(Vortragskurzfassung)

# Das Objekt-Paradigma und seine Bedeutung für eine Informatik-Systemtechnik

Klaus Quibeldey-Cirkel*
Universität Siegen, Technische Informatik

Um die „Defizite im Software-Engineering“ zu überwinden, bedarf es der Einsicht, daß „die Probleme des Software-Engineering nicht gelöst werden können, solange sie nicht als Probleme des Systems-Engineering erkannt sind“ [1]. In den letzten Jahren wurden international Stimmen laut, die Kenntnis- und Ausbildungsdefizite in der Modellierung komplexer Systeme zu kompensieren: Eine neue Ingenieurdisziplin wird diskutiert, die sich der Komplexitätsbewältigung diesseits und jenseits des gegenwärtigen Soft- und Hardware-Entwurfs annimmt [2–5]. Die Förderung des „Systemdenkens“ unter Informatikern und Ingenieuren steht im Mittelpunkt [6]. Für die wissenschaftliche Fundierung der geforderten „Informatik-Systemtechnik“ wird dem Objekt-Paradigma eine wichtige Rolle zugeschrieben [7]: Es ist eine problemadäquate *Philosophie* und zugleich das *Instrumentarium*, um „Informatiksysteme“ kooperativ und konsistent zu entwickeln.

**Inhalt des Vortrags:** Zunächst charakterisiere ich das Nutzenpotential der Objektorientierung für die „frühen Phasen“ der Systementwicklung, das heißt für die Analyse und Modellierung komplexer Anwendungen. Als adäquate Problemklasse führe ich die Informatik-Systemtechnik ein, wie sie derzeit als fachübergreifende neue Ingenieurdisziplin diskutiert wird. Ich eruiere die Bedeutung der Objektorientierung für diese neue Disziplin und skizziere schließlich Realität und Ideal einer objektorientierten Entwurfslehre.

## 1 Das Manko der Objektorientierung

Die methodische und pragmatische Bedeutung des Objekt-Paradigmas ist unbestritten; hier bedarf es keiner weiteren Übersichtsbeiträge [8–13]. Dem Methodenboom der 90er Jahre [14–20] folgt aber nur zögernd die erforderliche Werkzeugunterstützung. Wo beides — Methode und Werkzeug — gegeben ist, irritiert ein Manko: die Beschränkung auf den *Dualismus* von Struktur & Verhalten. Wohlgemerkt: Es ist gerade das Wesen der Objektorientierung, Datenstrukturen und Zugriffsoperationen zu kapseln. Was aber bislang fehlt, sind die Mittel, um *invariante* Objekteigenschaften *phasendurchgängig* zu spezifizieren und diese Invarianz werkzeugtechnisch zu prüfen. Die Konsistenzprüfung anhand von Vor- und Nachbedingungen, Regeln und Constraints wird nur rudimentär und dann auch nur für einzelne Entwicklungsphasen unterstützt. Bertrand Meyers Sprachschöpfung Eiffel [21] ist die rühmliche Ausnahme für die Phase der Programmierung. Die meisten Methodenschöpfer indes beschränken sich auf die „Kästchenmalerei“ ohne semantische Prüfung.

Die Praxis komplexer Entwürfe fordert eine konzeptionelle und zugleich werkzeugtechnische *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:

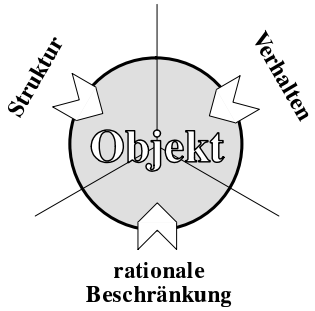


Bild 1: *Mind-map* zum Tripel des Objekts

Das „Tripel des Objekts“ liegt nahe: Denn, wenn eine Klasse einen Abstrakten Datentyp (ADT) implementiert, wo bleiben dann die beschränkenden *Axiome*? Die *Signatur* eines ADT kommt zweifellos rüber: *Sorten* werden zu Klassenvariablen und *Operationen* zu (Smalltalk-)Methoden — aber die Axiome? Sie sichern erst die Korrektheit und Vollständigkeit, dienen also der *semantischen* Spe-

*E-mail: quibeldey@sieis.informatik.uni-siegen.de

zifikation. Obwohl diese ADT-Charakteristik in jeder Definition zu finden ist [22], geht sie beim Übergang zur Klasse verloren!

## 2 Das Plus der Objektorientierung

Mit „Entwurf“ verknüpfe ich die „frühen Phasen“ der *deskriptiven* Systementwicklung, deren Ergebnisse die Grundlage bilden für die technische Konstruktion. „Entwerfen“ ist der kreative *und* ingenieurmäßige Vorgang, um aus einer informalen Beschreibung des gewünschten Systemverhaltens ein formales Anwendungsmodell zu erstellen: das *Was*-Modell des Kundenwunsches wird entworfen. Dieses gilt es, *restriktiv* auf die Struktur und das Verhalten wohlverstandener Systemkomponenten zu transformieren: das *Wie*-Modell der Ingenieurarbeit wird entworfen. Bildlich gesprochen, bedeutet Entwerfen die Transformation einer *Black-box*- in eine *White-box*-Systembeschreibung, die Transformation des *Was*-Modells der Analyse in das *Wie*-Modell der Konstruktion.

Das Objekt-Paradigma unterstützt diese Modell-Transformation ohne Methodenbruch. Die methodische Durchgängigkeit im Systementwurf trägt entscheidend zur Komplexitätsbewältigung bei. Was ist nun das Spezifische der Entwurfskomplexität? Was macht den Entwurf von Soft- und Hardware so kompliziert? Das Spezifische ist keine *quantifizierbare* Größe, wie die Anzahl von Speicherbits oder Programmzeilen. Die Entwurfskomplexität ist eine Frage der *Betrachtung*: Der allgemeine Entwurf komplexer Systeme ist nicht so sehr ein Mengenproblem, vielmehr liegt die Crux in der Vernetzung und Sichtenvielfalt der Entwurfsdaten und Entwurfsoperationen. So sind charakteristisch für den VLSI-Entwurf die Vielfalt an Abstraktionsebenen und Sichten (Verhalten, Struktur und Physik) einerseits und die Vernetzung von ebenen- und sichtenübergreifenden Analyse- und Syntheseschritten andererseits.

Das Spezifische der Entwurfskomplexität ist also die Beschreibungsvielfalt — der deskriptive Pluralismus. Hier setzen die objektorientierten Begriffe und Methoden an. Der Ansatz ist im Sinne einer ökonomischen Komplexitätsbewältigung optimal, denn er beginnt in den frühen Phasen der Systementwicklung. Hier wirken sich die Entwurfsfehler und -mängel überproportional auf die Kosten der Fehlerbeseitigung und Nachbesserung in späteren Phasen aus.

Die Objektorientierung fördert die Komplexitätsbewältigung mehrfach:

- durch Datenabstraktion
  (Konzentration auf das Wesentliche)
- durch Vererbung
  (Ökonomie im Ausdruck, Wiederverwendbarkeit)
- durch Polymorphie
  (abstrakte Klassen, virtuelle Funktionen)
- durch objektorientierte Oberflächen
  (intuitiver Benutzerzugang zu reaktiven Systemen)

Das Leitmotiv der Objektorientierung geht auf deren methodisch-operativen Durchgängigkeit zurück. Der objektorientierte Weg im Produktlebenszyklus läßt sich durchgängig beschreiten: OO-Analyse und OO-Design führen zu den Weltmodellen der Anwendung. OO-Programmierung und OO-Datenbanken, wobei Entwurfswerkzeuge als Objekte verwaltet werden, setzen die Weltmodelle und die aus ihnen gewonnenen Objekte in ausführbare Rechnermodelle dauerhaft um. Das Venn-Diagramm in Bild 2 soll die Schnittmengen an gemeinsamen Konzepten und Sprachmitteln verdeutlichen, die im objektorientierten Entwurf auftreten und zur begrifflichen und methodischen Durchgängigkeit beitragen.

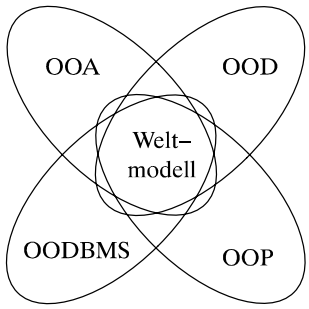


Bild 2: OO-Welt des Software-Entwurfs

Auf die Gemeinsamkeiten in der Modellierung eines Weltausschnitts, wie Klassen, Objekte, Vererbung und Polymorphie, wird beispielsweise in [23] näher eingegangen. Diese Quelle kommt mit einfachen Analogien aus dem Dienstleistungssektor aus: Die Begriffe *Auftraggeber*, *Auftragnehmer* und das *Dienstleistungsverhältnis* zwischen beiden genügen, um das Wesen der Objektorientierung zu erfassen.

Ein weiteres zum Leitmotiv-Charakter der Objektorientierung tragen die zahlreichen Metaphern bei: Vererbung, Nachrichtenaustausch, Client-Server-Vertrag, Round-Trip-Gestalt-Design [14], Desktop Publishing, Fenster und Menüs. Sie begründen die Intuitivität objektorientierter Methoden und Werkzeuge [13].

# 3 Informatik-Systemtechnik

Der primäre Gegenstand der Objektorientierung sind *datenintensive, komplex und heterogen strukturierte Anwendungssysteme*. Das gilt auch für den Gegenstand der im Entstehen begriffenen Ingenieurdisziplin: Informatik-Systemtechnik. Im folgenden werde ich den Zusammenhang zeigen und dabei den Blick auf die curricularen Konsequenzen lenken.

Moderne rechnergestützte Anlagen und Geräte sind *komplexe* und *heterogene* Gebilde: Sie sind komplex, da sie sich aus verteilten und parallelen Systemen zusammensetzen, die wiederum verteilte Systeme sein können. Sie sind heterogen, da sie soft- und hardwaretechnische Komponenten umfassen, wie Rechner und Peripherie, Datenbanken, Benutzungsschnittstellen, Sensoren und Aktoren. Die Systemteile können geographisch verteilt oder lokal eng gekoppelt sein. Einige Beispiele mögen diese Systemklasse verdeutlichen: integrierte Fertigungssteuerung, Leittechnik für Kraftwerke, Flugleit- und Telekommunikationssysteme.

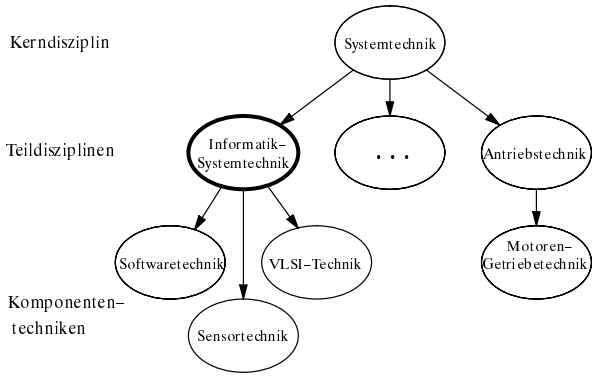


Bild 3: Einordnung der Informatik-Systemtechnik

Für diese Klasse rechnergestützter Systeme wird der Begriff „Informatiksysteme“ vorgeschlagen, im Englischen CBS: *Computer-Based Systems* [7]. Die Ingenieurdisziplin, die sich mit der Konzeption und Entwicklung von Informatiksystemen befaßt, heißt „Informatik-Systemtechnik“, englisch ECBS: *Engineering of Computer-Based Systems* [4, 5]. Bild 3 skizziert ihre Einordnung. Demnach ist die Informatik-Systemtechnik eine Teildisziplin der allgemeinen Systemtechnik (*Systems-Engineering*). 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.

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 konzeptionell 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 erster Konsens gefunden [2]. Die Merkmale decken sich im wesentlichen mit denen der objektorientierten Client-Server-Architektur, wie sie vom internationalen Industriekonsortium OMG (*Object Management Group*) vorgeschlagen wird [24].

Zweitens, das Problem der heterogenen Fachsprachen und Kommunikationsschnittstellen — das *Turmbau-zu-Babel*-Syndrom — ist durch die objektorientierte Terminologie überwindbar. Ich werde den Forschungsbedarf in diesen Aussagen motivieren.

# 4 Objektorientierung als Team-Philosophie

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

> „In recent years, attention has focused on software engineering — especially its computer-aided aspects — as a ‚cure‘ for the life-cycle problems of complex computer-based systems. Unfortunately, this emphasis is like placing the cart before the horse. Software engineering methods and tools are important, but they should be the natu-

> ral result of a well-developed philosophy for solving the application domain." [7]

Unter einer „Philosophie" versteht Lawson die gemeinschaftliche Sicht aller Mitglieder eines Projekts, wie ein Problem oder eine Klasse von Problemen *prinzipiell* angegangen werden sollte. Die Gemeinsamkeit der Sicht basiert auf den gemeinsamen *Begriffen* und *Strategien* aller Beteiligten: Auftragnehmer wie Auftraggeber und „all other parties with vested interests". Die Methoden und Werkzeuge einer ingenieurmäßigen Systementwicklung sollten einer *problemrelevanten* Philosophie entspringen; die meisten gescheiterten Projekte begannen ohne Philosophie — mit einer rühmlichen Ausnahme, die für den Objektansatz Symbolcharakter hat: die Entwicklung von Simula. Lawson, ein Skandinavier, erhellt die Philosophie dieser skandinavischen Entwicklung:

> „This programming language was designed in the early 1960s as a result of a philosophy for simulating industrial work environments and social systems. Dahl, Myhrhaug, and Nygaard, the Simula founders, later assisted Norwegian labor unions in gaining insight into various production models. They evolved concepts they felt were important for programming industrial work environment simulators, including the notions of classes, subclasses, objects with attributes, and inheritance. These concepts became the heart of the Simula philosophy, which lives on among avid Simula users and is perpetuated by the Simula User's Group." [7]

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

- *Management of Complexity*

  Das Entwurfsproblem und die zu entwerfenden Artefakte werden als kompliziert eingestuft. Ein Projektmanagement wird eingerichtet, um die Komplexität zu bewältigen. Neue Artefakte sind erforderlich, um die zu entwerfenden Artefakte zu beherrschen (CASE-Werkzeuge zum Beispiel).

- *Management of Formalism*

  Linguistische Notationen und ihre Semantik werden eingesetzt, um das zu entwerfende System zu spezifizieren und das entworfene System zu verifizieren.

- *Management of Creativity*

  Der Projekterfolg hängt von kreativen und visionären Entwerfern ab. Nur sie sind fähig, die Philosophie im Projekt zu etablieren und verständlich zu machen. 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 durchgängig unterstützt. Die *gemeinsame* Philosophie ist der wichtigste Aspekt im Entwurf. Artefakte, wie Entwurfswerkzeuge und Programmiersprachen sind wichtig, aber sekundär. Auch die IEEE-Task-Force zur ECBS-Initiative kommt zu dieser Auffassung: Eine ECBS-Philosophie muß primär die Kommunikation und das Verständnis im Projektteam fördern; hierzu bedarf es einer gemeinsamen *intuitiven* Terminologie, um die *holistic properties of CBS* zu erfassen. Eine derartige Philosophie ist auch für die Ingenieurausbildung bedeutsam:

> „The basic scientific knowledge required covers many scientific domains, which a single individual cannot grasp in depth. Hence, a proper balance has to be found in curricula to provide adequate knowledge in each. In some cases, finding unifying concepts could possibly reduce the learning load, and this is certainly an object for ECBS research." [2]

Wie die einschlägige Literatur zeigt, verfügt das Objekt-Paradigma über die *unifying concepts*, um Informatiksysteme ganzheitlich zu verstehen, zu spezifizieren und zu implementieren [8–13].

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 oder nonverbale Nachricht, und erst diese ist kommunikativer Ausdruck einer Information. Eine 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 [25] entnommen habe.

Die Abbildungen zwischen den Ebenen wirken im allgemeinen verkürzend, sind redundant und mehr-

deutig. Dies um so mehr, wenn Personen verschiedener Fachgruppen Wissensaspekte eines Diskursbereichs austauschen und dabei auf verschiedenenen Sprachebenen kommunizieren und 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"). Es sind gerade die objektorientierten Begriffe, wie Klassen, Objekte, Nachrichtenaustausch, Generalisierung und Komposition, die die gewünschte Kongruenz zwischen den Ebenen sichern.

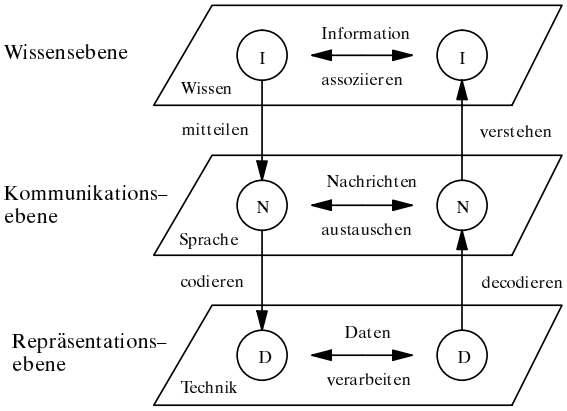


Bild 4: Terminologisches Umfeld

Analog zur konventionellen Terminologie der Softwaretechnik [25] wäre eine *objektzentrierte* Terminologie für Informatiksysteme und deren Technik geboten. Die konsistente und systematische Begriffsbildung ist ein wissenschaftlicher Anspruch einer jeden Disziplin. 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 anderen 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.

# 5 Curriculare Betrachtung

Entwerfen ist ein offener, exploratorischer Prozeß mit dem Ziel, einen Kompromiß zu finden unter allen technischen, wirtschaftlichen und rationalen Beschränkungen. Was nun den Fortschritt im rechnergestützten Entwerfen betrifft — ausgeklügelte CAx-Systeme für Analyse und Synthese —, so ist dieser *werkzeugzentrierte* Fortschritt Ausdruck eines Mythos: Denn Werkzeuge allein machen aus einem Laienentwerfer noch keinen Experten.[1] Es gibt eine *Kreativitätslücke*: Bild 5 zeigt sie für den Hardware-Entwurf [27].

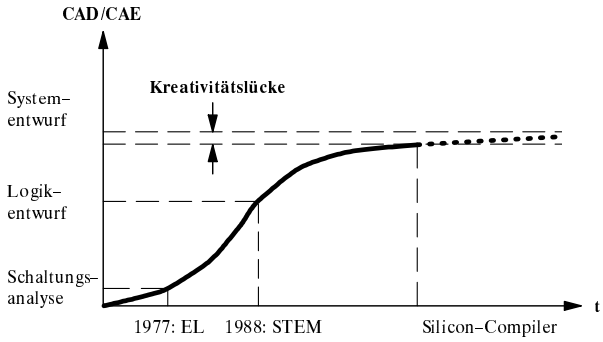


Bild 5: Trend der Entwurfsautomatisierung

Ihr eines Ende, zugewandt der Entwurfsautomatisierung, ist zwar bestimmt durch den „Stand der Technik", ihr anderes Ende aber bleibt operativ unerreichbar. Das Erkennen und Formulieren eines Problems und der Ansatz für seine Lösung 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 Beschränkungen *vollständig im voraus* spezifizierbar sind. In diese Kreativitätslücke fällt meine curriculare Betrachtung: Wie kann das allseits geforderte „Systemdenken" 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 Fachgruppe Technische Informatik der Universität Siegen integrieren wir objektorientierte Themen und den Entwurf komplexer Systeme derzeit so:

[1] Ein englisches Sprichwort sagt es treffend: „A fool with a tool is still a fool." Ähnlich prägnante Formulierungen findet man in [26].

- Proseminar OOKA: „Objektorientierte Konzepte und Anwendungen“
- Vorlesung OOS: „Objektorientierter Systementwurf“ mit Übungen und Projektgruppen
- VLSI-Informationsmodellierung in der objektorientierten Sprache Express [28]
- Programmiermethodik: Eiffel und C++

Diese Veranstaltungen haben primär die Analyse und Modellierung in den frühen Phasen des Systementwurfs zum Inhalt. Die Aufgabenstellungen der Projektgruppen (Diplom- und Studienarbeiter) stammen aus der regionalen Industrie: Anlagenbauer und Ingenieurbüros, die sich mit dem Entwurf von *Informatiksystemen* beschäftigen.

Ideale haben die Eigenschaft, nicht erreichbar zu sein, sonst wären sie keine. Das Ideal einer Entwurfslehre liegt meines Erachtens im Umfeld

- einer „Science of Design“ im Sinne von Herbert Simon [29],
- einer „Architektur des Entwerfens“ im Sinne von Heinz Zemanek [30] und
- des „prozeßzentrierten“ Ansatzes im Sinne von Christiane Floyd [31].

Der jeweilige Beitrag der Objektorientierung zu diesen interdisziplinären Vorschlägen einer Entwurfslehre würde den Vortragsrahmen sprengen; hier sei auf [13] verwiesen.

## Literatur


[1] Wendt, S.: *Defizite im Software-Engineering.* In: Informatik-Spektrum, Jg. 16, 1993, S. 34–38

[2] Agrawala, A.; Buhr, R.; Jackson, K.; Jackson, M.; Lang, B.; Lavi, J. Z.: *Computer Based Systems Engineering Workshop.* In: Proc. of the SEI 1991 Conf. on Software Engineering Education, Lecture Notes on Computer Science, Nr. 536, Springer-Verlag, 1991

[3] Schweizer, G.; Thomé, B.: *Informatiksystemtechnik (CBSE): Gedanken zu einer Disziplin.* In: Informatik-Spektrum, Jg. 16, 1993, S. 215–221

[4] Thomé, B. (Hrsg.): *Systems Engineering: Principles and Practice of Computer-Based Systems Engineering.* Wiley, New York, 1993

[5] White, S.: *Systems Engineering of Computer-Based Systems.* In: Computer, Jg. 26, H. 11, 1993, S. 54–65

[6] Müller, J.-A.: *Kommt die Entwicklung betrieblicher DV-Anwendungssysteme ohne Systems-Engineering aus?.* In: Informatik-Spektrum, Jg. 16, 1993, S. 167–169

[7] Lawson, H. W.: *Philosophies for Engineering Computer-Based Systems.* In: Computer, Jg. 23, H. 12, 1990, S. 52–63

[8] Themenheft: *Object-Oriented Design.* In: CACM, Jg. 33, H. 9, 1990

[9] Themenheft: *Inheritance and Classification in Object-Oriented Computing.* In: Computer, Jg. 25, H. 10, 1992

[10] Themenheft: *Making O-O Work.* In: Software, Jg. 10, H. 1, 1993

[11] Themenheft: *Projektmanagement für objektorientierte Software-Entwicklung.* In: Informatik-Spektrum, Jg. 15, S. 253–292, 1992

[12] Themenheft: *Objektorientierte Datenbanksysteme.* In: Informatik-Spektrum, Jg. 16, S. 67–97, 1993

[13] Quibeldey-Cirkel, K.: *Das Objekt-Paradigma in der Informatik.* Teubner-Verlag, Stuttgart, 1994

[14] Booch, G.: *Object-Oriented Analysis and Design with Applications.* The Benjamin/Cummings Publishing Company, Redwood City, CA, 2. Aufl., 1994

[15] Coad, P.; Yourdon, E.: *Object-Oriented Analysis.* Prentice Hall, New Jersey, 1991

[16] Coad, P.; Yourdon, E.: *Object-Oriented Design.* Prentice Hall, New Jersey, 1991

[17] Dillon, T.; Tan, P. L.: *Object-Oriented Conceptual Modeling.* Prentice Hall, Sidney, 1993

[18] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.: *Object-Oriented Modeling and Design.* Prentice Hall, New Jersey, 1991

[19] Shlaer, S.; Mellor, S. J.: *Object Lifecycles: Modeling the World in States.* Prentice Hall, New Jersey, 1992

[20] Wirfs-Brock, R. J.; Wilkerson, B.; Wiener, L.: *Designing Object-Oriented Software.* Prentice Hall, New Jersey, 1990

[21] Meyer, B.: *Object-Oriented Software Construction.* Prentice Hall, 1988

[22] *Duden Informatik.* Hrsg.: Lektorat des B.I.-Wissenschaftverlags unter Leitung von H. Engesser, bearb. von V. Claus und A. Schwill, 2. Aufl., Mannheim, 1993

[23] Snyder, A.: *The Essence of Objects: Concepts and Terms.* In: IEEE Software, Jg. 10, H. 1, 1993, S. 31–42

[24] Object Management Group: *Common Object Request Broker Architecture and Specification.* Hrsg.: Soley, R., Framingham, Mass., 1992

[25] Barkow, G.; von Braun, H.; Hesse, W.; Kittlaus, H.-B.; Scheschonk, G.: *Terminologie der Softwaretechnik: Ein Begriffssystem für die Analyse und Modellierung von Anwendungssystemen, Teil 1: Begriffssystematik und Grundbegriffe und Teil 2: Tätigkeits- und ergebnisbezogene Elemente.* In: Informatik-Spektrum, Jg. 17, 1994, S. 39–47, 96–105

[26] Brooks, F. P.: *The Mythical Man-Month.* Addison-Wesley Publishing Company, Reading, Mass., 1975

[27] Quibeldey-Cirkel, K.: *Objektorientierung als methodisch-operativer Ansatz zur Erstellung und Integritätssicherung von VLSI-Weltmodellen.* Dissertation, Universität Siegen, 1994

[28] Schenk, D.: *EXPRESS Language Reference Manual.* McDonnell Aircraft Company, St. Louis, MO., 1990

[29] Simon, H. A.: *The Sciences of the Artificial.* MIT Press, Cambridge, Mass., 2. Aufl., 1982

[30] Zemanek, H.: *Das geistige Umfeld der Informationstechnik.* Springer-Verlag, Berlin Heidelberg, 1992

[31] Floyd, C.: *Software-Engineering — und dann?.* In: Informatik-Spektrum, Jg. 17, 1994, S. 29–37