Paradigmenwechsel im Software-Engineering: Auf dem Weg zu objektorientierten Weltmodellen

Klaus Quibeldey-Cirkel*
Universität Siegen

Der Paradigma-Begriff, von Thomas Kuhn in die Wissenschaftstheorie eingebracht, findet vielerorts seinen Gemeinplatz — so auch in der Informatik. Neue Strategien in der Software-Konstruktion, wie „Concurrent Engineering“ oder „CAD-Framework“, tragen das marketingwirksame Etikett. Zuweilen und zur Rechtfertigung weiterer Programmiersprachen wird der Begriff zum bloßen Sprach-„stil“ mißbraucht: „multi-paradigm languages“ [32]. Der folgende Beitrag klärt die wissenschaftliche Bedeutung und Verwendung des Paradigma-Begriffs. Wir nennen die typischen Reaktionsmuster auf neue Paradigmen und beantworten die Frage: Wie lange dauert es, bis sich ein „neues Denken“ in Theorie und Praxis durchsetzt? Wir skizzieren die Methodenwechsel der Vergangenheit und kennzeichnen die gegenwärtige Hauptströmung: das Objekt-Paradigma. Den Hintergrund unserer Diskussion bildet die fachübergreifende Methodik des *konzeptionellen Modellierens*.

1 Paradigmenwechsel im großen

Das Selbstverständnis der Informatik ist im Umbruch. Bisweilen bestimmten es zwei Elemente: (a) die Ausrichtung an formalen Strukturen nach dem Vorbild der Mathematik und Logik (Datentypen und Algorithmen) und (b) die Implementierung dieser Strukturen nach einer ingenieurmäßigen Pragmatik, die sich im Beiwort *Engineering* ausdrückt, wie Requirements- und Software-Engineering. Der duale Anspruch an Theorie und Praxis spiegelt sich in den heutigen Schuldefinitionen wider. Ein Beispiel aus dem Informatik-Duden:

> „Informatik (*computer science*): Wissenschaft von der systematischen Verarbeitung von Informationen, besonders der automatischen Verarbeitung mit Hilfe von ↑ Digitalrechnern (↑ Computer).“ [21]

Auf den Menschen als Gestalter und Betroffenen informationstechnischer Prozesse wird, wie bereits Rafael Capurro bemerkt, nicht verwiesen [6]. In letzter Zeit aber hat sich das techno-zentrische Weltbild zu einem anthropozentrischen gewandelt: Der Mensch rückt in den Mittelpunkt der systematischen Informationsverarbeitung. Gegenüber der herkömmlichen Auffassung als Struktur- und Ingenieurwissenschaft wird die Informatik als eine *interdisziplinäre* Fachrichtung herausgestellt, „mit der Aufgabe der technischen Gestaltung menschlicher Interaktionen mit der Welt“ [6, S. 311]. Hierzu ein Definitionsvorschlag von Kristen Nygaard, Mitschöpfer der Sprache Simula:

> „Informatics is the science that has as its domain information processes and related phenomena in artifacts, society and nature.“ (zitiert in [6, S. 314])

Die Neuorientierung im Selbstverständnis der Informatik — vom „Rohstoff“ Information zum Thema „Mensch“ — wird in den Schlagworten deutlich, die derzeit in der Diskussion um die sozialen Aspekte hochgehalten werden: Ethik und Informatik, Informatik und Gesellschaft, verantwortliche Technikgestaltung und so fort. Im Kontext dieser philosophischen Neubesinnung soll im folgenden auf paradigmatische Wechsel im Kernbereich der Informatik, dem Software-Entwurf, näher eingegangen werden. Es zeigt sich, daß gegenwärtig das Objekt-Paradigma die Methodendiskussion in der Software-Entwicklung bestimmt [34, 48, 66, 69].

Die techno-philosophische Breite des Objektansatzes unterstützt die These, daß die Suche nach einer interdisziplinären „Wissenschaft des Entwerfens“, wie sie zuerst Herbert Simon und Heinz Zemanek und derzeit Christiane Floyd fordern [29, 62, 74], ihren Ausgangspunkt im Objekt-Paradigma nimmt. Objektorientiertes *konzeptionelles Modellieren* ist interdisziplinär [18, 26]. Gerade die jungen Informatikzweige weisen synergetische Ziele auf: Wissensrepräsentation in der Künstlichen Intelligenz (KI), semantische Datenmodelle in der

*E-mail: quibeldey@ti.e-technik.uni-siegen.d400.de

Datenbanktechnik und objektorientierte Analyse- und Entwurfsmethoden im Software-Engineering haben ihre Gemeinsamkeit in der Modellbildung auf konzeptionellen Ebenen. Sie teilen sich problemnahe Konzepte und Mechanismen, wie hierarchische Kategorien und Vererbung, um das Wissen über einen Weltausschnitt formal zu erfassen und für die informationstechnische Verarbeitung aufzubereiten.

2 Kuhns These

Die bahnbrechende Arbeit von Thomas Kuhn, *Die Struktur wissenschaftlicher Revolutionen* [41], führte zu der Erkenntnis, daß die Wissenschaftsentwicklung nicht geradlinig verläuft, es also keinen kontinuierlich kumulativen Fortschritt gibt. Am Beispiel der Naturwissenschaften verdeutlicht Kuhn, wie die stationären Phasen *normaler* Wissenschaft von Perioden *revolutionärer* Wissenschaft abgelöst werden. Zwei Beispiele: die Welt als flache Scheibe versus die Kugel als Weltbild, newtonsche Mechanik versus relativistische nach Einstein. Das gültige *Paradigma*[1] — die Konstellation von Meinungen, Werten und Methoden, die von den Mitgliedern einer wissenschaftlichen Gemeinschaft (*scientific community*) geteilt werden — wird in der revolutionären Phase durch ein innovatives Paradigma in Frage gestellt und schließlich überwunden.

Wissenschaft schreitet somit in zwei Modi voran, die sich zyklisch wiederholen. Erstens: Normale Wissenschaft verläuft empirisch und inkrementell. Beobachtungen und Fakten werden in das von der wissenschaftlichen Gemeinschaft aufgestellte Theoriengebäude eingefügt (Kuhn nennt die Haupttätigkeit in dieser Phase *Rätsellösen* und *Aufräumen*). Zweitens: Das gültige Paradigma gerät in eine Krise. Wichtige Rätsel können nicht zufriedenstellend gelöst werden. Es werden Ad-hoc-Modifikationen ersonnen, um die „Anomalien" zu lösen. Einzelne Wissenschaftler finden sich mit der Situation nicht ab und stellen neue Theorien auf. Die Krise des alten und der Drang des neuen Paradigmas führen schließlich zur wissenschaftlichen Revolution. Dem erfolgreichen Paradigmenwechsel schließt sich alsdann wieder eine Phase normaler Wissenschaft an.

[1] Definition nach Kuhn [41, Postskriptum]: (1) Paradigma als *Musterbeispiel*: die einheitliche Überzeugung über den Forschungsgegenstand und über die Methodik seiner Erforschung und (2) Paradigma als *disziplinäre Matrix*: die Gesamtheit dessen, was eine Gemeinschaft von Wissenschaftlern an kognitiven und sozialen Elementen verbindet.

Die Reaktionsmuster auf einen Theorie-Anwärter sind deutlich sozio-psychologischer Art. Peter Molzberger nennt drei typische Muster, die auf Konferenzen, an Universitäten und in der Industrie als Antwort auf innovative Software-Methoden überwiegen [47]:

1. Emotionale Ablehnung

   Die Mitglieder der wissenschaftlichen Gemeinschaft, die das alte Paradigma vertritt, fürchten um ihre wissenschaftliche Reputation, um die Bedeutung ihres Wissens und Methodenbestands. Sie fühlen zwar, daß etwas Wichtiges im Gange ist, unterdrücken es aber und vermeiden jede bewußte Auseinandersetzung mit den Widersprüchen in ihren alten Überzeugungen. Psychologen wie Leon Festinger kennen ein solches Reaktionsmuster als „kognitive Dissonanz" [25].

2. Unreflektierte Euphorie

   Geboren aus der Frustration durch die methodischen Unzulänglichkeiten des vorherrschenden Paradigmas, werden innovative Ideen euphorisch aufgenommen, ohne deren Gültigkeit zu hinterfragen („Prinzip Hoffnung").

3. *Déjà-vu*-Erlebnis

   Die dritte Reaktion auf einen Paradigmenkonflikt ist die häufigste. Nach Kuhn hätten die Beobachtungen, die das neue Paradigma unterstützen, auch zu Zeiten des alten gemacht werden können, aber diese waren damals nicht „sichtbar". Mit dem Aufzeigen des neuen Paradigmas kommt es bei vielen zum *Déjà-vu*-Erlebnis: Gegenwärtiges glaubt man, schon einmal in der Vergangenheit — meist unbewußt — erlebt zu haben. Der Neuigkeitswert wird angezweifelt.

Für die Informatik hat Volker Claus in empirischen Studien herausgefunden, wie lange es im allgemeinen dauert, bis neue Paradigmen in die Praxis umgesetzt sind [8]. Zwischen dem ersten Auftauchen in der Wissenschaft und der Einführung in die Praxis vergehen in der Regel:

- sieben bis zehn Jahre, wenn dadurch ein Engpaß in der wirtschaftlichen Machbarkeit eines Forschungsvorhabens beseitigt werden kann

  (Beispiel: Edsger Dijkstra argumentierte erstmals 1968 in seinem Brief an die ACM öffentlich gegen den Gebrauch von GOTO-Anweisungen: *Go To Statement Considered Harmful* [16]; die GOTO-Kontroverse erreichte ihren

Höhepunkt 1972 und wurde 1974 von Donald Knuth endgültig beigelegt [40]);

- 15 bis 20 Jahre bei breitbandiger Bedeutung für die Software-Industrie

  (Beispiel: Bereits 1965 wurden die wichtigsten Ideen und Konzepte der „strukturierten Programmierung“ in Dijkstras Aufsatz *Programming Considered as a Human Activity* veröffentlicht [17], zum Stand der Technik wurden sie erst in den 80er Jahren).

Die Praxisnähe der Methoden und die Verfügbarkeit effizienter Werkzeuge sind untrennbare Kriterien für einen erfolgreichen Paradigmenwechsel in der Industrie. Der Akzeptanzverzug in der industriellen Praxis ist im allgemeinen um den Faktor zwei größer als in der universitären Forschung. Kuhn selbst geht von einem langwierigen Wandel in den Denkmustern einer wissenschaftlichen Gemeinschaft aus: mindestens 25 Jahre vergehen, bis ein Paradigmenwechsel vollzogen ist. Kraß formuliert es Max Planck, rückblickend auf seine wissenschaftliche Laufbahn:

> „Eine neue wissenschaftliche Wahrheit pflegt sich nicht in der Weise durchzusetzen, daß ihre Gegner überzeugt werden und sich als belehrt erklären, sondern vielmehr dadurch, daß die Gegner allmählich aussterben und daß die heranwachsende Generation von vornherein mit der Wahrheit vertraut gemacht ist.“ (zitiert in [41, S. 162])

3 Paradigmenwechsel im kleinen

Jede Disziplin hat ihre Lehrmeinungen, Methoden und Werte und durchläuft mehr oder weniger abrupte Phasenwechsel im Erkenntnisprozeß. In der Informatik beschleunigt vor allem eine Triebfeder die Wissenschaftsentwicklung: die Suche nach übergeordneten Prinzipien und Methoden für Problemanalyse, Spezifikation, Entwurf und Implementierung. Ziel ist es, fehlerarme, wartbare, erweiterbare und wiederverwendbare Software mit wirtschaftlichem Aufwand herstellen zu können. Die wissenschaftlichen Anstrengungen, um dieses Ziel zu erreichen, streben eine Zunahme des „semantischen Ausdrucks“ an. Vier zum Teil parallel verlaufende Leitlinien durchziehen das Streben nach höheren Software-Abstraktionen:

1. prozedurale Abstraktion: Funktionen, Prozeduren, Programmodule
2. syntaktische Abstraktion: Bitmuster, Mnemonik, Sprachkonstrukte, Metaphern
3. Datenabstraktion: Datenstrukturen, abstrakte Datentypen, Klassen
4. Wissensabstraktion: Kategorien, Vererbung, Wissensrahmen (*Frames*)

Entlang den chronologischen Abstraktionslinien, den Spuren eines 40-jährigen Lernprozesses, lassen sich die Paradigmenwechsel in der Software-Konstruktion aufzeigen.[2] Sie spiegeln das unruhige Wachstum der jungen Disziplin wider, ursprünglich bedingt durch den Zulauf von Quereinsteigern: Mathematikern und Physikern. Als Sammelbekken für euphorische Technikerwartungen (Stichwort: CASE) kann das Software-Engineering nur langsam seine Eigenständigkeit entwickeln.

3.1 Die methodenlose Zeit

Im Sinne von Kuhn handelt es sich in der Zeit von 1940 bis etwa 1960 um eine *vor-paradigmatische Phase*, in der noch nach einem ersten Paradigma gesucht wird und noch nicht von einer Gemeinschaft von Wissenschaftlern gesprochen werden kann. Das Wort „Informatik“ oder „Computer Science“ wird erst Ende der 60er Jahre zum Begriff [6, S. 313]. Forschung ist hier eher eine Einzelanstrengung. Diese später von Kuhn als *Proto-Wissenschaft* bezeichnete frühe Phase der Wissenschaftsentwicklung ist geprägt durch das Bedürfnis und die Suche nach brauchbaren Methoden.

In den Anfängen der Rechnertechnik dominierten die geringen Hardware-Ressourcen an Rechenzeit und Speicherplatz die Forschung. Das Programmieren wurde nicht als eigenständige Aufgabe, sondern als pragmatische Nebentätigkeit der Hardware-Entwicklung gesehen. *One-man algorithmic programs* im Umfang von 100 bis 200 Programmzeilen in einer heutigen Hochsprache waren gerade noch machbar — mit verborgenen Tricks und Kniffen im Maschinencode (Programmierer hießen damals „Codierer“). So wie in der Automatentheorie nicht zwischen einer Maschine und ihrem Programm unterschieden wird, so bilden in der frühen Geschichte der Rechnerentwicklung Soft-

[2] Die geschichtlichen Daten wurden größtenteils [40, 68] entnommen.

und Hardware einen integrierten Forschungsgegenstand.

Die erste prozedurale Abstraktion, die *closed subroutine* (aufrufbares Code-Segment im Gegensatz zum Quelltext-Segment einer *open subroutine*), geht auf Konrad Zuse zurück Er führte sie 1945 in seinem *Plankalkül* ein, der ersten algorithmischen Sprache [60]. Unabhängig compilierbare Unterprogramme wurden zuerst in Fortran-II, 1958, und geschachtelte in Algol-60 unterstützt. 1947 wurde erstmals die Verwendung von Flußdiagrammen, um den Kontrollfluß zu visualisieren, durch Herman Goldstine und John van Neumann vorgeschlagen. Mitte der 50er Jahre setzte die Entwicklung des Makro-Assemblers ein. Der breite Einsatz der prozeduralen Abstraktion in der Programmierung begann also erst mit den 60er Jahren.

3.2 Die Kunst des Programmierens versus Software-Engineering

Der exponentiell wachsende Fortschritt in der Hardware-Technologie, der Ende der 60er Jahre einsetzte und bis heute andauert,[3] gab der Software-Entwicklung eine neue Gewichtung. Waren es zunächst kleine Entwürfe, die es unter größter Hardware-Ausnutzung zu erstellen galt, so sind nun „gute" Entwürfe gefragt, die komplexe Aufgaben angehen. Da systematische Entwurfsmethoden erst zögernd aufkommen, stilisiert sich das Programmieren ohne Hardware-Auflagen schnell zu einer Kunst.[4]

Das Wort von der „Softwarekrise" geht um: Die fehlende Lokalität und Robustheit der Entwurfsentscheidungen (Seiteneffekte sind nicht auszuschließen) und, damit eng verbunden, die aufwendige Wartbarkeit der sich aufblähenden *Spaghetti*-Programme (zwei Drittel der Softwarekosten fallen allein auf die Wartung), werden als ursächlich erkannt. Auf der Garmisch-Konferenz, im Oktober 1968, mit dem epochemachenden Titel *Software Engineering* wird erstmals eine „ingenieurmäßige" Pragmatik für die Software-Konstruktion gefordert [51]. Dollpunkte, die bis heute die Methodendiskussion bestimmen:

- Einführung eines Phasenmodells des Software-Lebenszyklus (Wasserfall-Modell)
- systematisches Analysieren, Entwerfen, Programmieren, Testen und Dokumentieren
- Definition und Standardisierung der Programmier-Schnittstellen
- Rechnerunterstützung mit dem langfristigen Ziel, den Entwurfsprozeß zu mechanisieren („ausführbare Spezifikation")

Als Theorie-Anwärterin für einen Paradigmenwechsel tritt die „strukturierte Programmierung" auf den Plan (erste Monographie von Dahl, Dijkstra und Hoare: *Structured Programming* [12]). Unter den Sammelbegriff fallen die schrittweise Verfeinerung der Entwurfsentscheidungen (*Top-down*-Entwurf), die Beschränkung auf drei Kontrollstrukturen (Sequenz, Auswahl und Schleife: *GOTO-less programming*), der modulare Entwurf aus einzeln prüfbaren *single-entry/single-exit*-Modulen und die Einführung von Programmier-Richtlinien (strukturierter Quellcode durch Einrückungen und *begin-end*-Blöcke, kleine Module).

Aus der Handwerkskunst des Programmierens wird langsam eine Wissenschaft. Die Etablierung der Fachdisziplin Informatik an deutschen Hochschulen datiert Alfred Luft auf die späten 60er Jahre [42].

3.3 Der Faktor „Mensch"

Mit der *Human-Factors*-Bewegung in der Informatik der 80er Jahre — die Probleme sind nicht mehr ausschließlich algorithmischer Art, sondern zunehmend schlecht definiert und erfordern die Mitwirkung des Benutzers in der Definitionsphase der Software-Entwicklung — kommt auch die Forderung nach einer Rückbesinnung auf das schöpferische Potential des Programmierers auf. Gerald Weinbergs legendäres Werk *The Psychology of Computer Programming* ist seiner Zeit weit voraus [67]: „People talked to me as if I was a crazy person", sagt er nach der ersten Konferenz über *Human Factors in Computer Systems* in Gaithersburg — elf Jahre später [47].

Christiane Floyd fordert die Wende von der produktorientierten (mechanistischen) Sicht hin zur prozeßorientierten, in der die Bedürfnisse und kreativen Fähigkeiten des Menschen als Entwerfer und Nutzer von Software im Mittelpunkt stehen [28].

[3] Gordon Moore, Mitbegründer von Intel, stellte in [49] die Regel auf, nach der sich die Integrationsdichte der Halbleiterschaltungen seit 1975 alle 18 bis 24 Monate verdoppelt.

[4] Über das Programmieren, verstanden als eine der „schönen Künste", siehe Donald Knuths Turing-Award-Lecture [39].

Die pragmatischen Aspekte in der Produktentwicklung, wie Qualitätssicherung, Methoden- und Dokumenteneinsatz, müssen den situativen Kontext und die kognitiven Beschränkungen des Menschen berücksichtigen. Das impliziert beiderseitige Lernprozesse durch Kommunikation und Kooperation zwischen den beteiligten Personenkreisen: *Reviewing* im Team als kritischer Diskurs über Entwurfsentscheidungen und *Prototyping* als exploratorischer *Trial-and-Error*-Prozeß zwischen Entwickler und Anwender.

Peter Molzbergers Forderung nach Suche und Förderung des kreativen und zugleich hocheffizienten *Superprogrammers* [47] und die von Frederick Brooks erhoffte *Silver Bullet* gegen monströse Software-Projekte, der *Great Designer* [4], stehen exemplarisch für die Diskussion um das kreative Potential des Programmierers. Beide wissen die Erkenntnisse der modernen Motivationsforschung auf ihrer Seite: siehe Abraham Maslows Bedürfnispyramide in Bild 1.

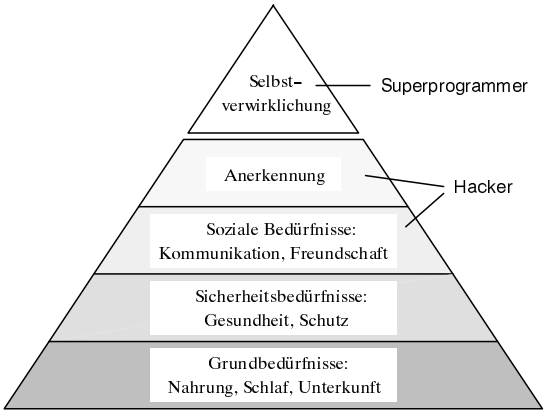

Bild 1: Superprogrammer versus Hacker

Es soll der *ganzheitliche* Mensch — seine analytischen-assoziativen Denkweisen und seine höheren Bedürfnisse nach Anerkennung (*ego needs*) und Selbstentfaltung — in den Entwurfsprozeß mit einbezogen werden.

„Ganzheitlichkeit" wird hier im Sinne der Neurophysiologie verstanden: In der linken Hemisphäre unseres Gehirns sind nach dem Nobelpreisträger Roger Sperry Sprache, Rationalität und Zielorientierung lokalisiert. Sie dominiert das Bildungswesen unseres Kulturkreises. Die rechte Hälfte beherbergt die assoziativen Erkenntnisprozesse, Kreativität und Intuition, und wird, zumindest was die Ausbildung der Programmierer betrifft, kaum gefördert. Das Phänomen des außergewöhnlichen Programmierers, wie Brooks und Molzberger es identifiziert haben, ist eindeutig dem rechten Potential unseres Gehirns zuzuordnen. Molzberger organisiert 1983 einen Workshop zum Thema: *Programming with the Right Hemisphere of the Brain.*

Auch in der Künstlichen Intelligenz wendet sich der Trend von der mechanistischen zur individuellmenschlichen Orientierung. Glaubte man vorher noch, Denkprozesse auf Symbolmanipulationen reduzieren zu können [62], so beschränkt man sich heute auf die formale Wissensrepräsentation und versucht, aus dem Wissen eines einzelnen, des Experten, weiteres logisch abzuleiten [2].[5]

3.4 SA/SD versus OOx

Wir kommen in unserer Chronik zum gegenwärtigen Paradigmenwechsel. Um zu zeigen, daß es sich hier in der Tat um einen Wechsel im Kuhnschen Sinne handelt, wollen wir streng im klassischen Dreisatz vorgehen: (1) Rätsellösen in der Phase normaler Wissenschaft, (2) Auftreten einer Krise und (3) Überwindung der Krise durch einen neuen Theorie-Anwärter.

1. Rätsellösen

   Seit etwa Mitte bis Ende der 70er Jahre werden zahlreiche funktionale Dekompositionsmethoden für das Struktur-Paradigma entwickelt und finden nach dem „Clausschen Akzeptanzverzug" zehn Jahre später Eingang in die industrielle Praxis. Die Analysemethode nach Tom DeMarco und die SA/SD-Methode (*structured analysis/structured design*) nach Edward Yourdon gelten als etabliert [14, 73].

2. Krise

   Der Freiraum des Software-Entwurfs wird nicht länger durch die Unzulänglichkeiten der Hardware beschnitten. Immer komplexere Probleme werden angegangen, die nur noch arbeitsteilig im Team zu lösen sind. Frank DeRemer und Hans Kron prägen hierfür den Begriff des *Programming-in-the-Large* [15]. Peter Schnupp macht einen „Methodenberg" des Struktur-Paradigmas aus [58], dessen Schwerpunkt über den technischen Phasen der

[5] Kritische KI-Forscher, wie Terry Winograd [27] oder die Gebrüder Hubert und Stuart Dreyfus [20], erfahren in ihrer *scientific community* das Reaktionsmuster erster Prägung, die „emotionale Ablehnung", und werden (vor)schnell als „Nestbeschmutzer" diffamiert [42, S. 71].

Software-Entwicklung liegt (Bild 2). Die methodische Unterstützung in den problemnahen konzeptionellen Entwurfsphasen wird als unzureichend empfunden.

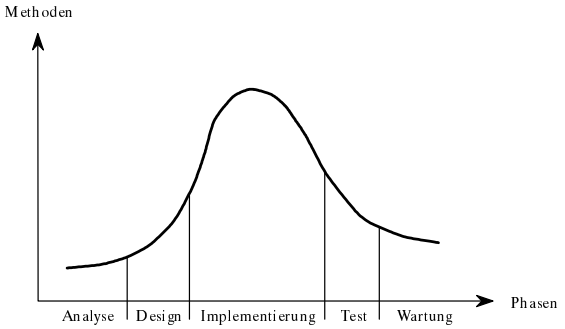

Bild 2: Methodenberg des Struktur-Paradigmas

Forderungen, die erkannten Defizite im Software-Lebenszyklus zu beheben, werden laut:

- Aufhebung der willkürlichen Trennung der Datenstrukturen von ihren Kontrollstrukturen oder mit anderen Worten: die Schaffung einer Einheit aus statischer Information und dynamischem Zugriff, der Arbeitsgegenstand wird ganzheitlich (Klassenkonzept in Simula, abstrakter Datentyp)
- Verlagern methodischer Ansätze vom Lösungs- in den Problembereich, somit eine größere Entfernung von den Vorgaben der Rechnerarchitektur eines John von Neumann
- durchgängige Konzepte und Konstrukte im Software-Lebenszyklus: die Ergebnisse der Phase i sollen sich ohne Konvertierungsaufwand als Grundlage für das weitere Vorgehen in der Phase $i+1$ eignen

3. Lösung

Die Indizien und Argumente für einen bevorstehenden Paradigmenwechsel häufen sich:

- Neue Forschungsanstrengungen

  Eine Literaturrecherche offenbart die mannigfaltigen Querbezüge der Objektorientierung in der Informatik: Nicht nur in den konventionellen Phasen der Systementwicklung, wie Analyse, Entwurf und Programmierung, wird ein objektorientierter Ansatz verfolgt, sondern auch in der Konzeption und Implementierung von Non-Standard-Datenmodellen, VLSI-Werkzeugen und CAD-Frameworks. Das Balkendiagramm in Bild 3 verdeutlicht die wissenschaftliche Breitenanwendung der Objektorientierung.

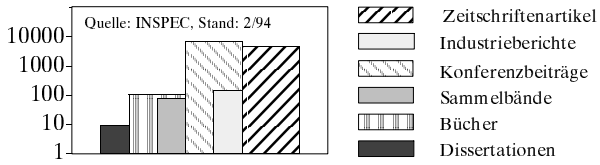

Bild 3: Spektrum und Zahl der OOx-Publikationen

Die Grafik in Bild 4 zeigt anschaulich, wie die objektorientierten Themen (OOx) den vor zwanzig Jahren aus der Softwarekrise geborenen strukturierten Ansätzen für Analyse, Design und Programmierung (SA/SD) den Rang ablaufen.[6]

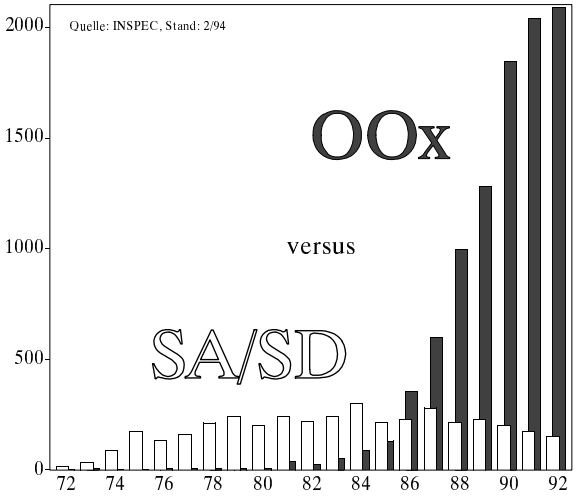

Bild 4: Statistisches Indiz für einen Trendwechsel

[6] OOx steht als Sammelkürzel für objektorientierte (OO) Methoden und Techniken in den Entwurfsphasen des Software-Lebenszyklus: Analyse (OOA), Design (OOD), Programmierung (OOP) und Datenbanksysteme (OODBMS). In der Statistik wurden ausschließlich disjunkte Einträge gezählt (nur englischsprachige), also entweder SA/SD oder OOx. Veröffentlichungen, die beide Ansätze verfolgen, fielen also heraus.

- Außergewöhnliche Forscher

  (a) Alan Kays Vision vom notizbuchgroßen *Personal Computer*, dem *Dynabook* [36], ist heute Realität und hat uns zahlreiche objektorientierte Errungenschaften beschert: neben der interaktiven Sprache Smalltalk vor allem hochflexible, an den kognitiven Fähigkeiten des Menschen orientierte Grafik-Rechner mit Fenstertechnik, Sinnbildern, Menüführung und der Maus als Eingabemedium [37]. Apples *Macintosh* mit all seinen ergonomischen Vorteilen ist das geistige Kind des Software-Visionärs Alan Kay.

  (b) Als ein weiteres Indiz für die Folgerichtigkeit der Objektorientierung in der Informatik mag Edward Yourdons Sinneswandel gelten: vom Wegbereiter der strukturierten Systemanalyse (SA) und des strukturierten Entwurfs (SD) zum Verfechter des Objekt-Paradigmas in diesen Disziplinen: OOA [9] und OOD [10].

- Bewahren erfolgreicher Lösungskonzepte und Versprechen neuer

  Das Objekt-Paradigma vereinigt drei Tendenzen in der Geschichte der Informatik: (a) die fortschreitende Entwicklung abstrakter Sprachkonstrukte, die dem Problem näher sind als der Maschine, (b) die zunehmende Entlastung des Systementwerfers von Entwurfsentscheidungen auf unteren Ebenen (*Information hiding* im Sinne von David Parnas [52]) und (c) die Suche nach interdisziplinären Prinzipien des Entwerfens.

Der Kürze und Prägnanz wegen skizzieren wir das objektorientierte Lösungspotential, die Konzepte, Methoden, Verfahren und Techniken, nach übergeordneten Gesichtspunkten: ETHOS [56]. Zusammen betonen sie den ganzheitlichen Anspruch des Objekt-Paradigmas:

E wie „economic“: Das Klassenkonzept aus Simula und das Kapselungsprinzip des *Information hiding* erfüllen wichtige wirtschaftliche Forderungen: Software-Wiederverwendung, lokale Wartbarkeit und Erweiterbarkeit. Klassenbibliotheken unterstützen den ingenieurmäßigen „Bausteinentwurf“, wie er im Maschinenbau, in der Bautechnik und Fließbandindustrie gang und gäbe ist. Dem *Time-to-Market*-Faktor wird Rechnung getragen: Die Entwicklungsproduktivität und die Produktqualität steigen [23, 46].

T wie „technical“: Die Objektorientierung führt zu Modellierungsmethoden und -techniken, die durchgängig in den klassischen Phasen des Entwurfs Verwendung finden: Analyse- und Designmethoden [1, 9, 57, 61, 71]; inkrementelle Programmierung in Smalltalk oder Eiffel; Datenarchivierung [19] und grafische Benutzungsoberflächen [22].

H wie „human“: Die wachsende Metaphorik in der Software-Konstruktion (Beispiele: *Vererbung*, *Austausch von Botschaften* und *Client-Server*) ruft Assoziationen und Analogien beim Benutzer hervor. Die rechte Hemisphäre des Gehirns wird angeregt und führt mit den algorithmischen Komponenten der linken zu einem *ganzheitlichen* Denken. Objektorientierung ist intuitiv. Sie unterstützt die Wahrnehmung und kognitive Verarbeitung eines Problems. Sie stellt zugleich ein ontologisches Grundprinzip[7] dar, das geeignet ist, auch technische Artefakte einfach und ganzheitlich zu modellieren [65]. Ontologisch fundierte Referenzmodelle erlauben die vorurteilslose Bewertung von Analyse- und Designmethoden [64].

O wie „organizational“: Die objektorientierten grafischen Oberflächen senken die Akzeptanzschwelle des Rechners in den betrieblichen Standardsituationen. Das konzeptionelle Potential des Objekt-Paradigmas wird zunehmend auch in der Unternehmensmodellierung und im Produktdaten-Management ausgeschöpft. Objektorientierte Erweiterungen des *Entity-Relationship*-Modells von Peter Chen setzen sich durch [7]. Produktdaten werden in der ISO-Norm STEP in Express beschrieben, einer objektorientierten Modellierungssprache [53]. Im Rahmen der *Programming-in-the-Large*-Projekte setzt die Projektführung verstärkt OO-Konzepte und -Methoden ein [24, 31].

S wie „social“: Die Interdisziplinarität des Objektansatzes fördert die Kommunikation und Kooperation sowohl innerhalb der Informatik (Software-Engineering, Datenbanken und KI) als auch zwischen den Vertragspartnern eines Software-Projekts, wo Auftraggeber und Auftragnehmer aus disjunkten Fachgebieten kommen. Die Schnittstellen-Probleme kommunikativer Art werden durch das *Requirements-Engineering* und *Rapid-Prototyping* als objektorientiertes dialogisches Bindeglied zwischen Benutzer und Systementwerfer entschärft [43]. Auf einer höheren Ebene fungiert das Objekt-Paradigma als *disziplinäre Matrix* einer „Wissenschaft des Entwerfens“.

[7] Ontologie ist die wissenschaftliche Philosophie von der Modellierung der Objekte in der realen Welt [5].

4 Ausblick: OO-Weltmodelle

Eingedenk des paradigmatischen Wechsels im philosophischen Selbstverständnis der Informatik und der aktuellen Methoden-Wende im Software-Engineering vom Struktur- zum Objekt-Paradigma läßt sich ein Bild des allgemeinen Entwurfsprozesses skizzieren, für Soft- und Hardware gleichermaßen. Als Beispiel diene der VLSI-Entwurf, verstanden als konzeptionelles Co-Design von Soft- und Hardware.

Die zahlreichen Abstraktionen und Sichten im VLSI-Entwurf reichern einen „Informationspool" an, der die Historie der Entwurfsobjekte fortschreibt. Die Generalanforderungen an die Modellierung des Informationspools nennen wir „Weltmodell", angelehnt an den Begriff des „Datenmodells" in der Datenbanktechnik. Wie Datenmodelle, zum Beispiel das relationale von Edgar Codd [11], umfassen Weltmodelle die konzeptionellen Mittel zur Modellierung eines Weltausschnitts, *Miniwelt* genannt.

In den Vordergrund rücken also zwei Facetten der Informationsverarbeitung:

1. Datenrepräsentation: Modellieren und Schematisieren der Entwurfsinformation
2. Datenverwaltung: Konsistentes Fortschreiben und Abfragen des Datenbestands

Um Systeme zu modellieren, setzen wir Mittel der Abstraktion ein, Konzepte und Formalismen, die die Entwurfskomplexität reduzieren und beherrschbar halten [55]. Der Mensch mit seinen kognitiven Grenzen tritt in den Vordergrund [50]. Die gemeinsamen Entwicklungslinien der Soft- und Hardware-Abstraktionen, wie sie beispielsweise von Carlo Séquin für die Hardware [59] und von Mary Shaw für die Software [60] aufgezeigt wurden, stehen für die Eckpfeiler einer fortschreitenden methodischen „Disziplinierung". Ob *Programming-in-the-Large* oder *System-on-Chip*, die Parallelen sind unverkennbar: Tabelle 1.

Wir entnehmen dieser Methodenentwicklung eine übergreifende Gesamtausrichtung, eine Grundströmung, die sich durch die Praktische und Technische Informatik zieht: Wissensrepräsentation in der KI [2], semantische Datenmodellierung in der Datenbankforschung [19], Objektmodelle in der Programmiermethodik [65] und Informationsmodellierung im Kontext von CAD-Frameworks [53]. Ohne es deutlich herauszustellen, betreiben die verschiedenen Informatikforschungen eine vergleichbare Methodik der Komplexitätsbewältigung: die Modellbildung auf konzeptioneller Ebene. Auch wenn es sicherlich Unterschiede in der Zielsetzung gibt, so sind doch die Methoden vergleichbar. Die fachübergreifenden Konferenzen und Veröffentlichungen in der Informatik zeugen davon: [3, 38, 44]. Es ist hier nicht der Raum, um die Parallelen im Detail aufzuzeigen, aber es genügt wohl ein allgemeines Verständnis der Methoden dieser Fachdisziplinen, um den Schluß nachzuvollziehen: Das augenfällig Gemeinsame ist die Suche nach einer genügend präzisen Beschreibung für das Inventar und die Dynamik eines Weltausschnitts.

Vor der Implementierung steht die Spezifikation und vor der Spezifikation das „Begreifen" eines Problems, das kognitive Erfassen der Miniwelt. Bislang weist die Entwurfsschiene von der Idee zum Produkt *konzeptionelle* Fugen auf: Die Beschreibungsmittel der Problemanalyse, der Spezifikation und der Implementierung sind in der Regel disjunkt, die „semantischen Lücken" sprichwörtlich. Objektorientiertes *konzeptionelles Modellieren* will alle Phasen durchgängig halten: methodisch und begrifflich. Brooks zählt sie zu den Hoffnungen auf eine *Silver Bullet* gegen monströse Software-Projekte:

Abstraktion	*Hardware*	*Software*
Modularisierung	Bausteinprinzip: Zell-Bibliotheken, Steuer- und Operationswerke, Speichermodule, Boards	Blockstrukturen: Funktionen, Prozeduren, abstrakte Datentypen, Klassen
Hierarchie	Abstraktionsebenen (*system, algorithmic, micro-architectural, logic, circuit*), Sichten (*behavioral, structural, physical*) [30]	*schrittweise Verfeinerung* [72], Haupt- und Unterprogramme, Klassenhierarchien und Vererbung [45]
Formalisierung	Datentransferformate, Hardware-Beschreibungssprachen, Silicon-Compiler	Algebraische Spezifikation [35], Programmgeneratoren

Tabelle 1: Entwicklung der Soft- und Hardware-Abstraktionen

„Fashioning complex conceptual constructs is the *essence*; *accidental* tasks arise in representing the constructs in language. Past progress has so reduced the accidental tasks that future progress now depends upon addressing the essence.“ [4]

Wie die *conceptual constructs* für den VLSI-Entwurf aussehen, wurde in [54] untersucht.

4.1 Ausführbare Weltmodelle

Wir stimmen überein mit David Harels Antwort auf Brooks' Pessimismus: *Biting the Silver Bullet: Toward a Brighter Future for System Development* [4, 33]. Die „Akzidenzien“ der Software-Entwicklung, das heißt die zufälligen Begleiterscheinungen und Schwierigkeiten, sind beherrschbar oder werden es in naher Zukunft sein. Grafik-Werkzeuge, wie Generatoren, Editoren und Browser, und Management-Werkzeuge für die Versionsverfolgung und Konfiguration von Objekten erleichtern dem Weltmodellierer die Arbeit. Seine Modelle konstruiert, analysiert und dokumentiert er bereits auf dem Rechner. Stets schwierig bleibt die „Essenz“, die in Struktur und Verhalten liegende Komplexität des Weltausschnitts.

Was wir brauchen, um die Essenz einer komplexen Entwurfsaufgabe in den Griff zu bekommen, sind Code erzeugende Werkzeuge, die das konzeptionelle Modell „zum Laufen bringen“, es in die Konstrukte einer Programmiersprache umformen oder für VLSI-Applikationen in eine Hardware-Beschreibungssprache wie VHDL. Das sind zum Beispiel Schema-Compiler, die ein Weltmodell in ein Datenbankschema überführen, oder Modell-Simulatoren, die eine Aussage über die Konsistenz der Objekte unter dynamischen Bedingungen erlauben. Das heißt also: Rechnerunterstützung nicht nur bei der textuellen und grafischen Darstellung von Weltmodellen, sondern auch bei der Inspektion, Integritätssicherung und Implementierung derselben.

4.2 OO-Szenario

Der objektorientierte Weg im Software-Lebenszyklus läßt sich durchgängig beschreiten: OO-Analyse und OO-Design führen zu den Weltmodellen der Applikation. OO-Programmierung und OO-Datenbanken, wobei Entwurfswerkzeuge als Objekte verwaltet werden [13], setzen die Weltmodelle und die aus ihnen gewonnenen Objekte in ausführbare Rechnermodelle dauerhaft um. Das Venn-Diagramm in Bild 5 soll die Schnittmengen an gemeinsamen Konzepten und Konstrukten verdeutlichen, die in einem voll objektorientierten Entwurfsprozeß auftreten und für die begriffliche und methodische Durchgängigkeit sorgen.

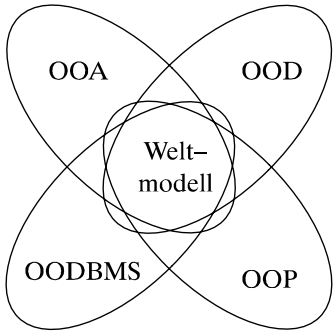

Bild 5: OO-Welt des Software-Entwurfs

Auf das Gemeinsame, wie Klassen, Objekte, und Vererbung, geht Alan Snyder in [63] ein.

CASE-Werkzeuge unterstützen bereits ausreichend die implementierungsnahen Entwurfsphasen: Objektorientierte Programmierumgebungen und Datenbanken sind Stand der Technik und werden erfolgreich eingesetzt [24]. Die mehr der Problemanalyse zugewandten Entwurfsabschnitte hingegen befinden sich werkzeugtechnisch *in statu nascendi* und schicken sich erst an, die etablierten strukturierten Analyse- und Entwurfsmethoden in Frage zu stellen. Oder, um das Bild des Methodenbergs ein zweites Mal zu bemühen: Der Methodenbestand strebt erst noch die Form eines Tafelbergs an, der alle Phasen der Produktentwicklung gleichermaßen überdeckt.

Fazit

Kuhns Begriff des Paradigmenwechsels wird auch in der Informatik strapaziert. Einschneidende Phasenwechsel in einer Wissenschaft bleiben rar, sind dann aber bedeutungsvoll für den Erkenntnisprozeß. Die Voraussetzungen für die Verwendung des Begriffs sind im allgemeinen nicht gegeben oder werden ignoriert: Auftreten einer Krise im Rätselraten mit Hilfe des gültigen Paradigmas, Gelehrtenstreit mit gruppenpsychologischen Reaktionsmustern und Überwindung der Krise durch einen neuen erfolgversprechenden Theorie-Anwärter.

Die Paradigmenwechsel in der Software-Konstruktion wurden skizziert. Die Methodenentwicklung folgt langfristig einer Pendelbewegung. Die Umkehrpunkte des Pendels sind abwechselnd überzogene Erwartungen an das Potential der aktuellen Methode oder allgemein empfundene Fortschrittskrisen K_i. Bild 6 zeigt den zyklischen Verlauf (auf steigendem Niveau!), den auch der Informatik-Duden konstatiert [21].

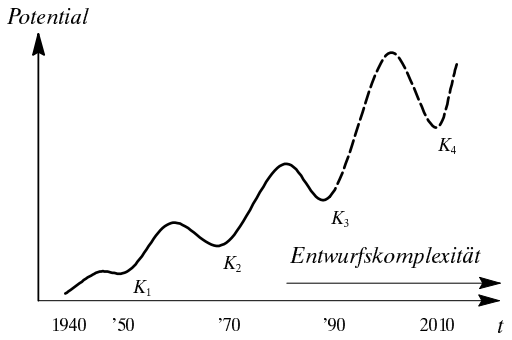

Bild 6: Methodenentwicklung

Demnach wandeln sich die Programmstrukturen alle zwanzig Jahre:

1. strukturlose Programmierung (methodenlose Zeit bis etwa 1950)

   Die *Methodenkrise* K_1 führt zur Entwicklung von Assemblern und Compilern für höhere Programmiersprachen.

2. künstlerische Strukturen (*Computer Programming as an Art* [39]: 1950–1970)

   Die *Softwarekrise* K_2 führt zur Entwicklung strukturierter Programmiersprachen.

3. prozedurale Strukturen (strukturierte Programmierung: 1970–1990)

   Die *Wiederverwendungskrise* K_3 führt zur Entwicklung objektorientierter Programmiersprachen.

4. objektorientierte Strukturen (1990 – ?)

   Die *Parallelitätskrise* K_4 führt ...

Der aktuelle Wechsel von SA/SD nach OOx wurde identifiziert. Der Handlungsbedarf zielt auf die Problembeschreibung durch den Endbenutzer und auf die Problemanalyse durch den Systementwickler. Das hier angesiedelte moderne Requirements-Engineering sucht den unproblematischen Anschluß an das traditionelle Software-Engineering. Die objektorientierte konzeptionelle Modellierung bietet hier eine tragfähige Bindung und ein universelles Sprachmittel, um das Wissen über einen Weltausschnitt ohne Strukturbruch zu repräsentieren und begrifflich durchgängig für die Verarbeitung aufzubereiten.

Ein neues Paradigma muß, will es erfolgreich sein, die Vorteile der alten Paradigmen in sich vereinen. Der Objektansatz, verbunden mit der konzeptionellen Modellierung, bewahrt die Errungenschaften seiner Vorgänger. Er überwindet zugleich die „Komplexitätsbarrieren“ [70] bisher ungelöster Rätsel. Die Ursprache der Objektorientierung, Simula-67, feierte bereits ihr 25-jähriges Jubiläum. Das ist gerade die Mindestzeitspanne laut Thomas Kuhn für einen erfolgreichen Paradigmenwechsel.

Literatur

[1] Booch, G.: *Object Oriented Design with Applications.* The Benjamin/Cummings Publishing Company, Redwood City, 1991

[2] Brachman, R. J.; Levesque H. J. (Hrsg.): *Readings in Knowledge Representation.* Morgan Kaufmann Publishers, San Mateo, Kalifornien, 1985

[3] Brodie, M. L.; Mylopoulos J.; Schmidt, J. W. (Hrsg.): *On Conceptual Modelling.* Springer-Verlag, New York, 1984

[4] Brooks, F. P.: *No Silver Bullet: Essence and Accidents of Software Engineering.* In: Computer, Jg. 20, H. 4, 1987, S. 10–19

[5] Bunge, M.: *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World* und *Vol. 4: Ontology II: A World of Systems.* Reidel, Boston, 1979

[6] Capurro, R.: *Ethik und Informatik: Die Herausforderung der Informatik für die praktische Philosophie.* In: Informatik-Spektrum, Jg. 13, 1990, S. 311–320

[7] Chen, P. P.: *The Entity-Relationship Model: Toward a Unified View of Data.* In: ACM Trans. on Database Systems, Bd. 1, 1976, S. 9–36

[8] Claus, V.: *Entwicklung von Informatik-Methoden.* In: Handbuch der modernen Datenverarbeitung, Jg. 26, H. 150, 1989, S. 26–44

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

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

[11] Codd, E. F.: *Extending the Database Relational Model to Capture More Meaning.* In: ACM Trans. on Database Systems, Bd. 4, 1979, S. 397–434

[12] Dahl, O. J.; Dijkstra, E. W.; Hoare, C. A. R.: *Structured Programming.* Academic Press Inc., New York, 1972

[13] Daniell, J. D.: *An Object Oriented Approach to CAD Tool Control.* Dissertation, Carnegie Mellon University, Pittsburgh, 1989

[14] DeMarco, T.: *Structured Analysis and System Specification.* Yourdon Press, Prentice Hall, 1978

[15] DeRemer, F.; Kron, H. H.: *Programming-in-the-Large versus Programming-in-the-Small.* In: IEEE Trans. on Software Engineering, Jg. 2, H. 2, 1976, S. 80–86

[16] Dijkstra, E. W.: *Go To Statement Considered Harmful.* In: CACM, Jg. 11, H. 3, 1968, S. 147–148

[17] Dijkstra, E. W.: *Programming Considered as a Human Activity.* Proc. of the IFIP Congress, Amsterdam, 1965, S. 213–217

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

[19] Dittrich, K. R.: *Objektorientierte Datenbanksysteme.* In: Informatik-Spektrum, Jg. 12, H. 4, 1989, S. 215–218

[20] Dreyfus, H. L.; Dreyfus, S. E.: *Künstliche Intelligenz: Von den Grenzen der Denkmaschne und dem Wert der Intuition.* Rowohlt, Reinbek, 1987

[21] *Duden Informatik.* Hrsg.: Lektorat des B.I.-Wissenschaftverlags, bearb. von V. Claus und A. Schwill, Mannheim, 1993

[22] Ege, R. K.: *Improving Object-oriented User Interfaces with Constraints.* In: Information and Software Technology, Bd. 33, H. 2, 1991, S. 143–150

[23] Endres, A.: *Software-Wiederverwendung: Ziele, Wege und Erfahrungen.* In: Informatik-Spektrum, Jg. 11, 1988, S. 85–95

[24] Endres, A.; Uhl, J.: *Objektorientierte Software-Entwicklung: Eine Herausforderung für die Projektführung.* In: Informatik-Spektrum, Jg. 15, 1992, S. 255–263

[25] Festinger, L.: *A Theory of Cognitive Dissonance.* Stanford University Press, Stanford CA, 1966

[26] Fiadeiro, J.; Sernadas, C.: *Towards Object-Oriented Conceptual Modeling.* In: Data & Knowledge Engineering, Jg. 6, 1991, S. 479–508

[27] Flores, F.; Winograd, T.: *Understanding Computers and Cognition: A New Foundation for Design.* Ablex Publishing Corporation, Norwood, 1986

[28] Floyd, C.: *Outline of a Paradigm Change in Software Engineering.* In: SIGSOFT Software Engineering Notes, Jg. 13, H. 2, 1988, S. 25–38

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

[30] Gajski, D. D.: *Silicon Compilation.* Addison-Wesley Publishing Company, Reading, Mass., 1988

[31] Gryczan, G.; Züllighoven, H.: *Objektorientierte Systementwicklung: Leitbild und Entwicklungsdokumente.* In: Informatik-Spektrum, Jg. 15, 1992, S. 264–272

[32] Hailpern, B.: *Multiparadigm Laguages and Environments.* In: IEEE Software, Jg. 3, H. 1, 1986, S. 6–13

[33] Harel, D.: *Biting the Silver Bullet: Toward a Brighter Future for System Development.* In: Computer, Jg. 25, H. 1, 1992, S. 8–20

[34] Henderson-Sellers, B.: *A Book of Object-Oriented Knowledge: Object-Oriented Analysis, Design and Implementation, a New Approach to Software Engineering.* Prentice Hall, New Jersey, 1992

[35] Van Horebeek, I.; Lewi, J.: *Algebraic Specifications in Software Engineering.* Springer-Verlag, Berlin Heidelberg, 1989

[36] Kay, A. C.: *Microelectronics and the Personal Computer.* In: Scientific American, Sept. 1977, S. 231–244

[37] Kay, A. C.: *The Early History of Smalltalk.* In: SIGPLAN Notices, Jg. 28, H. 3, 1993, S. 69–95

[38] Kim, W.; Lochovsky, F. H. (Hrsg.): *Object-Oriented Concepts, Databases, and Applications.* Addison-Wesley Publishing Company, Reading, Mass., 1989

[39] Knuth, D. E.: *Computer Programming as an Art.* In: ACM Turing Award Lectures: The First Twenty Years: 1966–1985, Addison-Wesley Publishing Company, Reading, Mass., 1987

[40] Koepke, D. J.: *The Evolution of Software Design Ideas.* Master-of-Science-Arbeit, California State University, Northridge, 1985

[41] Kuhn, T. S.: *Die Struktur wissenschaftlicher Revolutionen.* Suhrkamp, Frankfurt a. M., 10. Aufl., 1989

[42] Luft, A. L.: *Informatik als Technik-Wissenschaft: Eine Einführung für das Informatik-Studium.* BI-Wissenschaftverlag, Mannheim, 1988

[43] March, S. G.; Umphress, D. A.: *Object-Oriented Requirements Determination.* In: Journal of Object-Oriented Programming: Focus on Analysis & Design, Jg. 4, 1991, S. 35–40

[44] Meersman, R. A.; Sernadas A. C. (Hrsg.): *Data and Knowledge.* Proc. of the Second IFIP 2.6 Working Conference on Database Semantics, Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1988

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

[46] Meyer, B.: *The New Culture of Software Development.* In: Journal of Object-Oriented Programming, Jg. 3, H. Nov./Dez., 1990, S. 76–81

[47] Molzberger, P.: *Transcending the Basic Paradigm of Software Engineering.* Hochschule der Bundeswehr München, Bericht-Nr. 8405, 1984

[48] Monarchi, D. E.; Puhr, G. I.: *A Research Typology for Object-Oriented Analysis and Design.* In: CACM, Jg. 35, H. 9, 1992, S. 35–47

[49] Moore, G. E.: *Progress in Digital Integrated Electronics.* In: Proc. of the IEEE Int. Electron Devices Meeting, Talk 1.3, Washington DC, 1975

[50] Norman, D. A.: *Cognitive Engineering.* In: User-Centred System Design: New Perspectives on Human-Computer Interaction, Draper, S.; Norman, D. A. (Hrsg.), Hillsdale, New Jersey, 1986

[51] Naur, P.; Randell, B. (Hrsg.): *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee.* NATO Scientific Affairs Division, Brüssel, 1969

[52] Parnas, D. L.: *On the Criteria to be Used in Decomposing Systems into Modules.* In: CACM, Jg. 15, H. 12, 1972, S. 1053–1058

[53] Quibeldey-Cirkel, K.: *CAD-Frameworks: Die Probleme jenseits der Entwurfswerkzeuge.* In: Mikroelektronik, Jg. 7, H. 2, 1993, S. 72–76

[54] Quibeldey-Cirkel, K.: *Verbundprojekt DASSY: Datentransfer und Schnittstellen für offene integrierte VLSI-Entwurfssysteme.* BMFT-Forschungsbericht, TIB, Hannover, 1993

[55] Quibeldey-Cirkel, K.; Wojtkowiak, H.: *Modellierung des VLSI-Entwurfsprozesses: Rückschau und Ausblick.* Proc. 5. E.I.S.-Workshop, TU Dresden, GMD-Studie Nr. 188, 1991, S. 23–34

[56] Quibeldey-Cirkel, K.: *Das Objekt-Paradigma in der Informatik: ein Kompendium.* Teubner-Verlag, 1994 (in Vorbereitung)

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

[58] Schnupp, P.: *Softwaretechnologie für den kommerziellen Anwender: Bringen die 80er Jahre einen Paradigmenwechsel?.* In: Psychologische Aspekte der Software-Entwicklung, Hrsg: Schelle, H.; Molzberger, P., Oldenburg, München-Wien, 1983, S. 156–171

[59] Séquin, C. H.: *Managing VLSI Complexity: An Outlook.* In: Proc. of the IEEE, Jg. 71, H. 1, 1983, S. 149–166

[60] Shaw, M.: *Abstraction Techniques in Modern Programming Languages.* In: IEEE Software, Jg. 1, H. 4, 1984, S. 10–26

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

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

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

[64] Wand, Y.; Weber, R.: *An Ontological Evaluation of Systems Analysis and Design Methods.* In: Information System Concepts: An In-Depth Analysis, Hrsg.: Falkenberg, E. D.; Lindgreen, P., Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1989, S. 79–107

[65] Wand, Y.: *A Proposal for a Formal Model of Objects.* In: [38, S. 537–559]

[66] Wedekind, H.: *Objektorientierte Schemaentwicklung: Ein kategorialer Ansatz für Datenbanken und Programmierung.* BI-Wissenschaftsverlag, Mannheim, 1992

[67] Weinberg, G. M.: *The Psychology of Computer Programming.* Van Nostrand Reinhold Company, New York, 1971

[68] Weiser Friedman, L.: *From Babbage to Babel and Beyond: A Brief History of Programming Languages.* In: Computer Languages, Jg. 17, H. 1, 1992, S. 1–17

[69] Winblad, A. L.; Edwards, S. D.; King, D. R.: *Object-Oriented Software.* Addison-Wesley Publishing Company, Reading, Mass., 1990

[70] Winograd, T.: *Breaking the Complexity Barriere, Again.* In: SIGPLAN Notices, Jg. 10, H. 1, S. 13–30

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

[72] Wirth, N.: *Program Development by Stepwise Refinement.* In: CACM, Jg. 14, H. 4, 1971

[73] Yourdon, E.: *Managing the Structured Techniques.* 4. Aufl., Prentice Hall, 1988

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