SSOAR

Open Access Repository
www.ssoar.info

Software-Entwicklung zwischen Ingenieur- und Designwissenschaft: Überzeugungskraft und nützliche Widersprüchlichkeit von Software-Engineering und Software-Gestaltung

Schulz-Schaeffer, Ingo

Postprint / Postprint
Sammelwerksbeitrag / collection article

Zur Verfügung gestellt in Kooperation mit / provided in cooperation with:
SSG Sozialwissenschaften, USB Köln

Empfohlene Zitierung / Suggested Citation:
Schulz-Schaeffer, I. (1996). Software-Entwicklung zwischen Ingenieur- und Designwissenschaft: Überzeugungskraft und nützliche Widersprüchlichkeit von Software-Engineering und Software-Gestaltung. In H. D. Hellige (Hrsg.), *Technikleitbilder auf dem Prüfstand : Leitbild-Assessment aus Sicht der Informatik- und Computergeschichte* (S. 115-140). Berlin: Ed. Sigma. https://nbn-resolving.org/urn:nbn:de:0168-ssoar-122184

Nutzungsbedingungen:
Dieser Text wird unter einer Deposit-Lizenz (Keine Weiterverbreitung - keine Bearbeitung) zur Verfügung gestellt. Gewährt wird ein nicht exklusives, nicht übertragbares, persönliches und beschränktes Recht auf Nutzung dieses Dokuments. Dieses Dokument ist ausschließlich für den persönlichen, nicht-kommerziellen Gebrauch bestimmt. Auf sämtlichen Kopien dieses Dokuments müssen alle Urheberrechtshinweise und sonstigen Hinweise auf gesetzlichen Schutz beibehalten werden. Sie dürfen dieses Dokument nicht in irgendeiner Weise abändern, noch dürfen Sie dieses Dokument für öffentliche oder kommerzielle Zwecke vervielfältigen, öffentlich ausstellen, aufführen, vertreiben oder anderweitig nutzen.
Mit der Verwendung dieses Dokuments erkennen Sie die Nutzungsbedingungen an.

Terms of use:
This document is made available under Deposit Licence (No Redistribution - no modifications). We grant a non-exclusive, non-transferable, individual and limited right to using this document. This document is solely intended for your personal, non-commercial use. All of the copies of this documents must retain all copyright information and other information regarding legal protection. You are not allowed to alter this document in any way, to copy it for public or commercial purposes, to exhibit the document in public, to perform, distribute or otherwise use the document in public.
By using this particular document, you accept the above-stated conditions of use.

gesis Leibniz-Institut für Sozialwissenschaften

Mitglied der Leibniz-Gemeinschaft

erschienen in: Hans Dieter Hellige (Hrsg.) (1996): Technikleitbilder auf dem Prüfstand. Leitbild-Assessment aus Sicht der Informatik- und Computergeschichte, Berlin: Edition Sigma, S. 115-140

Software-Entwicklung zwischen Ingenieur- und Designwissenschaft
Überzeugungskraft und nützliche Widersprüchlichkeit von Software-Engineering und Software-Gestaltung

Ingo Schulz-Schaeffer
Technische Universität Berlin, Institut für Sozialwissenschaften, Franklinstr. 28/29, D-10587 Berlin, e-mail: schulz-schaeffer@tu-berlin.de

1. Einleitung

In einem zentralen Praxisfeld der Softwaretechnik, der organisationsbezogenen Software-Entwicklung, zeichnet sich mit zunehmender Deutlichkeit eine auf den ersten Blick paradoxe Situation ab: Die lokale Praxis innerhalb von Projekten der Software-Entwicklung weicht in wesentlichen Hinsichten von der etablierten Methodik des Software-Engineering ab. Dennoch wird diese Methodik selten grundsätzlich in Frage gestellt und neuere Konzepte, die jene lokale Praxis methodisch reflektieren, kommen nur in sehr beschränktem Maße zur Anwendung. Die Rede ist von der vorherrschenden Vorstellung der sequentiellen Abarbeitung von Entwicklungsschritten einer formal spezifizierten und in handhabbare Teilaufgaben zerlegten Problemstellung als Idealfall einer ingenieurmäßig "sauberen" Vorgehensweise; von der lokalen Praxis einer inkrementellen Anpassung von Vorgaben, Zielen und Vorgehensschritten an sich wandelnde oder erst spät erkennbare Erfordernisse; und von der Reflexion dieser Praxis in Konzepten der Software-Gestaltung, die nach Wegen suchen, solche "unscharfen" Anpassungs- und Einbettungsnotwendigkeiten dennoch methodisch in die Software-Entwicklung einbeziehen zu können.

Der vorliegende Beitrag nimmt diese Beobachtung zum Anlaß, nach der Bedeutung konstruktionsleitender Orientierungsmuster in der Software-Entwicklung zu fragen. Theoretischer Ausgangspunkt ist ein dreistufiges Modell kollektiver Orientierungskomplexe der Technikentwicklung, das zwischen Leitbildern, Konstruktionstraditionen und Konstruktionsstilen unterscheidet (Abschnitt 2). Unter Verwendung dieses Modells wird die konstitutive Bedeutung einer ingenieurwissenschaftlichen Perspektive bei der Entwicklung der Softwaretechnik zu einem eigenständigen Technikfeld untersucht (Abschnitt 3) und die Entstehung einer designwissenschaftlichen Sichtweise angesichts der besonderen Probleme organisationsbezogener Software-Entwicklung betrachtet (Abschnitte 4 und 5). Eine Erklärung für die scheinbar paradoxe Situation organisationsbezogener Software-Entwicklung und damit eine vorläufige Antwort auf die Frage nach dem Verhältnis der beiden Sichtweisen ergibt sich allerdings erst, wenn man die theoretische Konzeption erweitert und die Möglichkeit und Nützlichkeit bestimmter Widersprüche zwischen zugleich gültigen Orientierungsmustern in Rechnung stellt (Abschnitt 6).

2. Kollektive Orientierungsmuster der Technikentwicklung

Unter der Bezeichnung als Technikgeneseforschung formiert sich seit gut zehn Jahren ein sozialwissenschaftliches Forschungsprogramm, das sich zum Ziel setzt, "die sozialen Bedingungen der Erfindung, Konstruktion und Entwicklung von Techniken" (Rammert 1993: 9) zu entschlüsseln und Technikentwicklung "als Ergebnis gesellschaftlicher Prozesse zu rekonstruieren" (Dierkes 1990: 313). Inzwischen liegen eine Vielzahl empirischer Untersuchungen vor, die die Bedeutsamkeit sozialer Einflußfaktoren in der Technikentwicklung

belegen und damit die Berechtigung einer Perspektive untermauern, die von der sozialen Konstruiertheit von Technik ausgeht (vgl. Lutz 1987; Rammert 1992). Allerdings ist mit dieser Feststellung allein nicht allzuviel gewonnen, denn das "Daß" der sozialen Konstruiertheit von Technik beantwortet noch lange nicht die Frage nach dem "Wie" (vgl. Joerges 1995: 32).

Innerhalb der Ansätze, die sich um eine Beantwortung dieser Frage bemühen, hat eine Forschungsrichtung aufgrund ihrer breiten Rezeption in den verschiedensten Bereichen der Technikforschung in den letzten Jahren besondere Bedeutung erlangt: die Untersuchung von Technikleitbildern (vgl. Dierkes et al. 1992). In der Diskussion um den Leitbild-Begriff wird jedoch leicht übersehen, daß es sich hier um eine Beschreibung von Orientierungsmustern handelt, die erst in Abgrenzung bzw. Relationierung zu anderen Formen kollektiver Handlungsorientierung die nötige Trennschärfe gewinnt. Statt von einem isolierten Leitbild-Konzept gehe ich deshalb im folgenden von einem dreistufigen Modell kollektiver Orientierungskomplexe unterschiedlicher Funktion und Reichweite aus. Hierzu unterscheide ich mit Rückgriff auf Arbeiten von Constant II, Knie und Dierkes et al. zwischen technikbezogenen Leitbildern, professionsspezifischen Konstruktionstraditionen und lokalen Konstruktionsstilen:

Als Konstruktionstraditionen sollen solche Orientierungsmuster bezeichnet werden, die sich auf der Ebene von Gemeinschaften technischer Praktiker herausbilden und als "herrschender Stand der Technik" manifestieren. Als Praxis-Traditionen umfassen Konstruktionstraditionen diejenigen Methoden und Prinzipien, die von den Praktikern eines bestimmten Technikfeldes, ihren Standesorganisationen und der auf das Technikfeld bezogenen Forschung und Lehre als erfolgreich angesehen und als konsentierter Wissensbestand gepflegt und aufrechterhalten werden. Eine wesentliche Funktion der Etablierung von Konstruktionstraditionen besteht in der Verminderung von Risiken technischer Neuentwicklung, denn die Orientierung an bereits bewährten Verfahrensweisen erzeugt konstruktive Sicherheit. Zugleich verleiht sie dem Konstruktionshandeln ein konservatives Grundmoment. Knie vermutet, "daß solche 'kognitiven Orientierungsmuster' als eine Art 'Filter' wirken, die bereits in der Konzeptionsphase den Gedankengang und die Phantasieentwicklung stark an die bestehenden Realitäten 'zurückbinden' und damit auch für neue Aufgabenstellungen bindend wirken".[1]

Allerdings fallen in der Technikentwicklung in aller Regel eine Vielzahl von Konstruktionsentscheidungen an, für die die jeweils etablierten Konstruktionstraditionen zu unspezifisch sind, um konkret handlungsleitend wirken zu können. Hier übernehmen auf ein Unternehmen oder auf ein Forschungs- und Entwicklungsteam begrenzte Problemlösungsmuster eine wichtige Orientierungsfunktion. Constant II illustriert dies am Beispiel des Unterschieds zwischen einem Honda Civic und einem Lincoln Town Car: Beide Fahrzeugtypen basieren zwar auf den gleichen Praxis-Traditionen des Verbrennungsmotors, der Reifenherstellung, der Fabrikorganisation etc. Nichtsdestotrotz repräsentieren sie sehr unterschiedliche Einschätzungen zweier Autohersteller, worin die Funktionen eines Autos bestehen und sehr unterschiedliche Lösungswege sie bereitzustellen (vgl. Constant II 1987: 232). Solche lokalen Formen der technischen Lösungssuche bezeichne ich mit Knie als Konstruktionsstile (vgl. Knie 1989: 49).

Zwischen diesen beiden Formen von Orientierungsmustern besteht Knie zufolge ein wechselseitiges Bedingungsverhältnis: Konstruktionsstile entstehen durch unternehmensspe-

1. Knie 1990: 99f; vgl. auch ders. 1989: 7, 45ff; ders. 1994: 44; Knie/Helmers 1991: 437; Constant II 1987: 224, 231f.

zifische Anpassung, Interpretation und Weiterentwicklung bestehender Konstruktionstraditionen, neue Konstruktionstraditionen durch Diffusion und Verallgemeinerung lokal erfolgreicher Konstruktionsstile. Die lokale Praxis ist der Ort, an dem die generalisierten Prinzipien und Methoden der Konstruktionstraditionen in konkrete Leitlinien des Konstruktionshandelns umgesetzt werden. Sie ist damit zugleich eine wichtige Geburtsstätte neuer Konstruktionstraditionen. Hier, so beobachtet Knie, "können neue Wissens- und Erfahrungselemente generiert werden, die, zunächst von einem Unternehmen getragen, nur dann dauerhaft stabil bleiben, wenn sie im branchenweiten Verständigungsprozeß und dann als gemeinsam anerkannter 'Stand der Technik' festgeschrieben werden" (ders. 1994: 45).

Wie aber können gemeinsam geteilte Orientierungsmuster der Technikentwicklung entstehen, die über eine Anpassung und Neukombination bestehender Konstruktionstraditionen hinausgehen? Hinsichtlich dieser Frage weist die Unterscheidung von Konstruktionstradition und Konstruktionsstil eine Erklärungslücke auf. Hier bietet das Leitbild-Konzept meines Erachtens eine sinnvolle Ergänzung der voranstehenden Überlegungen. Insbesondere bei grundlegenden technischen Neuentwicklungen ist es unwahrscheinlich, daß innerhalb der lokalen Praxis genügend Erfahrungen vorliegen oder zum Aufbau einer entsprechenden lokalen Praxis in ausreichendem Maße auf gesicherte Konstruktionstraditionen zurückgegriffen werden kann. Vermutet man, daß auch in solchen Situationen bestimmte Orientierungsmuster das Konstruktionshandeln anleiten, so stellt sich die Frage nach "funktionale(n) Äquivalente(n) für noch nicht existierende diskursive Regelsysteme" (Dierkes et al. 1992: 49). Der Konzeption von Dierkes und Mitarbeitern zufolge übernehmen Technikleitbilder diese Funktion. Technikleitbilder entstehen als Vorstellungen über eine angestrebte Technik, die die beteiligten Akteure überzeugt, weil sich dieses Ziel ihnen als wünschbar und zugleich als prinzipiell machbar darstellt (vgl. ebd.: 42ff). Der für meine Überlegungen springende Punkt ist, daß die Überzeugungskraft von Technikleitbildern nicht auf dem gesicherten Fundament vergangener Erfahrung beruht, sondern auf dem grundsätzlich unsicheren Versprechen zukünftigen Erfolgs. Die rhetorische Leistung des Leitbildes besteht dementsprechend darin, die Wünschbarkeit der neuen Technik prägnant zum Ausdruck zu bringen und Zweifel an ihrer Realisierbarkeit zu zerstreuen.

Erweist sich die Unterscheidung von Konstruktionstraditionen und Konstruktionsstilen als unvollständig, so gilt gleiches umgekehrt auch, wenn man den Blick ausschließlich auf Technikleitbilder richtet. Denn selbst bei grundlegenden technischen Neuentwicklungen werden selbstverständlich nicht gleich sämtliche lokalen Handlungsregeln und etablierten Wissensbestände in allen Hinsichten nutzlos. Man muß deshalb sinnvollerweise davon ausgehen, daß es sich bei Leitbildern und erprobten Wissens- und Erfahrungsbeständen nicht nur um ein Substitutionsverhältnis, sondern zugleich auch um ein Ergänzungsverhältnis handelt (vgl. Dierkes 1993: 270). Eine hinreichend differenzierte Betrachtung der Bedeutung von Leitbildern in der Technikgenese gelingt mithin erst auf der Basis von Unterscheidungen, die die Reichweite, Struktur und Funktionsweise auch anderer Komplexe von Handlungsorientierungen berücksichtigen.[2] In dem von mir vorgeschlagenen Modell übernehmen Konstruktionstraditionen die Funktion des kollektiven Gedächtnisses für erfolgreiche Problemlösungsmuster eines Technikfeldes, Technikleitbilder die Funktion, unsichere, aber vielversprechende Abweichungen von diesen Mustern zu motivieren und zu koordinieren. Zwischen diesen beiden Polen haben Konstruktionsstile eine Mittel- und Mittlerstelle inne: Sie

2. Dies wird bei einer isolierten Betrachtung und Kritik des Leitbildkonzeptes leicht übersehen, der Dierkes et al. (1992) mit einer ebenfalls isolierten Darstellung dieses Konzepts allerdings deutlich Vorschub geleistet haben (vgl. Hellige 1993: 196).

repräsentieren einerseits das vom kollektiven Gedächtnis in gewissem Sinne abhängige organisationale Gedächtnis, sind andererseits aber zugleich auch der Ort, an dem die Abweichung von den Erfahrungsbeständen des kollektiven Gedächtnisses erprobt und aus neuen Problemlösungsstrategien gelernt werden kann.

Die nachfolgende Verwendung dieses Modells bei der Untersuchung von Orientierungsmustern der Software-Entwicklung beansprucht weder, alle hier angesprochenen Funktionen und Wechselbeziehungen ausloten zu können, noch, einen vollständige Darstellung der Methodenentwicklung dieses Technikfeldes zu präsentieren. Im Mittelpunkt meiner Betrachtung steht die Frage nach der Wirksamkeit und nach dem Verhältnis zweier übergreifender Versuche, Software-Entwicklung methodisch zu verorten: als Software-Engineering oder als Software-Gestaltung.

3. Software-Engineering als konstituierendes Leitbild einer neuen Profession

Die etablierten Wissensbestände wissenschaftlicher oder technischer Fachgemeinschaften lassen sich dort besonders gut beobachten, wo sie zum Zweck der Einübung zukünftiger Mitglieder niedergelegt sind: in Lehrbüchern (vgl. Fleck 1980: 73ff, 157ff). Betrachtet man die gängigen Einführungen in die Software-Entwicklung, so findet sich insbesondere mit Blick auf zwei Bestimmungsmerkmale weitreichende Übereinstimmung: Zum einen wird Software-Entwicklung fast durchgängig als eine Ingenieurwissenschaft charakterisiert, deren Ziel es sei, auf ökonomische Weise zuverlässige und effiziente Software zu erstellen (vgl. Bauer 1973: 524), eine Charakterisierung, die sich in der Bezeichnung der Ende der sechziger Jahre entstehenden neuen Disziplin als "Software-Engineering" niederschlägt. Diese Bezeichnung transportiert die Forderung, es den bereits etablierten Ingenieurdisziplinen gleichzutun und "sound engineering principles" (ebd.: 524) zu entwickeln und zu benutzen. Sie artikuliert das Ziel "genaue(r) Kenntnis und gezielte(r) Anwendung von Prinzipien, Methoden und Werkzeugen für die Technik und das Management der Software-Entwicklung und -Wartung auf der Basis wissenschaftlicher Erkenntnisse und praktischer Erfahrungen sowie unter Berücksichtigung des jeweiligen ökonomisch-technischen Zielsystems".[3]

Weitgehende Übereinstimmung findet sich zum anderen auch, was die konkrete Umsetzung dieser Forderung anbelangt. Gleichfalls in Anlehnung an die etablierten Ingenieurdisziplinen entwickelte vermutlich zuerst Royce 1970 ein Vorgehensmodell für die Software-Entwicklung (vgl. Sommerville 1989: 6f; Chroust 1992: 42), das sich so schnell als gängige Konstruktionsmethode etablierte, daß es bereits 1976 als "klassisches" bzw. "herkömmliches" Modell der Programm-Entwicklung bezeichnet werden konnte (vgl. Schnupp/Floyd 1976: 149ff), und das - zumeist in der von Boehm vorgenommenen Weiterentwicklung als "Wasserfallmodell" - auch heute noch in kaum einem Lehrbuch fehlt. Konstitutives Merkmal dieser Konstruktionsmethode ist die Aufteilung des Entwicklungsprozesses in eine sequentielle Abfolge von Entwicklungsphasen: "Der Entwicklungsprozeß wird i.a. in eine Reihe von Einzeltätigkeiten zergliedert, deren Ergebnisse einzelne Teilprodukte sind und die den zeitlichen Ablauf der Entwicklung festlegen." (Balzert 1982: 1). Läßt man die Vielzahl kleinerer Unterschiede zwischen den verschiedenen, auch als Kaskaden- oder Lebenszyklusmodelle bezeichneten Phasenkonzepten unberücksichtigt, so ergibt sich die folgende typische Phasenaufteilung: Anforderungsermittlung, Systementwurf, Implementierung (bestehend aus:

3. Gewald et al. 1979: 26; vgl. auch die folgenden Lehrbücher und Einführungen: Kimm et al. 1979: 15f; Boehm 1981: 16f; Balzert 1982: 1ff; Macro/Buxton 1987: 14f; Sommerville 1989: 4f; Nagl 1990: 4; Schulz 1990: 14; Chroust 1992: 18f.

Komponentenentwurf, -kodierung und Integration der Komponenten), Installation und Test sowie Wartung. Jede dieser Phasen wird im Idealfall genau einmal durchlaufen, wobei das Ergebnis einer Phase das Vorprodukt für die jeweils nachfolgende Phase darstellt. Iterationen zwischen jeweils benachbarten Phasen sind in einigen Konzepten vorgesehen (vgl. z.B. Boehm 1981: 35ff), nicht aber Iterationen über mehrere Phasen hinweg.[4]

Historisch betrachtet entstand das Software-Engineering als Reaktion auf die sogenannte "Software-Krise" der sechziger Jahre. Bauer beschreibt die damalige defizitäre Situation folgendermaßen: "Existing software comes too late and at higher costs than expected, and does not fulfill the promise made for it." (Bauer 1973: 524). Als wesentliche Ursache für diese Situation benennt er einen Mangel an methodischen Konstrutktionsprinzipien und -regeln: "The whole trouble comes from the fact that there is so much tinkering with software." (ders. 1993: 259; vgl. auch ders. 1973: 523). Zu einem drängenden Problem wurde dieser Mangel, als wachsende Rechenkapazitäten die Ausführung ständig umfangreicherer Programme ermöglichte. Vor allem für diese größeren Entwicklungsvorhaben erwies sich das Vorgehen auch des gekonntesten Bastlers als unzureichend (vgl. Sommerville 1989: 3; Nagl 1990: 1).

Der Begriff "Software-Engineering", im Vorfeld einer NATO-Tagung in Garmisch von Bauer geprägt (vgl. Bauer 1993: 259), wurde in dieser Situation zur Zauberformel, von der sich ein Großteil der Beteiligten einen Ausweg aus der Krise versprach. Nach Auskunft von Bauer wurde der Begriff "zur Leitformel für die weiteren Beratungen" (ebd.) und Gewald et al. urteilen: "Der Begriff Software Engineering wurde in dieser Zeit im Sinne einer Provokation gegenüber der im Vergleich zu traditionellen technischen Disziplinen wie 'Chemical Engineering', 'Electrical Engineering' u. dgl., unbefriedigenden Vorgehensweise in der Software-Entwicklung und -Wartung gebraucht. Von der Existenz einer eigenen Ingenieurdisziplin konnte noch keine Rede sein; man gebrauchte den Begriff vielmehr zur Charakterisierung einer veränderten Einstellung und Arbeitshaltung zur Erstellung von Software; Software sollte solide, zuverlässig, kontrollierbar und meßbar erzeugt werden können, wie andere Produkte auch." (Gewald et al. 1979: 19f)

Mit der Entwicklung der Prinzipien strukturierter Programmierung, hier insbesondere des Top-Down-Verfahrens zur methodischen Erstellung des Software-Entwurfs (vgl. Bauer 1973: 532f), wird der Grundstein für die Realisierung einer ingenieurwissenschaftlichen Konstruktionsmethodik gelegt. Ausgehend von einer Spezifikation der Anforderungen an das zu entwickelnde Software-Produkt, deren Vollständigkeit, Konsistenz und Unzweideutigkeit vorausgesetzt wird (vgl. Boehm 1976: 1227, 1230), wird das Gesamtproblem so lange in Teilprobleme zerlegt, "bis diese auf unmittelbar einsichtige Weise in maschinell interpretierbare Ausdrücke einer Programmiersprache transformiert werden können" (Luft 1988: 43). Im Software-Entwurf entspricht diese Aufteilung in Teilprobleme einer hierarchischen Aufteilung des angestrebten Software-Systems in Teil-Systeme, Komponenten und Module (vgl. Hesse 1980: 111ff). Das Verfahren der hierarchischen Zerlegung der zu lösenden Aufgabe in zunehmend handhabbarere Teilprobleme führt gleichzeitig zu einer Sequenzialisierung des Entwicklungsprozesses. Die skizzierten Phasenkonzepte weiten diese Entwurfsmethode auf den gesamten Prozeß der Software-Entwicklung aus.

---

4. Eine Übersicht über unterschiedliche Phasenkonzepte bietet Balzert 1982: 469. Vgl. auch Boehm 1976: 1227; Kimm et al. 1979: 18f; Boehm 1981: 35ff; Macro/Buxton 1987: 28ff; Sommerville 1989: 7ff; Spitta 1989: 26f; Nagl 1990: 4ff; Chroust 1992: 42ff.

Die Art, wie der Ausweg aus der Software-Krise der 60er Jahre gesucht und gefunden wurde, zeigt sehr deutlich die Funktion und Wirkungsweise von Leitbildern der Technikentwicklung. Angesichts des unzureichenden Lösungspotentials lokaler Konstruktionsstile (des Herumbastelns) und eines Mangels etablierter Konstruktionsmethoden und -regeln übernimmt eine Vision handlungsleitende Funktionen: die Vision der Software-Entwicklung als einer Ingenieurwissenschaft. Der Begriff des Software-Engineering weist in dieser Situation die zentralen Merkmale eines Leitbildes auf. Er stellt eine vielversprechende, aber erst vage umrissene Möglichkeit zukünftiger Technikentwicklung in Aussicht, bietet selbst aber noch keine gesicherten Problemlösungsverfahren. Als Abweichung von bisherigen Denk- und Handlungsgewohnheiten ist die negative Abgrenzung von der bestehenden Praxis zunächst präziser formulierbar als die positive Charakterisierung des Leitbildes.[5] Seine Stärke liegt, wie Theißing herausarbeitet, "mehr in seiner Imagination als in einer exakten Bestimmung dessen, was kennzeichnend für Ingenieurdisziplinen sei und was davon bei der Softwareentwicklung Erfolg verspräche." (Theißing 1995: 82)

Die Frage nach seiner Wirksamkeit als Leitbild ist zuallererst eine Frage danach, woraus sich die Überzeugungskraft der Vorstellung vom Software-Engineering speist. Technikleitbilder artikulieren lediglich das Versprechen zukünftig erfolgreichen Problemlösens. Ein solches Versprechen ist nur dann in der Lage, Aufmerksamkeit zu erregen, Resourcen zu mobilisieren und Interessen zu bündeln, wenn erstens überzeugend zum Ausdruck gebracht werden kann, daß die Realisierung der Vision gute Erfolgschancen hat (Machbarkeit), und wenn die relevanten Akteure zweitens den Eindruck gewinnen, daß die zukünftige Problemlösung ihren jeweiligen Interessen entspricht (Wünschbarkeit) (vgl. Dierkes et al. 1992: 42f). Der Vision des Software-Engineering ist beides ganz offenkundig gelungen: der Verweis auf bereits etablierte Ingenieurdisziplinen war geeignet, Zweifel an der prinzipiellen Machbarkeit einer entsprechenden Methodik für die Software-Entwicklung zu zerstreuen; die Vision einer sauber strukturierten Vorgehensweise konnte alle diejenigen, die mit der Praxis des ungeordneten Herumbastelns leidvolle Erfahrungen gemacht hatten - und das waren Auftraggeber, Entwickler und Nutzer -, leicht davon überzeugen, daß eine solche Methodik im jeweils eigenen Interesse ist.

Anders als im Leitbildkonzept von Dierkes und Mitarbeitern vorgesehen, ist es allerdings weniger die Bildhaftigkeit, die die Überzeugungskraft der mit dem Begriff des Software-Engineering ausgedrückten Zielvorstellung ausmacht, als vielmehr die rhetorische Prägnanz einer Metapher.[6] Deutlich wird dies, wenn man Metaphern als Sprachfiguren versteht, die durch erstmalige Verbindung zweier bislang getrennter sprachlicher Sinnbezirke neuen Sinn erzeugen, wobei "die Effektivität der Metapher nicht so sehr auf dem Gebiet der Veranschaulichung zu suchen ist ..., sondern in der *Bereicherung* liegt, die sie entweder dadurch liefert, daß sie etwas völlig Neues hervorbringt oder das bislang Vertraute eine neue Komponente zusätzlich erhält" (Hülzer 1987: 251). Der Begriff des Software-Engineering ist in diesem Sinne eine neuartige Kombination bislang getrennter Sinnbezirke, mit der das Produkt der Kunstfertigkeit einer Gemeinschaft findiger Bastler neubestimmt wird als Resultat eines regelgeleiteten Produktionsprozesses. Seine Überzeugungskraft besteht darin, daß die Metapher mit dem Begriff des Engineering auf Konstruktionsverfahren rekurriert, die sich in

---

5. Bauer trifft deshalb ein zentrales Merkmal von Leitbildern, wenn er bezogen auf den Begriff des Software-Engineering auf der NATO-Tagung in Garmisch ironisch anmerkt: "The concept seems to be clear by now, since it has been defined several times by examples of what it is not." (Bauer 1993: 260)

6. Zur Interpretation von Leitbildern als Metaphern vgl. Mambrey, Paetau und Tepper (1995).

anderen Technikfeldern angesichts ähnlich gelagerter Probleme bereits als wünschenswert und machbar erwiesen haben.[7]

Betrachtet man die Methodenentwicklung insbesondere in den 70er Jahren, so ist der Einfluß, den das Leitbild des Software-Engineering auf die Ausarbeitung der Konstruktionsregeln der Software-Entwicklung genommen hat, unübersehbar. Es ist der zentrale Orientierungsrahmen für die Konstruktionsmethodik der Phasenkonzepte und damit das konstituierende Leitbild einer neuen Profession von Software-Ingenieuren. Dies nicht in dem Sinne, daß Konstruktionsmethoden anderer Ingenieurdisziplinen direkt übernommen worden wären, wohl aber insofern als sich die Methodenentwicklung nun der Erarbeitung von Konstruktionsverfahren verschreibt, welche sich an denselben übergreifenden Zielen orientieren wie die dortige Methodik: der erfolgreichen Planung, Strukturierung und Kontrolle von Prozessen der Herstellung komplexer Techniken. Auch heute noch nehmen die Phasenkonzepte sowohl in der Informatikausbildung wie auch in der Praxis der Software-Entwicklung eine herausragende Stellung ein. Sofern es in einem so dynamischen Technikfeld wie dem der Software-Entwicklung überhaupt möglich ist, von etablierten Konstruktionstraditionen zu sprechen, ist es sicherlich diese Konstruktionsmethodik, der eine solche Bezeichnung zusteht.

4. Software-Gestaltung als neues Leitbild einer etablierten Profession?

Chancen für die Entwicklung neuer Leitbilder bestehen jeweils dann, wenn ein bestimmter technischer Problemlösungsbedarf diagnostiziert wird und die Vision einer Problemlösung formuliert werden kann, deren Realisierung als grundsätzlich möglich gilt. Entsprechende Situationen finden sich nicht nur bei der Entstehung neuer Technikfelder, sondern auch dann, wenn im Einflußbereich bereits etablierter Wissensbestände neuartige Probleme auftreten, für deren Bearbeitung diese keinen Orientierungsrahmen bieten. Letzteres scheint bezogen auf die ingenieurwissenschaftliche Konstruktionsmethodik der Software-Entwicklung insbesondere im Übergang von gut strukturierten zu schlecht strukturierten Problemstellungen der Fall zu sein.

Für die Phasenkonzepte des Software-Engineering ist die gut strukturierte Problemstellung eine unabdingbare Voraussetzung: Jeder der folgenden Entwicklungsschritte ist davon abhängig, daß die Anforderungen an das angestrebte Programm zu Beginn der Technikentwicklung vollständig und widerspruchsfrei erfaßt wurden. Dies setzt voraus, daß man die für die Problemlösung relevanten Einflußgrößen kennt, sie in ihrem Verhalten genau beschreiben und mögliche Störfaktoren wirkungsvoll ausschließen kann. Solange mathematische und naturwissenschaftlich-technologische Fragestellungen das hauptsächliche Anwendungsgebiet von Software-Systemen waren, ließen sich entsprechende Bedingungen weitgehend herstellen: Zur Formulierung der Aufgabenstellung konnte auf die formalisierten Beschreibungsmittel der jeweiligen Fachsprache zurückgegriffen werden, die Kontrolle der Kontextvariablen wurde durch die Laborsituation oder durch bereits technisierte Einsatzumgebungen erleichtert.

In weiten Bereichen der Software-Entwicklung für Industrie und Verwaltung dominieren dagegen Problemstellungen, die wesentlich weniger gut strukturiert und wesentlich schwerer strukturierbar sind (vgl. Koslowski 1988: 97). Hier hat man es mit Einsatzfeldern zu tun, in denen "Probleme nicht fest vorgegeben, Anforderungen veränderlich und Softwareprodukte im Einsatz eng mit Arbeits- und Kommunikationsprozessen von Einzelpersonen oder von

7. Ähnliche Verweise auf bereits erprobte Techniken finden sich auch bei anderen technischen Metaphern, etwa der des Schreibklaviers (vgl. Mambrey et al. 1995: 74ff).

Gruppen verzahnt sind" (Floyd 1994: 29). Die heutigen Schwierigkeiten der Software-Entwicklung, soweit sie durch diesen Problemkomplex gekennzeichnet sind, unterscheiden sich mithin deutlich von der Software-Krise der sechziger Jahre. Sie beruhen nicht mehr in erster Linie auf Mängeln bei der methodischen Strukturierung des Entwicklungsvorganges. Das Problem besteht vielmehr darin, die nur schwer formalisierbaren Anforderungen organisationaler Einsatzumgebungen bei der Erstellung von Software zu berücksichtigen.

Organisationsbezogene Software-Entwicklung wird damit zu einer Aufgabe, bei der die technische Problemlösung und ihre Einbettung in einen sozialen Einsatzzusammenhang zwangsläufig miteinander verschmelzen. Eine solche Sichtweise repräsentiert der Begriff der Software-Gestaltung. Hier wird Software-Entwicklung nicht mehr exklusiv als Ingenieurwissenschaft betrachtet, sondern gleichermaßen als Designwissenschaft (vgl. Floyd 1994: 29; Schulz-Schaeffer 1994), also als ein Handlungsbereich, der "sich nicht mit dem Notwendigen (befaßt), sondern mit einem Freiheitsspielraum: nicht damit, wie die Dinge sind, sondern damit, wie sie sein könnten" (Simon 1990: VIII). Freiheitsspielräume für Gestaltungsentscheidungen werden sichtbar, wenn man nicht allein nach den technisch effizientesten Verfahren fragt, sondern mögliche technische Lösungen auch unter dem Kriterium der Angemessenheit im jeweiligen Einsatzzusammenhang auswählt. Dies wiederum setzt voraus, daß man die möglichen Entscheidungen im Entwicklungsprozeß nicht vollständig als durch die Vorgaben einer innertechnischen Rationalität vorgezeichnet ansieht, sondern als einen Gestaltungsprozeß, bei dem mit dem technischen System zugleich auch die sozialen Zusammenhänge seiner Verwendung modelliert werden müssen.

Ansätze, die sich an einer solchen designwissenschaftlichen Vorstellung orientieren, finden sich im Bereich der Software-Entwicklung vereinzelt bereits seit den 70er Jahren. Eine breitere Strömung entstand unter Bezeichnungen wie partizipative, arbeitsorientierte, prozeßorientierte oder evolutionäre Software-Gestaltung allerdings erst im vorigen Jahrzehnt. Handelt es sich bei dem Begriff der Software-Gestaltung um ein neues Leitbild der Software-Entwicklung? Ist es für die Konstruktionsmethodik ähnlich handlungsleitend wie das Leitbild des Software-Engineering? Und in welchem Verhältnis stehen diese neueren Vorstellungen zu jenen Orientierungskomplexen? Bevor ich mich diesen Fragen zuwende, nehme ich eine kurze Sichtung der wichtigsten Ansätze der designwissenschaftlichen Perspektive vor. Ich beschränke mich dabei auf drei Konzeptionen, die der sozio-technischen, der arbeitsorientierten und der evolutionären Gestaltung:

4.1 Sozio-technische Gestaltung

Eine Wurzel der gestaltungswissenschaftlichen Perspektive liegt in den ergonomischen Untersuchungen im britischen Kohlenbergbau, die in den vierziger Jahren von Mitarbeitern des Tavistock-Instituts durchgeführt wurden, um die Ursachen geringer Arbeitsmoral und niedriger Produktivität zu untersuchen. Diese Untersuchungen, bei denen unterschiedliche Formen der Arbeitsorganisation im Zusammenhang mit Techniknutzung verglichen wurden, führte zu dem Ansatz der sozio-technischen Gestaltung. Er geht davon aus, daß der Einsatz von Technik in Organisationen sozio-technische Systeme konstituiert, die aus einer Verbindung technischer Instrumente und Verfahren einerseits und einer sozialen Struktur andererseits bestehen. Die Verbesserung einer dieser beiden Komponenten auf Kosten der jeweils anderen führt - so die Untersuchungen - zu einem suboptimalen Ergebnis. Gestaltungsziel des sozio-technischen Ansatzes ist deshalb die gleichzeitige Optimierung von technischen

und sozialen Komponenten, konkret: die Gestaltung von Arbeitssystemen, die technisch effizient sind und zu hoher Arbeitszufriedenheit führen.[8]

Etwa seit Mitte der siebziger Jahre finden sich Bestrebungen, dieses Konzept im Bereich organisationsbezogener Software-Entwicklung einzusetzen (vgl. Mumford/Welter 1984: 162ff). Das von Mumford vorgeschlagene methodische Vorgehen besteht dabei im wesentlichen aus drei Schritten: der Diagnose von Effizienzbedürfnissen der Organisation und der Festlegung von Effizienzzielen, der Messung der Arbeitszufriedenheit und der Festlegung von Zielen zu deren Verbesserung sowie der Bereitstellung eines institutionellen Rahmens, in dem die erhobenen Effizienz- und sozialen Ziele unter Beteiligung der Betroffenen ausgehandelt werden (vgl. Mumford 1983: 64ff; Mumford/Welter 1984: 202ff). Die angestrebte sozio-technische Lösung umfaßt neben technischen Innovationen auch Veränderungen der Arbeitsorganisation. Die partizipative Ausrichtung dieses Entwicklungskonzeptes ist nach Mumford sowohl aus der Perspektive des an Steigerung der Arbeitsproduktivität interessierten Managements wie auch aus der Perspektive der an Steigerung ihrer Arbeitsqualität interessierten Beschäftigten vorteilhaft. Für die Beschäftigten vergrößert sich der Spielraum der Wahrnehmung eigener Interessen, für das Management zahlt sich die dadurch bewirkte Erhöhung von Arbeitsmotivation und Sachkompetenz in Form erhöhter Arbeitseffizienz aus (vgl. Mumford 1983: 23f, 37ff).

4.2 Arbeitsorientierte Gestaltung

Schon in den frühen sechziger Jahren war der sozio-technische Ansatz in den skandinavischen Ländern von den Gewerkschaften eingesetzt worden mit dem Ziel, die industrielle Demokratie zu fördern. Anfang der siebziger Jahre geriet der sozio-technische Ansatz bei den dortigen Gewerkschaften allerdings in Mißkredit. Unterschiedliche Anwendungsfälle hatten gezeigt, daß die Unternehmerseite bei technischen Innovationen nur in dem Maße bereit waren, soziale Ziele zu berücksichtigen, wie dies in Einklang mit Managementzielen stand. Partizipation bei technischen Innovationen, so deshalb die Position der Gewerkschaften, führe nur dann zur Wahrnehmung von Beschäftigteninteressen, wenn die Beschäftigten selbst ihre Interessen in Hinblick auf technisch veränderte Arbeitsprozesse formulieren können und Ressourcen zu deren Durchsetzung besitzen. Im Gegensatz zu der im sozio-technischen Ansatz enthaltenen Annahme der Harmonisierbarkeit der unterschiedlichen Interessen und Ziele, rechnet das von Ehn und Mitarbeitern entwickelte Konzept mit tiefgreifenden Interessenkonflikten zwischen Beschäftigten und Management, die auch die Software-Entwickler zwingen, für eine Seite Partei zu ergreifen (vgl. Ehn 1988: 263ff; Mumford 1987: 70ff).

Der Ansatz der arbeitsorientierten Gestaltung basiert auf einer Parteinahme zugunsten der zukünftigen Nutzer der angestrebten Software. Stand ab Mitte der siebziger Jahre zunächst die Bereitstellung von Gegenexpertise im Zentrum der Überlegungen, die es den betroffenen Beschäftigten ermöglichen sollte, die Informatisierungsstrategien des Managements besser einschätzen zu können (vgl. Ehn 1988: 281ff), so entwickelte sich ab Beginn der achtziger Jahre eine deutlich offensivere Strategie der Einflußnahme der zukünftigen Nutzer und ihrer Interessenvertreter auf die Software-Entwicklung. Es sollte nicht mehr nur der dequalifizierende Technikeinsatz verhindert, sondern Software-Entwicklung mit dem Ziel der Verbesserung und Vermehrung qualifizierter Arbeit betrieben werden. Gestaltungsziel war es nun, Software als Werkzeug herzustellen, das sich flexibel an die Fertigkeiten der Beschäftigten anpaßt bzw. qualifikationssteigernd wirkt. Mit der Metapher des Computers als Werkzeug

8. Vgl. Mumford 1983; Mumford/Welter 1984: 90ff; Mumford 1987: 62ff; Emery/Trist 1981: 327ff.

konkretisiert Ehn die Vorstellung der Software-Gestaltung in doppelter Hinsicht: Zum einen orientiert er sich an einem handwerklichen Produkt-Ideal, der Idealvorstellung des Werkzeugs, das unter der vollständigen Kontrolle seiner Nutzer steht und dessen handwerkliche Fertigkeiten nicht ersetzt, sondern unterstützt. Zum anderen legt er ein handwerkliches Gestaltungs-Ideal zugrunde, die Idealvorstellung einer Orientierung an der handwerklichen Tradition und den Fertigkeiten der Nutzer bei der Entwicklung neuer Software (vgl. ebd.: 371ff).

Wichtigstes methodisches Hilfsmittel zur Erreichung dieser Gestaltungsziele ist das Verfahren des "Designing-by-Doing". Es beruht auf der Annahme, daß die konkrete alltägliche Arbeitssituation der Beschäftigten der Ausgangspunkt für eine an ihren Fertigkeiten und Interessen orientierten Technikgestaltung sein muß. "Designing-by-Doing" besteht darin, mit einfachen technischen Mitteln die künftigen Arbeitszusammenhänge, die beim Einsatz unterschiedlicher Varianten des geplanten Software-Systems jeweils entstehen würden, zu simulieren. Die künftigen Nutzer werden dadurch in direkter Weise mit den Auswirkungen der technischen Innovation auf ihren Arbeitsalltag konfrontiert und erhalten die Möglichkeit, auf der Basis ihrer eigenen praktischen Erfahrungen und Fertigkeiten Verbesserungsvorschläge zu machen. Umgekehrt dient dieses Verfahren den Entwicklern als Vehikel, um ihr Wissen über technische Möglichkeiten und Grenzen in einer für die Nutzer unmittelbar verständlichen Weise zu transportieren. Es soll damit ermöglichen, die Merkmale des angestrebten Software-Produktes in einem wechselseitigen Lernprozeß gemeinsam zu erarbeiten (vgl. ebd.: 327ff).

4.3 Evolutionäre Gestaltung

Mit dem Ansatz der arbeitsorientierten Gestaltung wird die Einrichtung direkter Rückkopplungsschleifen zwischen dem Entwicklungs- und dem Verwendungskontext zu einem wesentlichen Merkmal gestaltungsorientierter Konstruktionsstile. Die weitere Entwicklung der designwissenschaftlichen Perspektive ist gekennzeichnet durch die Ausarbeitung von methodischen Instrumenten und Verfahren, die der Verbesserung dieses Rückkopplungszusammenhanges dienen. Sie erfährt damit zugleich eine pragmatische Wendung. Die großen Versprechen und die hehren Ziele, die sich sich anfangs mit der Gestaltungsperspektive verbanden, rücken ein wenig in den Hintergrund. Dagegen tritt die konkrete methodische Schwierigkeit der geringen Planbarkeit organisationsbezogener Software-Entwicklung als das zentrale Problem in den Vordergrund, dessen Lösung man sich von einer an den Nutzungszusammenhängen orientierten Vorgehensweise verspricht. Inzwischen liegen eine Vielzahl prozeßorientierter, prototypingorientierter bzw. evolutionärer Ansätze zur Software-Gestaltung vor, die diese Problemstellung bearbeiten. Die Stoßrichtung dieser Ansätze soll hier am Beispiel des von Floyd und Mitarbeitern entwickelten Konzeptes der evolutionären partizipativen Systementwicklung skizziert werden.

Die Annahme einer weitestgehenden Planbarkeit der Software-Entwicklung auf der Basis vorweg ermittelter Anforderungen, wie sie die Phasenkonzepte voraussetzen, wird in dem Maße fraglich, in dem soziale Bedingungen und Folgen im Entwicklungsprozeß mitberücksichtigt werden müssen. Dies erweist sich besonders bei organisationsbezogener Software-Entwicklung zunehmend als erforderlich. Man muß dann damit rechnen, daß unterschiedliche Sichtweisen und konkurrierende Interessenlagen betrieblicher Akteure, schwer formalisierbares Expertenwissen und informelle Organisationsstrukturen es praktisch unmöglich machen, zu Beginn der Software-Entwicklung über ein vollständiges und widerspruchsfreies Bild der an das Software-Produkt gestellten Anforderungen zu gelangen (vgl. z.B. Malsch 1987: 85ff). Auch muß mit Veränderungen der Einsatzorganisation während des Entwicklungsvorhabens -

und zum Teil auch durch den Entwicklungsprozeß selbst - gerechnet werden. Die Angemessenheit einer softwaretechnischen Lösung erweist sich aus diesen Gründen häufig erst bei seinem tatsächlichen Einsatz (vgl. Floyd et al. 1990/91: 37; Sommerville 1989: 11).

Evolutionäre Gestaltung begegnet dieser Problemlage mit einem experimentell ausgerichteten und rekursiv angelegten Projektmodell. Software-Entwicklung wird in einer Folge von Entwicklungszyklen durchgeführt. In jedem Zyklus wird eine Produktvorversion, ein Prototyp, hergestellt und in seinem zukünftigen Einsatzzusammenhang getestet. Der experimentelle Einsatz der jeweiligen Produktversion im angestrebten Verwendungskontext eröffnet den künftigen Nutzern einen Einfluß auf die Software-Gestaltung und ermöglicht die Wahrnehmung bislang unerkannter, unklarer oder veränderter Anforderungen. Durch die zyklische Vorgehensweise können die im Verwendungszusammenhang gewonnenen Ergebnisse wieder in den Entwicklungszusammenhang einfließen. Die daraufhin vorgenommene Weiterentwicklung des Software-Produktes wird dann im folgenden Revisionszyklus ausgetestet. Der evolutionäre Ansatz verspricht somit, trotz unsicheren bzw. unvollständigen Wissens zu Beginn des Entwicklungsprozesses sowie angesichts geringer Planbarkeit zweckangemessene und benutzergerechte Software-Systeme erzeugen zu können (vgl. Floyd et al. 1990/91: 35ff; Floyd 1993: 249f).

5. Leitbildrhetorik: Prägnanz und Überzeugungskraft der Gestaltungsperspektive

Ist Software-Gestaltung das neue Leitbild, das einen Paradigmenwechsel in der Profession der Software-Hersteller einleitet? Betrachtet man die auch weiterhin bestehende Dominanz der Phasenkonzepte als Standardverfahren in den Lehrbüchern und als gängiges Referenzmodell der Projektierung von Software-Entwicklungsvorhaben, so ist es sicherlich verfrüht, von einem solchen Paradigmenwechsel zu sprechen (gegen Floyd 1993). Andererseits weist der Begriff der Software-Gestaltung durchaus Merkmale eines Technikleitbildes auf: Wiederum liegt eine Situation vor, in der die bestehenden Wissens- und Erfahrungsbestände nicht ausreichen, um ein drängendes Problem, das der organisationalen Einbettung von Software-Produkten, in den Griff zu bekommen. Und wiederum ist es eine methodisch zunächst erst vage umrissene Zielvorstellung, die das Problem zu lösen verspricht. Dennoch scheint Software-Gestaltung als Leitbild bei weitem nicht die gleiche Wirksamkeit zu entfalten wie das Leitbild des Software-Engineering.

Untersucht man den Begriff der Software-Gestaltung in seiner Eigenschaft als Leitbild, so ist die Ursache hierfür in einer geringeren Überzeugungskraft zu suchen, darin also, daß es ihm weniger gut gelingt, die relevanten Akteure der Technikentwicklung für die ausgedrückte Zielvorstellung zu gewinnen. Besonders drei Beobachtungen sind in diesem Zusammenhang meines Erachtens von Bedeutung: Erstens leidet die Prägnanz des Leitbildes zumindest anfänglich darunter, daß sich mit dem Begriff der Gestaltung heterogene Gestaltungsziele verbinden. Dies hat zweitens einen Einfluß auf die kollektive Wünschbarkeit der Zielvorstellung des Leitbildes, dann nämlich, wenn die Gestaltungsziele den Interessen jeweils nur bestimmter Akteursgruppen in Entwicklungsprojekten entsprechen. Drittens schließlich fehlt dem Begriff der Software-Gestaltung der metaphorische Verweis auf bereits erprobte Konstruktionstechniken, der in der Metapher des Software-Engineering die Machbarkeit ingenieurmäßiger Software-Entwicklung plausibel erscheinen läßt. Ich betrachte diese drei Punkte im folgenden etwas näher:

Die Gestaltungsperspektive basiert auf der Feststellung, daß sich bestimmte Probleme unbeschadet technischer Funktionalität so oder auch anders lösen lassen. Dies ist für sich

genommen natürlich auch für ingenieurwissenschaftlich ausgerichtete Software-Entwickler nichts Neues. Die Neuheit der Betrachtung von Software-Entwicklung als Designwissenschaft liegt erst in der Vorstellung, diese Gestaltungsspielräume nun auch bewußt in einem sozio-technischen Sinne nutzen zu wollen. Hinter der allgemeinen Zielrichtung, dies zu tun, um die technische und die soziale Seite softwaretechnisch unterstützter Handlungszusammenhänge besser aufeinander abstimmen zu können, verbergen sich, wie die voranstehenden Abschnitte gezeigt haben, jedoch unterschiedliche, und nur zum Teil miteinander vereinbare konkrete Zielvorstellungen: neben dem Ziel der methodischen Reflexion der Bedingungen organisationsbezogener Software-Entwicklung etwa auch Ziele wie das der Humanisierung der Arbeit, das der Stärkung der betrieblichen Mitbestimmung beim Technikeinsatz oder auch das der Effizienzsteigerung betrieblicher Abläufe. Der Begriff der Gestaltung erweist sich in seiner Interpretierbarkeit damit als zu vieldeutig, um die Stoßrichtung der angestrebten methodischen Weiterentwicklung in einer Weise vorzeichnen zu können, in der dies mit dem Begriff des Engineering ganz offensichtlich gelungen ist.

Die Verschiedenheit der möglichen Gestaltungsziele beeinträchtigt das Leitbild der Software-Gestaltung aber nicht nur in seiner Eigenschaft, als prägnante Vision Aufmerksamkeit zu erregen. Sie verringert zugleich auch seine Überzeugungskraft mit Blick auf die Wünschbarkeit der propagierten Ziele. Solange Ansätze der Software-Gestaltung Gestaltungsziele verfolgen, die in erkennbarer Weise den Interessen bestimmter, aber nicht aller der im Entwicklungsprozeß relevanten Akteure dienen, kann das Leitbild seine Funktion als kollektive Projektion, als Fluchtpunkt gemeinsamer Wünsche und Hoffnungen, natürlich nicht erfüllen. Geht man von dem empirisch gut gesicherten Befund aus, daß sich organisationsbezogene Software-Entwicklung gegen den aktiven oder hinhaltenden Widerstand einzelner der relevanten Akteursgruppen in der Einsatzorganisation (Entscheider, zukünftige Nutzer, tangierte Fachabteilungen) nur selten erfolgreich durchführen läßt (vgl. z.B. Behr et al. 1991: 43, 67f), so ist offenkundig, daß ein auf Partialinteressen abgestimmtes Konzept der Software-Gestaltung die Funktion eines Leitbildes nur in sehr begrenztem Maße übernehmen kann.[9]

Solange es keine erprobten Verfahren gibt, die die Machbarkeit der Zielvorstellung dokumentieren, basiert die Einschätzung der Realisierbarkeit eines Technikleitbildes wesentlich auf den Plausibilisierungsleistungen, die das Leitbild selbst zu erbringen in der Lage ist. Besonders wirkungsvoll gelingt dies dort, wo ein Leitbild als Metapher formuliert werden kann, die auf Problemlösungsverfahren verweist, die in anderen Technikfeldern bereits erfolgreich angewendet werden. Das Leitbild des Software-Engineering und eine Vielzahl anderer technischer Metaphern benutzen dieses rhetorische Darstellungsmittel. Zwar kann man auch den Begriff der Software-Gestaltung als eine Metapher interpretieren. Der Bezugspunkt des metaphorischen Vergleichs bleibt jedoch recht unbestimmt. Mit dem Begriff der Gestaltung verbinden sich keine erprobten Konstruktionstechniken, die in ähnlicher Weise per Analogie für die prinzipielle Machbarkeit des Leitbildes ins Feld geführt werden können.

Begründen diese Beobachtungen eine eingeschränkte Wirksamkeit von Software-Gestaltung als Leitbild, so lassen sich auf der anderen Seite Entwicklungen feststellen, die in die

9. Selbstredend ist damit nicht der Umkehrschluß impliziert, daß sich Software-Engineering als Leitbild gegenüber konkurrierenden Interessen neutral verhält. Der Unterschied besteht vielmehr darin, daß die scheinbare Beschränkung eines ingenieurwissenschaftlichen Vorgehens auf technische Fragen eine solche Neutralität suggeriert, während jeder Versuch einer bewußten Gestaltung sozio-technischer Zusammenhänge die Frage danach, wem es nützt, zwangsläufig stellen muß.

entgegengesetzte Richtung weisen. Insbesondere ist eine zunehmende Vereinheitlichung der Gestaltungsziele zu verzeichnen. Wie im vorangegangenen Abschnitt skizziert, stehen innerhalb der designwissenschaftlichen Perspektive inzwischen eher pragmatische Gestaltungsziele im Vordergrund. Ansätze wie der der evolutionären Gestaltung tragen die weitreichenden Ziele der sozio-technischen Modellierung oder der Humanisierung der Arbeit zwar noch in ihrem Wappen, sind in ihren methodischen Überlegungen jedoch sehr viel mehr an den konkreten Problemen orientiert, die sich aus den Bedingungen organisationsbezogener Software-Entwicklung für die Durchführung entsprechender Entwicklungsvorhaben ergeben.

Spätestens hier rekurriert die designwissenschaftliche Perspektive auf ein methodisches Defizit der Praxis organisationsbezogener Software-Entwicklung, an dessen Beseitigung alle relevanten Akteure dieses Technikfeldes interessiert sein müssen. Denn so wie sich in der Software-Krise der sechziger Jahre mangelnde Zuverlässigkeit und Effizienz der Produkte für Auftraggeber, Entwickler und Nutzer gleichermaßen nachteilig auswirkte, so gilt Entsprechendes nun hinsichtlich mangelnder Angemessenheit und Eingebettetheit der Software-Systeme in der Einsatzorganisation.[10] Mit einem Gestaltungsziel, das sich die Lösung dieser Probleme zur vordringlichsten Aufgabe macht, wird Software-Gestaltung zumindest mit Blick auf die Wünschbarkeit der ausgedrückten Zielvorstellung zu einem möglichen Leitbild organisationsbezogener Software-Entwicklung.

Dennoch ist Software-Gestaltung auch nach diesem Reformulierungsschritt bei weitem nicht in dem Maße zu einem Orientierungsrahmen für die Methodik der Software-Entwicklung geworden, wie es das Leitbild des Software-Engineering bis heute ist: Korrespondierende designwissenschaftliche Konstruktionsmethoden bleiben zumeist auf durch akademische Forschung initiierte oder begleitete Entwicklungsvorhaben beschränkt. Selbst im Bereich organisationsbezogener Software-Entwicklung ist das Phasenkonzept in der ein oder anderen Variante nach wie vor das vorherrschende Vorgehensmodell. Dies, obwohl der Erfolg gerade in diesem Bereich der Software-Entwicklung deutlich von einer fortlaufenden Rückkopplung zwischen Herstellungs- und Nutzungskontext abhängt, weil eine solche Vorgehensweise es vielfach überhaupt erst ermöglicht, ein Produkt zu erstellen, das den Bedingungen und Erfordernissen der Einsatzorganisation entspricht. Und obwohl in der lokalen Praxis angesichts solcher Erfordernisse entsprechende zyklisch-inkrementelle Abläufe, die den Vorgaben des offiziellen Projektverlaufs widersprechen, nicht selten zu beobachten sind (vgl. Weltz/Ortmann 1992: 112ff). Wie ist das zu erklären?

6. Zwischen Ingenieur- und Designwissenschaft: nützliche Widersprüche und unerreichbare Ideale

Greift man zur Beantwortung dieser Frage auf das Modell kollektiver Orientierungskomplexe zurück, so liegt es nahe, das Beharrungsvermögen der Phasenkonzepte mit dem konservativen Charakter etablierter Konstruktionstraditionen zu erklären. Allerdings bleibt eine solche Erklärung unbefriedigend. Das Vorgehen technischer Praktiker-Gemeinschaften, bewährtes Wissen zu konservieren und neues Wissen nur zurückhaltend in den Wissenskanon aufzunehmen, begründet zwar ein gewisses "Nachhinken" der Konstruktionstraditionen hinter aktuellen Erfordernissen und Entwicklungen, nicht aber die Abkopplung davon. Ein meines

10. Ein deutliches Indiz für die mangelnde Angemessenheit ist der ständig wachsende Anteil der sogenannten Wartungskosten an den gesamten Entwicklungskosten. Denn sie sind zum großen Teil Kosten der nachträglichen Anpassung des Software-Systems an die Anforderungen des Nutzungskontextes (vgl. Boehm 1976: 1227; Friedrich 1992: 52).

Erachtens sehr viel interessanterer Antwortversuch ergibt sich, wenn man die Annahme aufgibt, daß die in einem Technikfeld als Orientierungsmuster korrespondierenden Leitbilder, Konstruktionstraditionen und Konstruktionsstile vollständig miteinander kompatibel sein müssen. Oder andersherum formuliert: wenn man damit rechnet, daß die Orientierungsleistung bestimmter Komplexe von Leitbildern, Konstruktionstraditionen und Konstruktionsstilen erst aufgrund einer gewissen internen Widersprüchlichkeit erreicht wird.

Die meisten Projekte organisationsbezogener Software-Entwicklung werden nach wie vor nach Maßgabe phasenorientierter Vorgehensmodelle geplant. Betrachtet man jedoch die tatsächliche Durchführung dieser Projekte, so stößt man fast durchgängig auf eine Praxis, die durch zyklisch-inkrementelle Anpassungsleistungen gekennzeichnet ist, ein Vorgehen, das dem Leitbild der Software-Gestaltung viel eher entspricht als dem des Software-Engineering. Eingangs hatte ich dieses Nebeneinander als paradoxe Situation charakterisiert. Im folgenden versuche ich zu begründen, daß es sich hierbei um einen nützlichen Widerspruch handelt. Worin aber besteht der Orientierungsgewinn einer intern widersprüchlichen Kombination des Leitbildes des Software-Engineering und des Phasenkonzepts einerseits und andererseits einer lokalen Praxis, die - wenn auch methodisch wenig reflektiert - so doch Grundzüge einer designwissenschaftlichen Perspektive zum Ausdruck bringt? Um meine Antwort vorwegzunehmen: Die Kombination dieser Orientierungsmuster bietet die Möglichkeit, die Vorteile eines geplanten Vorgehens zu nutzen, ohne anschließend an den Bedingungen tatsächlicher Unplanbarkeit zu scheitern. Dieser Zusammenhang läßt sich anhand von Untersuchungen, die Weltz und Ortmann (1992) durchgeführt haben, deutlich aufzeigen:

Weltz und Ortmann begründen das Festhalten an den Phasenkonzepten mit den Vorteilen dieser Vorgehensweise für die Legitimation und formale Steuerung von Projekten: der Eindeutigkeit der Aufgabenstellung, der Planbarkeit des Projektablaufes sowie der Vorhersehbarkeit und Überprüfbarkeit der projektierten Ergebnisse und Zwischenergebnisse (vgl. Weltz/Ortmann 1992: 171). Daß es sich hierbei um Merkmale handelt, an denen die Auftraggeber und die Entwickler von Software-Produkten ein vitales Interesse haben, liegt auf der Hand. Besonders deutlich aber wird der hohe Stellenwert solcher Möglichkeiten der Stukturierung und die Vorbehalte gegen Vorgehensmodelle, die auf eine durchkomponierte Projektplanung verzichten, wenn man die Präferenzen der zukünftigen Nutzern und indirekt Betroffenen mit Blick auf die eine oder die andere Entwicklungsmethode betrachtet:

Auch bei den zukünftigen Nutzern ist vielfach eine deutliche Zurückhaltung gegenüber Ansätzen der Software-Gestaltung zu beobachten. Und das, obwohl es sich doch um denjenigen Kreis von Akteuren handelt, an den die Bemühungen der designwissenschaftlichen Ansätze um Angemessenheit des Software-Produkts am direktesten adressiert sind. Der Grund hierfür ist wahrscheinlich vor allem in Problemen der Unsicherheit zu suchen. Insgesamt sind Informatisierungsprozesse häufig gekennzeichnet durch ein hohes Maß an Unsicherheit darüber, wer zu den Rationalisierungsgewinnern bzw. -verlierern zählen wird (vgl. Lullies et al. 1990: 102ff). Die Phasenkonzepte versprechen, Unsicherheit durch Planung zu reduzieren, die evolutionären oder inkrementellen Ansätze dagegen müssen den Betroffenen zumuten, mit dieser Unsicherheit zu leben, versprechen dafür aber, Fehlentwicklungen auch in späten Entwicklungsphasen noch revidieren zu können. Kontrastiert mit dem Versprechen früher Gewißheit über die Veränderungen, die die neue Technik für den eigenen Arbeitsplatz mit sich bringt, scheint dies keine sehr attraktive Alternative zu sein.

Daß es sich bei den per Phasenkonzept festgelegten Aussagen über Zeit, Kosten und Ablauf des Projektes "meist um weitgehend fiktive Größen handelt, fällt im konkreten Entscheidungszusammenhang trotz vergangener leidvoller Erfahrungen in vielen Unternehmen offenbar nicht so sehr ins Gewicht. Der Wunsch nach berechenbaren Projekten scheint hier

nicht selten der Vater des Verfahrens zu sein, für den man auch billigend in Kauf nimmt, daß man von vorneherein weiß, daß die Plandaten gar nicht eingehalten werden können." (Weltz/Ortmann 1992: 171f) Für die vorgeschlagene Betrachtung eines Zusammenwirkens widersprüchlicher Orientierungsmuster ist diese letzte Beobachtung - befreit man sie von der im Zitat mitschwingenden negativen Beurteilung - von wesentlicher Bedeutung: Selbst, wenn man um die operationale Unangemessenheit der Phasenkonzepte angesichts der genannten Rahmenbedingungen organisationaler Software-Entwicklung weiß, auf der Planungsebene scheinen sie in ihrer Funktion, eine - wenn auch möglicherweise fiktive - Planungssicherheit herzustellen, dennoch unverzichtbar zu sein. Hier geht es offenbar mehr um den Prozeßwert als um den Produktwert, also eher darum, Planungsprozesse überhaupt in Gang setzen zu können und damit für alle Beteiligten ein gewisses Maß an Handlungssicherheit zu erzeugen, als um ein präzise verwirklichtes Planungsziel.

Dennoch stellt sich die Frage, wie eine solche Fiktion der Planbarkeit aufrechterhalten werden kann, ohne daß die entsprechenden Projekte im Verlauf der Software-Entwicklung an den unplanbaren Anforderungen und Veränderungen der Einsatzorganisation anschließend um so eindrucksvoller scheitern. Hier nun zeigt sich die Bedeutung einer lokalen Praxis, die von der etablierten Methodik deutlich abweicht: jener inkrementellen, zyklischen und durch wechselseitige Lern- und Rückkopplungsprozesse zwischen Entwicklungs- und Einsatzkontext gekennzeichneten Vorgehensschritte, die als "stille" Leistungen der Entwickler (so Weltz/Ortmann 1992: 112ff) dazu beitragen, die Distanz zwischen einem simplifizierten, statischen Vorgehensmodell und der komplexen, dynamischen oranisationalen Wirklichkeit zu überbrücken. Wo also die Situation vorliegt, einerseits trotz Unplanbarkeit Planbarkeit unterstellen zu müssen, um ein entsprechendes Vorhaben projektieren oder organisationsintern überhaupt erst durchsetzen zu können, andererseits jedoch wandelnde Anforderungen, Rahmenbedingungen etc. berücksichtigen zu müssen, um zu einer dem Einsatzkontext angemessenen technischen Lösung zu gelangen, erweist sich die Widersprüchlichkeit dieser beiden gleichzeitig verwendeten Orientierungsmuster als durchaus funktional.

Was folgt aus dieser Betrachtungsweise für die Leitbilder des Software-Engineering und der Software-Gestaltung und deren Einfluß auf das Konstruktionswissen und die Konstruktionspraxis dieses Technikfeldes? Berücksichtigt man die Bedeutung, die der (und zwar auch der fehlerhafte) Nachweis geplanten Vorgehens bei der Durchsetzung und Legitimation organisationsbezogener Software-Entwicklung besitzt, so ist auch in Zukunft kaum damit zu rechnen, daß Vorgehensmodelle, die Unplanbarkeit voraussetzen, in größerem Umfang zur etablierten Konstruktionsmethode werden. Das Leitbild der Software-Gestaltung wird sich dementsprechend gegenüber der ingenieurwissenschaftlichen Perspektive erst dann durchsetzen können, wenn entweder der Gegensatz von Planung und Gestaltung aufgegeben oder aber die Unplanbarkeit organisationsbezogener Software-Entwicklung so offensichtlich würde, daß bereits dadurch jeglicher Planungsversuch desavouiert wäre. Keine dieser beiden Varianten ist besonders wahrscheinlich. Zum einen enthalten zwar auch die inkrementellen Ansätze Planungsaspekte, ihre Eigenständigkeit beruht jedoch auf dem methodischen Umgang mit dem Problem der Unplanbarkeit angesichts eines komplexen Zusammenhangs soziotechnischer Gestaltung. Zum anderen ist selbst im Fall organisationsbezogener Software-Entwicklung mit einer vollständigen Unplanbarkeit nicht zu rechnen, da bereits die Möglichkeit, eine Aufgabenstellung zu formulieren, ein gewisses Maß an Strukturierbarkeit des entsprechenden Entwicklungsvorhabens impliziert.

Wie verhält es sich aber mit der ingenieurwissenschaftlichen Perspektive? Kann man heute überhaupt noch von Software-Engineering als einem Leitbild sprechen? Folgt man dem Leitbildkonzept, so müßte man diese Frage verneinen. Denn so wie Metaphern in dem Maße,

in dem sie in den normalen Sprachgebrauch integriert werden, ihren metaphorischen Gehalt verlieren, so verflüchtigen sich Leitbilder mit zunehmender Realisierung der jeweiligen Vision in den durch sie angestoßenen Konstruktionsregeln, deren funktionales Äquivalent sie anfänglich waren. Daß ein solcher Prozeß der Verwandlung einer Vision in verfügbares Wissen auch hier der Fall ist, sieht man etwa an der Selbstverständlichkeit, mit der das entsprechende Wissensgebiet der Informatik inzwischen als Software-Engineering oder Softwaretechnik bezeichnet wird. Zugleich aber bleibt die Vorstellung einer ingenieurmäßigen Herstellung von Software - jedenfalls bei organisationsbezogener Software-Entwicklung - eine prinzipiell unerreichbare Vision: Denn jeder methodische Versuch, komplexe und veränderliche soziale Zusammenhänge durch Verfahren der Isolierung und eindeutigen Verknüpfung ausgewählter Parameter zu technisieren, erkauft die Lösbarkeit der Aufgabe zwangsläufig mit einem bestimmten Maß an Unangemessenheit der Lösung. In dieser Hinsicht verhält sich Software-Engineering genau komplementär zu Software-Gestaltung, deren prinizipielle Unerreichbarkeit als Vision darin besteht, daß bei dem Versuch, die Komplexität und Dynamik organisationaler Zusammenhänge bei der Technikentwicklung methodisch zu berücksichtigen, eine zunehmende Angemessenheit der Lösung mit einer zunehmenden Unlösbarkeit der Aufgabe korrespondiert.

Zusammengenommen ergibt sich ein Bild, das mit Blick auf die Frage der Wirksamkeit und des Zusammenwirkens kollektiver Orientierungsmuster in der Technikentwicklung weiteren Klärungsbedarf signalisiert. Zum einen zeigt sich, daß das Verhältnis von etablierter Methodik (Konstruktionstradition) und lokaler Praxis (Konstruktionsstil) nicht allein eines von Generalisierung und problemspezifischer Rekombination ist, sondern zugleich ein Verhältnis, mit dem Widersprüche zwischen dekontextualisierten Verfahrensregeln und situationsspezifischen Bedingungen und Anforderungen bearbeitbar werden. In dem hier beschriebenen Fall führt dies dazu, daß eine lokale Praxis, die methodisch ausformuliert einen Gegenentwurf zu den etablierten Verfahren repräsentiert, deren Scheiternsrisiko verringert und damit zumindest kurz- und mittelfristig zu ihrem Fortbestand als Konstruktionstradition beiträgt.

Zum anderen deutet sich an, daß der Begriff des Leitbildes in einer wichtigen Hinsicht weiter ausdifferenziert werden muß, nämlich mit Blick auf den Aspekt der Machbarkeit: Im Gegensatz etwa zu dem Leitbild des Schreibklaviers beeinflussen bestimmte technikbezogene Leitbilder wie das des Software-Engineering oder der Software-Gestaltung Technikentwicklung offensichtlich, obwohl sie nach jedem Schritt, sie umzusetzen, zumindest teilweise unrealisiert bleiben. Das heißt, sie sind als Leitbilder Idealbilder, an die man sich bestenfalls annähern kann.[11] Sobald sich aber die durch ein solches Orientierungsmuster mobilisierten Akteure dessen bewußt werden, verliert das Leitbild als Orientierungsrahmen an Exklusivität. Auch wenn man das Idealbild im Grundsatz weiterhin für erstrebenswert hält, kommt man nicht umhin anzuerkennen, daß es von idealisierten Annahmen ausgeht, daß also dort, wo sich diese Annahmen nicht bestätigen, Grenzen seiner Realisierbarkeit bestehen, die nach anderen Lösungswegen verlangen. Betrachtet man Software-Engineering und Software-Gestaltung als Idealbilder, die sich hinsichtlich der jeweils vorausgesetzten idealisierten Annahmen und den angezielten Lösungswegen zueinander komplementär verhalten, so wird deutlich, daß die Auseinandersetzung zwischen ingenieurwissenschaftlicher und designwis-

11. Ein solches, sich ständig entziehendes Leitbild kann sich, so beobachtet Woolgar am Beispiel der Künstlichen Intelligenz, forschungspolitisch geradezu als ideale Strategie erweisen: "the argument that any specific appearance of 'intelligence' may not turn out actually to be 'intelligence' provides the AI community with a seemingly endless research programme" (Woolgar 1985: 564).

senschaftlicher Perspektive in absehbarer Zeit wohl kaum mit dem Sieg oder der Niederlage einer der beiden Sichtweisen enden wird.

Literatur:

Balzert, Helmut (1992): Die Entwicklung von Software-Systemen. Prinzipien, Sprachen, Werkzeuge, Mannheim u.a.: BI-Wissenschaftsverlag

Bauer, Friedrich L. (1973): Software Engineering, in: ders. (Hg.), S. 522-545

Bauer, Friedrich L. (1993): Software Engineering - wie es begann, Informatik-Spektrum 16, S. 259-260

Bauer, Friedrich L. (Hg.) (1973): Software Engineering. An Advanced Course, New York u.a.: Springer-Verlag

Behr, Michael / Martin Heidenreich / Gert Schmidt / Hans-Alexander Graf von Schwerin (1991): Neue Technologien in der Industrieverwaltung. Optionen veränderten Arbeitskräfteeinsatzes, Opladen: Westdt. Verlag

Biervert, Bernd / Kurt Monse (Hg.) (1990): Wandel durch Technik? Institution, Organisation, Alltag, Opladen: Westdt. Verlag

Bijker, Wiebe E. / Thomas P. Hughes / Trevor Pinch (Hg.) (1987): The Social Construction of Technological Systems. New Directions in the Sociology and History of Technology, Cambridge, Mass. u.a.: MIT Press

Boehm, Barry W. (1976): Software-Engineering, in: IEEE Transactions on Computers 25, S. 1226-1241

Boehm, Barry W. (1981): Software Engineering Economics, Englewood Cliffs, N. J.: Prentice-Hall

Chroust, Gerhard (1992): Modelle der Software-Entwicklung, München u.a.: Oldenbourg Verlag

Colburn, Timothy R. / James H. Fetzer / Terry L. Rankin (Hg.): Program Verification. Fundamental Issues in Computer Science, Dordrecht: Kluwer Academic Publishers

Constant II, Edward W. (1987): The Social Locus of Technological Practice: Community, System or Organisation?, in: W. Bijker et al. (Hg.), S. 232-242

Dierkes, Meinolf (1990): Technikgenese: Einflußfaktoren der Technisierung jenseits traditioneller Technikfolgenforschung, in: B. Biervert / K. Monse (Hg.), S. 311-331

Dierkes, Meinolf (1993): Die Technisierung und ihre Folgen. Zur Biographie eines Forschungsfeldes, Berlin: Edition Sigma

Dierkes, Meinolf / Ute Hoffmann / Lutz Marz (1992): Leitbild und Technik. Zur Entstehung und Steuerung technischer Innovationen, Berlin: Edition Eigma

Ehn, Pelle (1988): Work oriented Design of Computer Artifacts, Stockholm: Almqvist & Wiksell

Emery, Frederick E. / Eric L. Trist (1981): Socio-technical Systems, in: F. E. Emery (Hg.), S. 322-337

Emery, Frederick E. (Hg.): Systems Thinking. Selected Readings, Bd 1, Revised Edition, Middlesex, England u.a.: Penguin Education

Fleck, Ludwig (1980): Entstehung und Entwicklung einer wissenschaftlichen Tatsache. Einführung in die Lehre vom Denkstil und Denkkollektiv, Frankfurt/Main: Suhrkamp

Floyd, Christiane (1993): Outline of a Paradigm Change in Software Engineering, in: T. R. Colburn et al. (Hg.), S. 239-259

Floyd, Christiane (1994): Software-Engineering - und dann?, in: Informatik-Spektrum 17, S. 29-37

Floyd, Christiane / M. Castner / Reinhard Keil-Slawik / Jürgen Pasch / Fanny-Michaela Reisin / Gerhard Schmidt (1990/91): Einführung in Software Engineering STEPS, Arbeitsunterlagen zur Lehrveranstaltung, TU Berlin

Friedrich, Jürgen (1992): CASE-Tools und Software Factories - Software-Entwicklung als Fabrikarbeit?, in: G. Trautwein-Kalms (Hg.), S. 44-74

Gewald, Klaus / Gisela Haake / Werner Pfadler (1979): Software Engineering. Grundlagen und Technik rationeller Programmentwicklung, 2. verbesserte Aufl., München u.a.: Oldenbourg Verlag

Halfmann, Jost / Gotthard Bechmann / Werner Rammert (Hg.) (1995): Technik und Gesellschaft. Jahrbuch 8: Theoriebausteine der Techniksoziologie, Frankfurt/Main u.a.: Campus Verlag

Hellige, Hans Dieter (1993): Von der programmatischen zur empirischen Technikgeneseforschung: Ein technikhistorisches Analyseinstrumentarium für die prospektive Technikbewertung, in: Technikgeschichte 60(3), S. 186-223

Hellige, Hans Dieter (Hg.) (1994): Leitbilder der Informatik und Computer-Entwicklung, artec-Paper Nr. 33, Bremen

Hesse, Wolfgang (1980): Das Projektmodell - Eine Grundlage für die ingenieurmäßige Software-Entwicklung, in: R. Wilhelm (Hg.), S. 107-122

Hülzer, Heike (1987): Die Metapher. Kommunikationssemantische Überlegungen zu einer rhetorischen Kategorie, Münster: Nodus Publikationen

Joerges, Bernward (1995): Prosopopoietische Systeme. Probleme konstruktivistischer Technikforschung, in: J. Halfmann et al. (Hg.), S. 31-48

Kimm, Reinhold / Wilfried Koch / Werner Simonsmeier / Friedrich Tontsch (1979): Einführung in Software Engineering, Berlin u.a.: de Gruyter

Knie, Andreas (1989): Das Konservative des technischen Fortschritts. Zur Bedeutung von Konstruktionstraditionen, Forschungs- und Konstruktionsstilen in der Technikgenese, WZB, FS II, 89-101, Berlin

Knie, Andreas (1990): Was leistet die Technikgeneseforschung? Der "herrschende Stand der Technik" als unsichtbarer "Käfig" im Entstehungsprozeß neuer technischer Artefakte, in: R. Tschiedel (Hg.), S. 91-105

Knie, Andreas (1994): Gemachte Technik. Zur Bedeutung von "Fahnenträgern", "Promotoren" und "Definitionsmacht" in der Technikgenese, in: W. Rammert / G. Bechmann (Hg.), S. 41-66

Knie, Andreas / Sabine Helmers (1991): Organisationen und Institutionen in der Technikentwicklung. Organisationskultur, Leitbilder und "Stand der Technik", in: Soziale Welt 42, S. 427-444

Koslowski, Knut (1988): Unterstützung von partizipativer Systementwicklung durch Methoden des Software Engineering, Opladen: Westdt. Verlag

Luft, Alfred L. (1988): Informatik als Technikwissenschaft. Eine Orientierungshilfe für das Informatik-Studium, Mannheim u.a.: BI-Wissenschaftsverlag

Lullies, Veronika / Heinrich Bollinger / Friedrich Weltz (1990): Konfliktfeld Informationstechnik: Innovation als Managementproblem, Frankfurt/Main u.a.: Campus Verlag

Lutz, Burkhart (1987): Das Ende des Technikdeterminismus und die Folgen - soziologische Technikforschung vor neuen Aufgaben und neuen Problemen, in: ders. (Hg.), S. 34-52

Lutz, Burkhart (Hg.) (1987): Technik und sozialer Wandel, Verhandlungen des 23. Dt. Soziologentages in Hamburg 1986, Frankfurt/Main u.a.: Campus Verlag

Macro, Allan / John Buxton (1987): The Craft of Software Engineering, Wokingham u.a.: Addison-Wesley

Malsch, Thomas (1987): Die Informatisierung des betrieblichen Erfahrungswissens und der "Imperialismus der instrumentellen Vernunft". Kritische Bemerkungen zur neotayloristischen Instrumentalismuskritik und ein Interpretationsvorschlag aus arbeitssoziologischer Sicht, in: Zeitschrift für Soziologie 16(2), S.77-91

Mambrey, Peter / Michael Paetau / August Tepper (1995): Technikentwicklung durch Leitbilder. Neue Steuerungs- und Bewertungsinstrumente, Frankfurt/Main u.a.: Campus Verlag

Mumford, Enid (1983): Designing Human Systems for New Technology. The ETHICS Method, Manchester: Manchester Business School

Mumford, Enid (1987): Sociotechnical Systems Design. Evolving Theorie and Practice, in: G. Bjerknes et al. (Hg), S.59-76

Mumford, Enid / Günter Welter (1984): Benutzerbeteiligung bei der Entwicklung von Computersystemen, Verfahren zur Steigerung der Akzeptanz und Effizienz des EDV-Einsatzes, Berlin: Erich Schmidt Verlag

Nagl, Manfred (1990): Softwaretechnik: Methodisches Programmieren im Großen, Berlin u.a.: Springer-Verlag

Rammert, Werner / Gotthard Bechmann (Hg.) (1994): Technik und Gesellschaft. Jahrbuch 7: Konstruktion und Evolution von Technik, Frankfurt/Main u.a.: Campus Verlag

Rammert, Werner (1992): Entstehung und Entwicklung der Technik: Der Stand der Forschung zur Technikgenese in Deutschland, in: Journal für Sozialforschung 32(2), S. 177-208

Rammert, Werner (1993): Technik aus soziologischer Perspektive, Opladen: Westdt. Verlag

Schnupp, Peter / Christiane Floyd (1976): Software. Programmentwicklung und Projektorganisation, Berlin u.a.: de Gruyter

Schulz, Arno (1990): Software-Entwurf. Methoden und Werkzeuge, 2. Aufl., München u.a.: Oldenbourg Verlag

Schulz-Schaeffer, Ingo (1994): Informatik als Designwissenschaft? Techniksoziologische Überlegungen zur Entwicklung gestaltungsorientierter Konstruktionsstile im Software-Engineering, in: H. G. Hellige (Hg.), S. 277-303

Simon, Herbert A. (1990): Die Wissenschaften vom Künstlichen, Berlin: Kammerer & Unverzagt

Sommerville, Ian (1989): Software Engineering, 3rd Edition, Wokingham u.a.: Addison-Wesley Publishing Company

Spitta, Thorsten (1989): Software Engineering und Prototyping. Eine Konstruktionslehre für administrative Softwaresysteme, Berlin: Springer-Verlag

Theißing, Florian (1995): Auf dem Weg in die Softwarekrise? Computeranwendungen und Programmentwicklung in den USA der fünfziger und sechziger Jahre, Forschungsberichte des Fachbereichs Informatik der TU Berlin, 95-14

Trautwein-Kalms, Gudrun (Hg.) (1992): KontrastProgramm Mensch-Maschine. Arbeiten in der HighTech-Welt, Köln: Bund-Verlag

Tschiedel, Robert (Hg.): Die technische Konstruktion der gesellschaftlichen Wirklichkeit, München: Profil

Weltz, Friedrich / Rolf G. Ortmann (1992): Das Softwareprojekt. Projektmanagement in der Praxis, Frankfurt/Main: Campus Verlag

Wilhelm, Reinhard (Hg.): GI - 10. Jahrestagung, Saarbrücken 30. September - 2. Oktober 1980, Informatik-Fachberichte hg. von W. Bauer im Auftrag der Gesellschaft für Informatik (GI), Berlin u.a.: Springer-Verlag

Woolgar, Steve (1985): Why not a Sociology of Machines? The Case of Sociology and Artificial Intelligence, in: Sociology 19, S. 557-572