class-Responsibilities-Collaborators), um die *Pflichten* einer Klasse und ihre *Zusammenarbeit* mit anderen Klassen zu dokumentieren. Zum andern spielt der Aspekt der Beschränkung die zentrale Rolle in der objektorientierten Datenbanktechnik. Dort erreicht man Datenintegrität durch objektbezogene *Integritätsbedingungen*.

OOx: intuitiv & durchgängig

Das Leitmotiv der Objektorientierung ist phasenübergreifend: Der objektorientierte Weg im Produktlebenszyklus läßt sich durchgängig beschreiten. Analyse und Design führen zum *Weltmodell* der Anwendung. Programmierung und Datenhaltung, wobei Entwurfswerkzeuge als Objekte verwaltet werden, setzen das Weltmodell und die aus ihnen gewonnenen Objekte in ausführbare Rechnermodelle um. Die Schnittmengen in Bild 2 sollen die gemeinsamen Konzepte und Sprachmittel verdeutlichen, die im objektorientierten Entwurf zur begrifflichen und methodischen Durchgängigkeit beitragen.

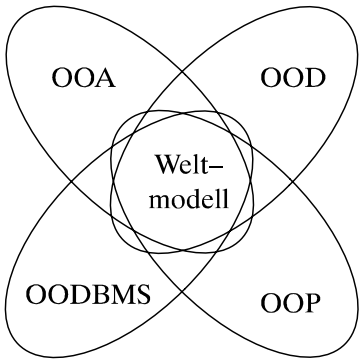

Bild 2: OOx-Welt des Software-Entwurfs

Den Konsens in der Modellierung eines Weltausschnitts — das Begriffssystem aus Klassen, Objekten, Vererbung und Polymorphie — hat das internationale Industriekonsortium OMG (Object Management Group) formuliert. Hier kommt man mit sehr einfachen Analogien aus dem Dienstleistungssektor aus: Die Begriffe *Auftraggeber*, *Auftragnehmer* und das *Dienstleistungsverhältnis* zwischen beiden genügen, um das Wesen der Objektorientierung intuitiv zu erfassen [Sny93].

Ein weiteres zum Leitmotiv-Charakter der Objektorientierung tragen die zahlreichen Metaphern bei: Vererbung, Nachrichtenaustausch, Klassenvertrag oder Desktop-Publishing begründen die Intuitivität objektorientierter Modelle und Methoden. Metaphern sind Brücken im Lernprozeß: Wir lernen, indem wir altes Wissen um neues bereichern; wir entwickeln neue Begriffe über Assoziationen mit bereits bekannten Begriffen. Metaphern machen also komplexe Begriffe über die Assoziation mit einfachen Begriffen intuitiv verständlich.

Entwurfslehre im großen: „Informatik-Systemtechnik“

Primär zielt die Objektorientierung auf den Entwurf *datenintensiver, komplex und heterogen strukturierter Anwendungssysteme*. Das gilt auch für die im Entstehen begriffene Ingenieurdisziplin „Informatik-Systemtechnik“. Im folgenden werde ich den Zusammenhang zeigen und dabei den Blick auf die curricularen Konsequenzen lenken, das heißt auf neue Lehrpläne für die Informatik.

Moderne rechnergestützte Anlagen und Geräte sind *komplexe* und *heterogene* Gebilde: Sie sind komplex, da sie sich aus verteilten Systemen zusammensetzen, die wiederum verteilte Systeme sein können. Sie sind heterogen, da sie soft- und hardwaretechnische Komponenten umfassen, wie Rechner und Peripherie, Datenbanken, Sensoren und Aktoren. Die Systemteile können geographisch verteilt oder lokal eng gekoppelt sein. Beispiele sind Leitsysteme für Gebäudekomplexe oder Systeme der Telekommunikation.

Für diese Klasse *rechnergestützter* Systeme wird der Begriff „Informatiksysteme“ vorgeschlagen, im Englischen CBS: Computer-Based Systems [Law90]. Die Ingenieurdisziplin, die sich mit dem Entwurf von Informatiksystemen befaßt, heißt „Informatik-Systemtechnik“ [ST93], englisch ECBS: Engineering of Computer-Based Systems. Bild 3 skizziert ihre Einordnung. Demnach ist die Informatik-Systemtechnik eine Teildisziplin der allgemeinen Systemtechnik. Sie integriert die Methoden, Werkzeuge und Produkte etablierter Ingenieurtechniken, um rechnergestützte Multisysteme zu entwerfen. Diese steuern Anlagen und Geräte oder sind in Geräten und Anlagen eingebettet.

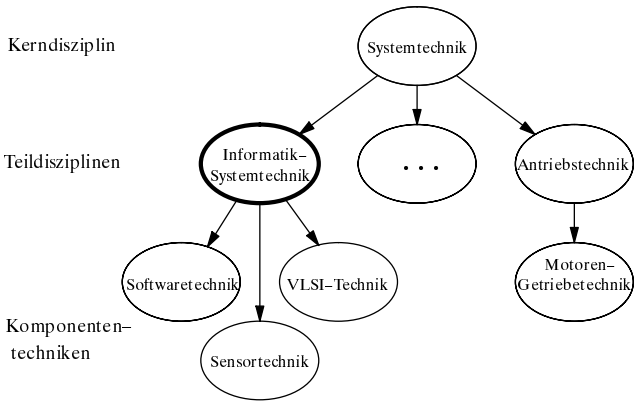

Bild 3: Entwurfslehre im großen

Die Komplexität und Heterogenität der Informatiksysteme erfordert übergeordnete und ausdrucksmächtige Modelle, effiziente Methoden und Werkzeuge. Hier sehe ich die zukünftige Hauptanwendung der *objektorientierten* Systemmodellierung, und zwar aus zwei Gründen:

Erstens, die Struktur- und Verhaltensmerkmale der Informatiksysteme sind begrifflich und methodisch durch das Objekt-Paradigma erfaßbar und in Modelle für den Rechner umsetzbar. Auf dem ersten ECBS-Workshop 1991 wurden die Struktur- und Verhaltensmerkmale von Informatiksystemen diskutiert und ein Konsens gefunden. Die Merkmale decken sich im wesentlichen mit denen der objektorientierten Client-Server-Architektur, wie sie von der OMG vorgeschlagen wird. Zweitens, das Problem der heterogenen Fachsprachen — das *Turmbau-zu-Babel*-Syndrom — ist durch die objektorientierte Terminologie überwindbar. Ich werde den terminologischen Aspekt an zwei Faktoren verdeutlichen:

Faktor „Team-Philosophie“

Der Pragmatiker Harold Lawson hat den Anspruch der Softwaretechnik der späten 80er Jahre ad absurdum geführt. Seine Kritik belegt er anhand der Ursachenanalyse gescheiterter Großprojekte:

„In den letzten Jahren zog die Softwaretechnik die Aufmerksamkeit auf sich — besonders ihre rechnergestützten Aspekte. Sie gelten als Heilmittel gegen die Probleme bei der Entwicklung komplexer rechnerbasierter Systeme. Leider bedeutet dies das Pferd beim Schwanz aufzäumen. Methoden und Werkzeuge der Softwaretechnik sind wichtig, aber sie sollten sich zwanglos aus einer fundierten Philosophie ergeben, wie man ein Applikationsproblem löst.“ [Law90]

Unter einer „Philosophie“ versteht Lawson die *gemeinschaftliche* Sicht, wie ein Problem prinzipiell angegangen werden sollte. Eine problemnahe Team-Philosophie basiert auf den *Begriffen* und *Strategien* aller Beteiligten: Auftragnehmer wie Auftraggeber. Die meisten gescheiterten Projekte begannen ohne Philosophie. Das Musterbeispiel einer philosophisch motivierten Entwicklung ist Simula, die Ursprache der Objektorientierung: Dahl, Myhrhaug und Nygaard, die Simula-Entwerfer, halfen den norwegischen Gewerkschaften bei der Analyse von Produktionsmodellen. Sie entwickelten Programmierkonzepte, wie Klassen von Objekten mit vererbbaren Eigenschaften, um industrielle Arbeitsumgebungen zu simulieren!

Lawson hat drei Grundeinstellungen für den erfolgreichen Entwurf von Informatiksystemen identifiziert, die seinem Philosophie-Begriff genügen:

- Management der Komplexität: Das Entwurfsproblem und die zu entwerfenden Artefakte (Kunstprodukte) werden als kompliziert eingestuft. Ein Projektmanagement wird eingerichtet, um die Komplexität zu bewältigen. Neue Artefakte sind erforderlich, um die zu entwerfenden Artefakte zu beherrschen (zum Beispiel CASE-Werkzeuge).
- Management der Formalismen: Linguistische Notationen und ihre Semantik werden eingesetzt, um das zu entwerfende System zu spezifizieren und das entworfene System zu verifizieren.
- Management der Kreativität: Der Projekterfolg hängt von kreativen und visionären Entwerfern ab. Nur sie sind fähig, die Philosophie im Projekt zu etablieren. Das Projektmanagement sollte diesen Schlüsselpersonen vertrauen und ihnen Verantwortung übertragen.

Es ist die Objektorientierung, die Lawson für eine geeignete Philosophie hält. Sie wird den drei Grundeinstellungen gerecht und kann die Entwicklung von Informatiksystemen leiten. Fördert man die Kommunikation und das Verständnis aller Projektbeteiligten, werden auch alle Phasen im Lebenszyklus eines Informatiksystems unterstützt. Die *gemeinsame* Philosophie ist der wichtigste Aspekt im Entwurf. Entwurfswerkzeuge und Programmiersprachen sind wichtig, aber der Team-Philosophie nachgeordnet.

Faktor „Team-Kommunikation“

Die kommunikative Bedeutung der Terminologie will ich nochmals unterstreichen: Die Vermengung der Begriffe ist umgangssprachlich und auch in der Informatik weitverbreitet. So spricht man fälschlicherweise von „Informationsverarbeitung“, was aus der unzulässigen Gleichsetzung resultiert: „Daten = Information“. Daten *codieren* aber nur eine verbale Nachricht, und erst diese ist kommunikativer Ausdruck der Information. Die Information konstituiert sich auf der Wissensebene, Nachrichten konstituieren sich auf der Kommunikationsebene und Daten auf der Maschinenebene. Den Zusammenhang zwischen Information (I), Nachricht (N)

und Daten (D) zeigt Bild 4, das ich etwas modifiziert [Hes94] entnommen habe.

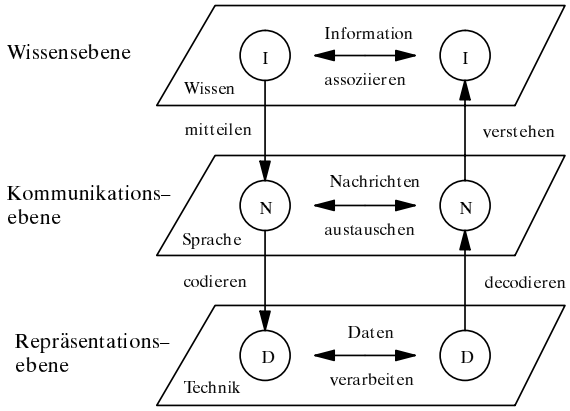

Bild 4: Daten $\neq$ Information

Die Abbildungen zwischen den Ebenen wirken im allgemeinen verkürzend, sind redundant und mehrdeutig. Dies um so mehr, wenn Personen verschiedener Fachgruppen Wissensaspekte auf verschiedenenen Sprachebenen diskutieren und dabei verschiedene technische Repräsentationen verwenden. Will man also komplexe und heterogene Anwendungssysteme gemeinsam analysieren und modellieren, bedarf es eines einheitlichen Begriffssystems (in meinem Sprachgebrauch eines „Weltmodells“, siehe Bild 2). Es sind gerade die objektorientierten Begriffe, wie Klassen, Objekte, Nachrichtenaustausch, Generalisierung und Komposition, die die gewünschte *Kongruenz* zwischen den Ebenen sichern [Qui94].

Analog zur konventionellen Terminologie der Softwaretechnik [Hes94] wäre eine *objektzentrierte* Terminologie für Informatiksysteme und deren Entwurfslehre geboten. Die systematische Begriffsbildung ist der Anspruch einer jeden Didaktik. Im Fall der Informatik-Systemtechnik kommt die *Konsolidierung* hinzu. Hier stoßen die Begriffswelten verschiedener Fachgruppen aufeinander: zum einen bei der *perspektivischen* Analyse und Modellierung der Anwendung und zum andern bei der Übernahme der Fachkonzepte in die Fachsprachen der Entwicklungsabteilungen. Eine objektzentrierte Terminologie könnte die Abteilungsgrenzen durchgängig halten: ohne Sprach-, Methoden- und Strukturbrüche zwischen Fach- und DV-Konzept. Eine konsolidierte Begriffssystematik des Objekt-Paradigmas steht aber noch aus.

Entwurfslehre im kleinen: curricularer Alltag

Wir wollen Studienanfänger nicht auf ein „Leben als Programmierer“ vorbereiten, die im Syntax-Dschungel ihrer Programmiersprache nächtelang knifflige Detailprobleme codieren. In der Tat hießen Programmierer früher einmal „Codierer“, und das beschreibt treffend ihre begrenzte Sicht auf den Systementwurf. Man sollte auch stets an die Sprachthesen von Wittgenstein und Whorf denken: Erkenntnisgewinn ist sprachbedingt! Unser Ziel ist eher der Wald als die Bäume, eher der Weg als das Ziel (Methode heißt „Weg zu etwas“). Das Berufsbild des Informatikers läßt sich mit „Systementwerfer“ realistischer und wohl auch „standesgemäßer“ umreißen, denn bloßes Programmieren lernt man besser in Kursen oder zuhause am PC.

Der Entwurf komplexer Systeme ist ein evolutionärer Anpassungsprozeß, um einen Kompromiß zu finden unter allen technischen, wirtschaftlichen und rationalen Beschränkungen. Was den Fortschritt im rechnergestützten Entwerfen betrifft — ausgeklügelte CAx-Systeme für Analyse, Simulation und Synthese —, so ist dieser *werkzeugzentrierte* Fortschritt Ausdruck eines Mythos. Denn Werkzeuge allein machen aus einem Laienentwerfer noch keinen Experten — „a fool with a tool is still a fool“. Es gibt die *Kreativitätslücke*: Bild 5 zeigt sie für den Hardware-Entwurf.

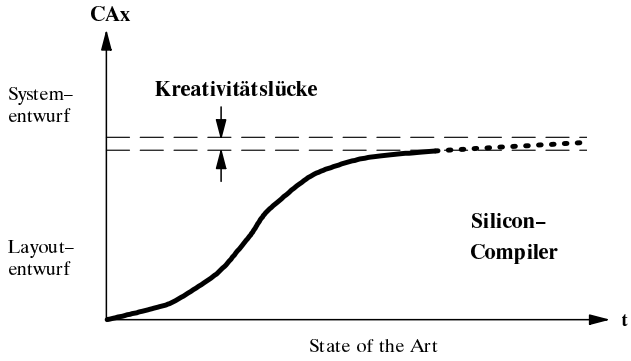

Bild 5: Entwurfsautomatisierung

Der „Stand der Technik“ verschiebt zwar stetig die Kreativitätslücke einerseits, die andere Seite aber bleibt per Werkzeug unerreichbar. Das Erfassen und Formulieren des Entwurfsproblems und der Lösungsansatz sind dort angesiedelt. Die Trendkurve im Bild 5 nähert sich nur asymptotisch dem Ideal des Silicon-Compilers: Es gibt ihn nur für *wohlstrukturierte* Anwendungen, deren Be-

schränkungen *vollständig im voraus* spezifizierbar sind. In diese Kreativitätslücke fällt meine curriculare Betrachtung: Wie kann das allseits geforderte „Systemdenken“ [Wei94] gelehrt oder zumindest nachhaltig angeregt werden? Die Objektorientierung in den frühen Phasen einer Informatik-Systemtechnik ist ein geeigneter Ansatz. Wie sieht die curriculare Realität aus?

In der Technischen Informatik der Universität Siegen integrieren wir die objektorientierten Themen derzeit wie folgt (SWS = Semester-Wochenstunden):

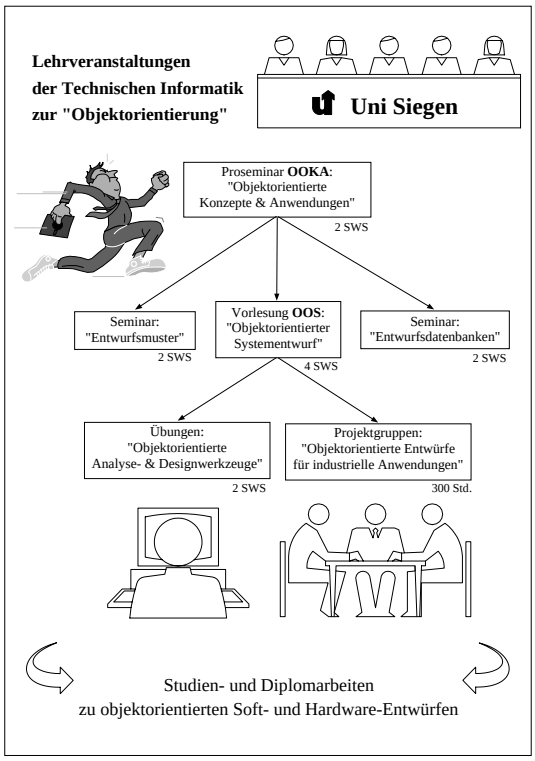

Bild 6: Entwurfslehre im kleinen

- Proseminar OOKA: „Objektorientierte Konzepte und Anwendungen“ (2 SWS)

  Wir fragen: „Ist die Objektorientierung nur ein populär-wissenschaftliches Schlagwort, ein verkaufsförderndes Marketing-Etikett für Software-Produkte, eine Elfenbeinturm-Technik oder aber ein Grundprinzip der Informatik?“ Die Teilnehmer diskutieren die Antworten anhand eines „Readers“ aus aktuellen Fachaufsätzen zum techno-philosophischen Spektrum der Objektorientierung.

- Vorlesung OOS: „Objektorientierter Systementwurf“ mit Übungen (6 SWS)

  Objektorientierte Themen durchdringen alle Entwurfsphasen — von der Spezifikation bis zur Programmierung komplexer Systeme. Mehr noch: selbst die kognitive Phase der Ideenfindung orientiert sich am Objekt. Die Vorlesung analysiert, erläutert und bewertet das Spektrum der objektorientierten Modelle und Methoden. Technische, wirtschaftliche und menschliche Gründe motivieren den objektorientierten Systementwurf [Qui94].

- Seminar „Entwurfsmuster“ (2 SWS):

  Es gibt nur wenige Universal-Strategien für den technischen Entwurf — Entwurfsmuster zählen zu den bedeutendsten. Wie kann der unerfahrene Entwerfer die Erfahrung des Experten nutzen? Wie läßt sich Erfahrung für die Wiederverwendung dokumentieren? Durch Muster! Die Teilnehmer diskutieren Vorschläge aus der Gebäudearchitektur, der Softwaretechnik [Cop95] und dem Schaltungsentwurf.

- Seminar „Entwurfsdatenbanken (2 SWS):

  Der Chip-Entwurf ist äußerst datenintensiv. Komplexe CAx-Objekte sind auf verschiedenen Abstraktionsebenen und unter verschiedenen Sichten im Team zu entwerfen und einheitlich zu verwalten. Datenintegrität ist das zentrale Problem. Ähnliches gilt für Software-Entwicklungsumgebungen. Der Ansatz über relationale Datenhaltungen erwies sich als ineffizient. Objektorientierte Datenbanken bieten sich an. Die Teilnehmer diskutieren aktuelle Forschungsergebnisse zum datenbankbasierten Soft- und Hardware-Entwurf.

- Projektgruppen (300 Arbeitsstunden)

  Die Aufgaben der Projektgruppen stammen aus der regionalen Industrie: von Anlagenbauern und Ingenieurbüros, die ihr Geld mit dem Entwurf von Informatiksystemen verdienen. Die Projektziele liegen in den frühen Phasen des objektorientierten Entwurfs: Analyse und Design (gegenwärtig im Bereich der Gebäude-Automation).

Unser Lehrplan zur Objektorientierung ist durchaus eigennützig angelegt: Wir „rekrutieren“ derzeit noch die Diplomanden aus unseren eigenen Veranstaltungen. Wir nähern uns dabei rasch der „kritischen Masse“, wie die Lehr-Evaluation zeigt: Die Teilnehmer von OOKA und OOS fühlen

sich mehrheitlich motiviert, ihr Studium auf OOx-Schwerpunkte auszurichten und ein entsprechendes Diplomthema zu bearbeiten.

Andere Fachgruppen bieten Einzelveranstaltungen zu weiteren Themen an: „Objektorientiertes paralleles Programmieren“, „Objektorientierter Entwurf grafischer Benutzungsoberflächen“ oder „Semantik objektorientierter Programmiersprachen“. Lehrsprache der Programmiermethodik ist im allgemeinen Eiffel oder Smalltalk. Eiffel hat bereits eine didaktische Bedeutung erlangt wie Pascal für die strukturierte Programmierung. Implementierungssprache ist in der Regel C++.

Fazit

Die universitäre Informatik-Ausbildung greift bereits die objektorientierten Themen auf. Es gibt zwar noch keinen ausgewiesenen Lehrstuhl für „objektorientierte Techniken“; die Vorlesungsverzeichnisse zeugen aber vom Einzug des Objekt-Paradigmas in die akademische Lehre. Die Objektorientierung beansprucht keine Alleinvertretung. Sie ist vielmehr Ausdruck der Umorientierung in der Informatik — von einer mathematiknahen Strukturwissenschaft, geprägt durch algorithmisches Denken, hin zu einer eigenständigen Ingenieurdisziplin, die übergreifendes *Systemdenken* fordert.

Literatur

[Bau94] R. Baumann, Algorithmen — wozu?, LOG IN, Oktober 1994

[Cop95] J. O. Coplien, Software-Entwurfsmuster, OBJEKTspektrum, 6/1995

[Hes94] W. Hesse et al., Terminologie der Softwaretechnik, Informatik-Spektrum, 1994

[Kuh89] T. S. Kuhn, Die Struktur wissenschaftlicher Revolutionen, Suhrkamp, 1989

[Law90] H. W. Lawson, Philosophies for Engineering Computer-Based Systems, Computer, Dezember 1990

[Qui94] K. Quibeldey-Cirkel, Das Objekt-Paradigma in der Informatik, Teubner, 1994

[ST93] G. Schweizer, B. Thomé, Informatik-Systemtechnik (ECBS): Gedanken zu einer Disziplin, Informatik-Spektrum, 1993

[Sim82] H. A. Simon, The Sciences of the Artificial, MIT Press, 1982

[Sny93] A. Snyder, The Essence of Objects: Concepts and Terms, IEEE Software, Januar 1993

[Wei94] G. Weinberg, Systemdenken und Softwarequalität, Hanser, 1994

[Who74] B. L. Whorf, Language, Thought and Reality, M.I.T. Press, 1974

[Wit73] L. Wittgenstein, Tractatus logico-philosophicus: Logisch-philosophische Abhandlungen, Suhrkamp, 1973