Partizipative Konzepte, Methoden und Techniken zur Optimierung der Softwareentwicklung

Reisin, F.-M., 1990: Kooperative Gestaltung in partizipativen Software-Projekten. Dissertation am FB 20. Berlin: TU Berlin

Brödner, P. / Simonis, G. / Paul, H. (Hg.): Arbeitsgestaltung und partizipative Systementwicklung. Opladen: Leske + Budrich, 1991

Matthias Rauterberg

Partizipative Konzepte, Methoden und Techniken zur Optimierung der Softwareentwicklung[1)]

1. Einleitung

Analysiert man aktuelle Softwareentwicklungsprozesse, so erkennt man eine Reihe von Problemen und Schwachstellen. Die Ursachen hierfür sind sowohl in den verwendeten theoretischen Konzepten und den traditionellen Vorgehensweisen (insbesondere Projektmanagement), als auch unzureichenden Kostenrechnungsmodellen begründet. Aufgrund verschiedener Ansätze in der konkreten Praxis der Softwareentwicklung liegt bereits eine zahlreiche Sammlung von Lösungsmöglichkeiten in der Literatur vor, welche auf die Bedeutung der Partizipation aller betroffenen Gruppen hinweist. Bei der Analyse dieser Fallbeispiele lassen sich drei wesentliche Barrieren erkennen: die Spezifikationsbarriere, die Kommunikationsbarriere und die Optimierungsbarriere.

Eines der wichtigsten Probleme besteht ganz allgemein darin, ein gemeinsames Verständnis aller betroffenen Personengruppen über den zu automatisierenden Anteil im betroffenen Arbeitssystem herzustellen (Weltz / Lullies / Ortmann 1991), also gemeinsam verbindliche Antworten auf die Fragen "ob", "wo" und "wie" des geplanten Technikeinsatzes zu finden. Hierzu gehört insbesondere die Bestimmung aller Eigenschaften des neu zu gestaltenden Arbeitssystems. Jedes Arbeitssystem besteht aus einem sozialen und einem technischen Teilsystem (Ulich 1991:151f). Beide Teilsysteme gilt es gemeinsam in einem optimalen Gesamtsystem zu integrieren.

2. Barrieren im Rahmen traditioneller Softwareentwicklung

Für die optimale Gestaltung des gesamten Arbeitssystems kommt es vorrangig darauf an, das soziale Teilsystem als ein mit für dieses Teilsystem spezifischen Ei-

1) Der vorliegende Beitrag entstand im Rahmen des Forschungsprojekt BOSS - Benutzer-orientierte Softwareentwicklung und Schnittstellengestaltung (Förderungskennzeichen 01 HK 706-0), das vom BMFT (AuT-Programm) gefördert wird.

genschaften und Bedingungen ausgestattetes System zu betrachten, welches es in seiner Verkopplung mit dem technischen Teilsystem zu optimieren gilt: "human resources are at the core of future growth and Europe's innovation capability" (CEC 1989:2). Die verschiedenen Arten der Verkoppelungen lassen sich z.B. mittels des Kontingenzmodells von Cummings und Blumberg (1987) bestimmen (siehe hierzu Ulich 1989). Dabei kommt der Arbeitsaufgabe als 'Schnittpunkt' zwischen der Organisation und dem Individuum eine zentrale Rolle zu (Volpert 1987:14). Mindestens die drei folgenden, miteinander verschränkten Barrieren gilt es bei der Gestaltung von Arbeitssystemen unter Einsatz moderner Technologie zu überwinden.

2.1 Die Spezifikationsbarriere

Als ein scheinbar zunächst vordergründiges Problem hat sich die "Spezifikationsbarriere" herausgestellt: wie kann der Softwareentwickler sicherstellen, daß der Auftraggeber seine Anforderungen an das zu entwickelnde technische Teilsystem vollständig, zutreffend und während der Projektdauer unverändert benennen kann. Je formaler und detaillierter das Darstellungsmedium ist, in dem der Auftraggeber seine Anforderungen formuliert, desto einfacher ist dann für den Softwareentwickler die Umsetzung dieser Anforderungen in ein entsprechendes Softwaresystem. Dieser Wunsch setzt jedoch ein Wissen und eine Qualifikation seitens des Auftraggebers voraus, welche dieser sich vor Beginn eines Softwareentwicklungsprozesses anzueignen nicht bereit, z.T. auch nicht in der Lage ist. Es müssen also andere Wege und Methoden gefunden und eingesetzt werden: über informale, semi-formale, hin zu formalen Spezifikationsmethoden (Pomberger et al. 1987:31ff; Sommerville 1989:91f).

Es ist oft ein folgenschwerer Irrtum, davon auszugehen, daß Auftraggeber - meistens Personen des mittleren und höheren Managements - über *alle* Anforderungen an ein interaktives Softwaresystem sinnvoll und adäquat Auskunft geben können (Pomberger et al. 1987:12ff). Folgende unterschiedliche Sichten müssen daher in der Analyse- und Spezifikationsphase berücksichtigt werden (Spinas / Waeber 1991:39).

Die Anwender-Sicht

Als Anwender werden alle Personen angesehen, die über die Anforderungen an das gesamte Arbeitssystem Aussagen treffen können. Diese Sicht ist oftmals die Sicht des Auftraggebers. Unter diese Sicht fallen allgemeine Anforderungen hinsichtlich der Effektivität, der organisationalen Strukturen, der Projektkosten, des globalen Einsatz- und Verwendungszweckes des technischen Teilsystems, sowie

der erhofften Auswirkungen auf das gesamte Arbeitssystem. Die Anwender-Sicht enthält somit die Anforderungen an die Organisationsschnittstelle (Dzida 1987).

Die Benutzer-Sicht

Die Gruppe der Benutzer setzt sich aus allen Personen zusammen, welche die Ergebnisse, die mittels des Softwaresystems erstellt werden, für die Erledigung ihrer Arbeit benötigen. Die Sicht dieser Gruppe über das Softwaresystem ist durch die Mensch-Mensch-Kommunikation mit den End-Benutzern geprägt (z.B. ChefIn ↔ SekretärIn, etc.). Die Benutzer-Sicht umfaßt in der Regel Anforderungen an die Werkzeugschnittstelle.

Die End-Benutzer-Sicht

Die Gruppe der End-Benutzer setzt sich aus allen Personen zusammen, von denen das Softwaresystem unmittelbar als Werkzeug zur Unterstützung ihrer Arbeit eingesetzt wird. Diese Gruppe kann sinnvoll Anforderungen an die Organisations-, die Werkzeug-, die Dialog- und die Ein- / Ausgabeschnittstelle stellen.

2.2 Die Kommunikationsbarriere

Die Kommunikationsbarriere zwischen Anwender, Benutzer, und End-Benutzer einerseits und Softwareentwickler andererseits liegt im wesentlichen in der unzureichenden Einbettung der "technischen Intelligenz" in die sozialen, historischen und politischen Zusammenhänge der Technikentwicklung begründet (Mai 1990:12). Die nicht-technischen Fakten in der Kommunikation zwischen allen am Entwicklungsprozeß Beteiligten "fallen durch das begriffliche Raster der technischen Fachsprache, die somit den gesellschaftlichen Charakter der Technik auf das Funktionale und Instrumentale beschränkt" (Mai 1990:13).

Die jeweilige Fachsprache prägt nicht nur den konkreten Kommunikationsprozeß, sondern auch die ihm zugrunde liegenden kognitiven Strukturen. Die anwendungsbezogene Fachsprache der Benutzer bricht sich an der technischen Fachsprache der Entwickler (Carroll 1988:158; Melzer 1989:113). Dieser "Bruch" läßt sich nur begrenzt mit rein sprachlichen Mitteln überbrücken, da die verwendeten Begriffe aufgrund ihrer kontextuell gebundenen Semantik unscharf sind. Um diese Unschärfe zu überwinden, müssen gemeinsam erlebte, sinnlich erfahrbare Kontexte hergestellt werden. Hierbei sind neben der verbalen Kommunikation im wesentlichen visuelle Kommunikationsmittel geeignet. Je stärker der semantische Kontext des jeweils Anderen sinnlich erfahrbar wird, desto besser läßt sich die Kommunikationsbarriere überwinden (vgl. auch Reisin, in diesem Band).

2.3 Die Optimierungsbarriere

Softwareentwicklung ist in der Regel ein Verfahren zur optimalen Gestaltung eines Produktes mit interaktiven Eigenschaften zur Unterstützung von Aufgabenbearbeitungsprozessen. Nachdem die Informatik einen Fundus an vielfach verwendbaren Algorithmen und Software-Werkzeugen erarbeitet hat, rückt der nicht-algorithmisierbare Anteil von anwendungsbezogener Software zunehmend in den Brennpunkt von Softwareentwicklung. Lassen sich die rein technischen Anteile des Software-Produktes mit primär auf den technischen Kontext abgestimmten Optimierungsverfahren bewältigen, so müssen für den nicht-technischen Kontext der jeweils angestrebten Anwendungsumgebung andere Optimierungsverfahren eingesetzt werden.

Optimieren heißt, alle nur beschränkt zur Verfügung stehenden Mittel im Rahmen eines ökonomischen, technischen und sozialen Prozesses so einzusetzen, daß unter den gegebenen Bedingungen das beste Ergebnis erzielt wird. Optimierungsbarrieren können in mehrfacher Hinsicht bestehen.

Es ist ein Irrtum davonauszugehen, daß zu Beginn einer größeren Umgestaltung eines Arbeitssystems irgendeine Gruppe von Personen eine vollständige, zutreffende und erschöpfende Sicht auf das Soll-Konzept des zu erstellenden Arbeitssystems haben kann. Erst im Laufe des Analyse-, Bewertungs- und Gestaltungsprozesses können alle an diesem Prozeß beteiligten Personen ein zunehmend klareres Bild von dem Soll-Konzept entwickeln (siehe auch Pomberger et al. 1987:13). Dies ist der wesentliche Grund dafür, daß sich die Anforderungen des Anwenders an das Soll-Konzept scheinbar "verändern"; sie verändern sich nicht, sondern sie *konkretisieren* sich unter den jeweils antizipierten Randbedingungen. Dieser Konkretisierungsprozeß sollte so vollständig, zutreffend und - insgesamt gesehen - so kostengünstig wie irgendmöglich vonstatten gehen. Die Vollständigkeit kann durch die repräsentative Beteiligung aller betroffenen Personengruppen gewährleistet werden; durch ein iterativ-zyklisches Vorgehen wird das Soll-Konzept zunehmend konkreter; der Einsatz von Methoden zur Unterstützung des Kommunikationsprozesses garantiert ein effizientes Vorgehen (Keil-Slawik 1989:131f; Nielson 1989, 1990).

3. Die Überwindung der Barrieren

Inzwischen gibt es ausreichend empirische Belege dafür, daß durch arbeits- und benutzer-orientiertes Vorgehen bei der Softwareentwicklung nicht nur nachweislich Kosten eingespart, sondern auch signifikant angemessenere Softwareprodukte entwickelt werden (Boehm / Gray / Seewaldt 1981; Bewley et al. 1983; Gomaa 1983; Baroudi / Olson / Ives 1986; Foidl / Hillebrand / Tavolato 1986; Peschke 1986; Jones 1987; Mantei / Teorey 1988; Spinas / Ackermann 1989; van der Schaaf 1989; Karat 1990; Strohm 1991a b; Rauterberg 1991). Wie lassen sich die drei oben erwähnten Barrieren nun im einzelnen überwinden? Ausgehend von der Beschreibung eines allgemeinen Optimierungszyklus werden die verschiedenen notwendigen Aktivitäten im Softwareentwicklungsprozess in der prinzipiell richtigen Reihenfolge, wie sie im "Wasserfall"-Modell beschrieben werden, vorgestellt.

3.1 Der Optimierungszyklus

In der Systemtheorie wird zwischen den beiden Lenkungsprinzipien der "Steuerung" und der "Regelung" unterschieden. Voraussetzung dafür, daß im Falle der "Steuerung" die angestrebte Anpassung der Steuergröße an die *Führungsgrößen* erreicht wird, ist mindestens:

" *(1) die genaue Kenntnis der Reaktion des gelenkten Systems (der Beziehung zwischen Steuergrößen einerseits, Stell- und Störgrößen (z.B. Änderungen der Anforderungen) andererseits;*

(2) die genaue Kenntnis der die beabsichtigte Beeinflussung beeinträchtigenden auf das System einwirkenden Größen (Störgrößen, z.B. technische Realisierbarkeit, etc.); reagiert das System mit Zeitverzögerung, ist eine Prognose der Störgrößen für mindestens diesen Zeitraum nötig;

(3) die Kenntnis von Verfahren, um aus diesen Informationen Stellgrößen abzuleiten.

Diese Voraussetzungen sind praktisch nie erfüllt. Deshalb wird man stets Steuerung durch Regelung ergänzen oder ersetzen müssen" (Schiemenz 1979: 1024).

Zu der gleichen Schlußfolgerung im Rahmen von Softwareentwicklungsprozessen kommen auch Floyd und Keil (1983:144), sowie Peschke (1986:143ff). Die Verwendung des sehr leistungsfähigen Lenkungsprinzips der "Regelung" setzt im Grunde nur die Kenntnis von Stellgrößen voraus, die sich auf die Regelgröße der Tendenz nach in der gewünschten Richtung auswirken. "Bei Regelung berücksichtigt das lenkende System die Ergebnisse des Lenkungsvorganges. Es findet also eine Rückkopplung des Ausganges des gelenkten Systems zum Eingang des lenkenden Systems statt" (Schiemenz 1979:1024). Die Rückkopplung zwischen jeweils zwei aufeinander folgenden Phasen wird vielerorts bereits heute schon gefordert (Sommerville 1989:10).

Der Unterschied zwischen den Randbedingungen und den Störgrößen besteht darin, daß die Randbedingungen bewußt und zielorientiert vorgegeben werden (z.B. Entwicklungskosten, Projektdauer, etc.), wohingegen die Störgrößen unbeabsichtigt und unvorhersehbar auftreten (siehe Abb. 1). Wir bezeichnen den "Test-

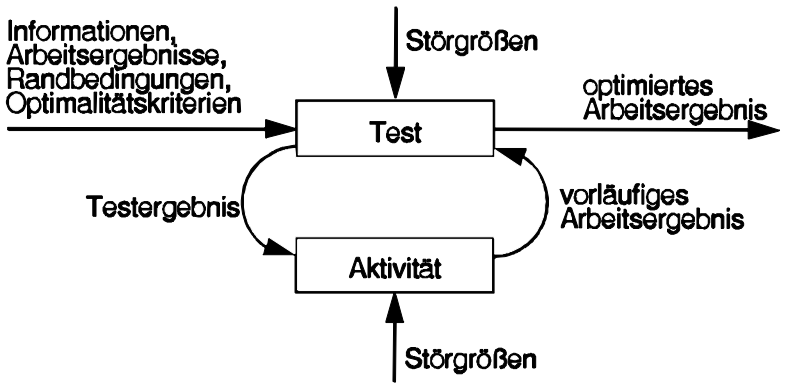

Abb. 1: Der Optimierungszyklus im Rahmen eines iterativ-zyklischen Softwareentwicklungskonzepts (nähere Erläuterungen im Text; siehe auch Peschke 1986:143ff).

Aktivitätszyklus" als Optimierungszyklus. Eine wesentliche Beschreibungsgröße des Optimierungszyklus ist die benötigte Zeit für einen Durchlauf. Je nach Beschaffenheit der Aktivität und der Testung ergeben sich Durchlauf-Zeiten von Sekunden bis Jahren. Je größer die Durchlaufzeit ist, desto kostspieliger ist der jeweilige Optimierungszyklus. Ziel benutzer-orientierter Softwareentwicklung ist es nun, möglichst effiziente Optimierungszyklen in den Software-Entwicklungsprozeß einzubauen (Nielson 1989, 1990; Karat 1990).

Übertragen wir das allgemeine Schema für Regelung auf den Kontext der Softwareentwicklung, so lassen sich die einzelnen Bestandteile der Regelung (wie in Abb. 1 dargestellt) zuordnen. Als Optimalitätskriterien gelten alle relevanten technischen und sozialen Faktoren. Die Testung überprüft, in wie weit die Optimalitätskriterien unter Einhaltung der Randbedingungen erfüllt sind. Als Aktivität können sehr unterschiedliche Verfahren, Methoden, Techniken in Frage kommen. Dies richtet sich nach der Art des Arbeitsergebnisses. Als Störgrößen können unter anderem die drei Barrieren, als auch technische und soziale Realisierungsprobleme angesehen werden.

Natürlich kommt auch das Lenkungsprinzip der "Steuerung" in aktuellen Softwareentwicklungen an verschiedenen Stellen zur Anwendung. Hierbei ist an Entscheidungen und Vorgaben durch den Auftraggeber, die Projektleitung oder andere Führungsgremien etc. zu denken, welche aus Erfahrung, Unwissenheit, Machtdemonstration oder schlicht Zeitdruck erfolgen. "In vielen Fällen arbeiten Steuerungssysteme wirtschaftlicher als (immer auch mögliche) Regelungssysteme"

(Buhr / Klaus 1975:1035) - aber nur, wenn die oben genannten Voraussetzungen zutreffen! Dies ist auch der gewichtige Grund dafür, warum versucht wird, möglichst ein gesteuertes System herzustellen, sprich: dem "Wasserfall"-Modell so nahe wie möglich zu kommen. Nehmen wir jedoch die oben aufgeführten Barrieren ernst, so gilt es zu entscheiden, an welchen Stellen im Softwareentwicklungsprozeß Optimierungszyklen *unabdingbar* sind (siehe hierzu auch Peschke 1986:143ff; Rettig 1991).

3.2 Die Analyse-Phase

Die Analyse-Phase ist die oft am meisten vernachlässigte Phase (Waeber 1991). Dies liegt im wesentlichen darin begründet, daß hier primär Methoden und Techniken eingesetzt werden müssen, wie sie die Arbeits- und Organisationswissenschaft entwickelt hat und anwendet (Macaulay et al. 1990). Der Nutzen dieser Methoden wird von Softwareentwicklern nach eigenen Angaben signifikant unterschätzt (Kraut / Streeter 1991). Die Fehlerbehebungskosten, die durch eine suboptimale Analyse entstehen, sind unverhältnismäßig hoch (Foidl / Hillebrand / Tavolato 1986). Es ist an der Zeit, für eine optimale Softwareentwicklung speziell ausgebildete Personen aus Arbeits- und Organisationswissenschaft für die Analysephase einzusetzen! Im Rahmen der soziotechnischen Systemkonzeption sind die folgenden Bedingungen für das Entstehen von Aufgabenorientierung unabdingbar (Emery 1959, zitiert nach Ulich 1991:155):

1. die arbeitende Person muß Kontrolle haben über die Arbeitsabläufe und die dafür benötigten Hilfsmittel.
2. Die strukturellen Merkmale der Aufgabe müssen so beschaffen sein, daß sie in der arbeitenden Person Kräfte zur Vollendung oder Fortführung der Arbeit auslösen.

In der Analyse-Phase steht somit die Gestaltung der Aufgaben im Zentrum ("Primat der Aufgabe", Ulich 1991:154). Um eine angemessene Gestaltung der Aufgaben zu erreichen, müssen die fünf Gestaltungsmerkmale "Ganzheitlichkeit", "Anforderungsvielfalt", "Möglichkeiten der sozialen Interaktion", "persönliche Autonomie" und "Lern und Entwicklungsmöglichkeiten" angestrebt werden (Ulich 1991). Bevor jedoch eine Gestaltungsmaßnahme durchgeführt werden kann, muß das Arbeitssystem analysiert und bewertet werden. Es lassen sich die folgenden drei Ebenen für die Analyse eines gegebenen Arbeitssystems unterscheiden.

(1) Analyse der Arbeitsaufträge

Das folgende Konzept der Auftrags- und Bedingungsanalyse beschreibt die schrittweise vertiefende Analyse von Arbeitsaufträgen (Hacker / Matern 1980). Die Auftragsanalyse dient der Gewinnung von organisationalen Gestaltungsvorschlägen und gliedert sich in sieben Schritte:

1. Gliederung des Produktionsprozesses und der betrieblichen Rahmenbedingungen.
2. Identifizierung des Arbeitsprozesses innerhalb des Produktionsprozesses.
3. Auflisten der Eigenschaften des zu bearbeitenden Produktes bzw. des zu steuernden Prozesses.
4. Analyse der Arbeitsteilung zwischen den Beschäftigten.
5. Beschreibung der Grobstruktur der Arbeitsaufträge.
6. Festlegung der objektiven Freiheitsgrade bei der Bewältigung der Arbeitsaufträge.
7. Erfassung der Häufigkeiten von identischen und seltenen Arbeitsaufträgen pro Arbeitsschicht.

Als Methoden werden eingesetzt: Analyse betrieblicher Dokumente (die Dokumentenanalyse); die sich ergebenden Informationen werden durch stichprobenartige Beobachtungen von Arbeitsabläufen und Befragungen ergänzt (das Beobachtungsinterview); spezifische Informationen können nur über ausführliche Befragungen betrieblicher Spezialisten erhalten werden (das Experteninterview) (siehe auch Macaulay et al. 1990).

Wenn es wichtig ist, auch die Umgebungsbedingungen des zu analysierenden Arbeitssystems in die Analyse mit einzubeziehen, dann empfiehlt es sich gemäß der soziotechnischen Systemanalyse vorzugehen (Mumford / Welter 1984, Ulich 1991).

(2) Analyse der Arbeitstätigkeiten

Für die Erarbeitung von arbeitsplatzbezogenen Gestaltungsvorschlägen ist es oft unumgänglich, eine Tätigkeitsanalyse durchzuführen (Ulich 1991:72ff). Sie liefert Kenntnisse über Abläufe, Auftrittshäufigkeiten und Zeitanteile der einzelnen Teiltätigkeiten. Folgende drei Schritte sind dabei zu beachten:

1. Analyse der Ablaufstruktur der Arbeitstätigkeit hinsichtlich der enthaltenen Teiltätigkeiten.
2. Entwicklung eines Kategoriensystems zur präzisen Erfassung aller Teiltätigkeiten.
3. Erfassung der Art, Auftrittshäufigkeit und Zeitanteil der einzelnen Teiltätigkeiten über die gesamte Arbeitsschicht der ArbeitnehmerInnen hinweg.

Während die globale Analyse der Arbeitsaufträge in dem zu analysierenden Arbeitssystem im Rahmen traditioneller Softwareentwicklung teilweise zum Tragen kommt, wird die Analyse der Arbeitstätigkeiten und der Auswirkungen dieser Tätigkeiten weitgehend außer acht gelassen. Einen systematischen Überblick über mögliche Analyse- und Gestaltungsmaßnahmen gibt Upmann (1989:114). Für den Bereich der Bürotätigkeiten bietet sich das VAB-Verfahren an (Debusmann 1984).

(3) Analyse der Auswirkungen auf das Befinden und Erleben der BenutzerInnen

Die Analyseergebnisse zu den objektiven Bedingungen eines Arbeitssystems müssen stets durch die subjektiven Bedingungen der ArbeitnehmerInnen ergänzt werden, um bei den gemeinsam mit allen Beteiligten zu erarbeitenden Gestaltungsmaßnahmen Konsens erzeugen zu können. In der Studie von Yaverbaum und Culpan (1990) konnten durch den Einsatz des "Job Diagnostic Survey (JDS)" (Hackman / Oldham 1975) wichtige Anhaltspunkte für weitere Qualifizierungs- und Gestaltungsmaßnahmen gewonnen werden.

3.3 Die Spezifikationsphase

Nach der erfolgreichen Analyse des zu optimierenden Arbeitssystems gilt es die gewonnenen Ergebnisse in eine implementierungsnahe Form zu giessen (Martin 1988). Hierzu sind Spezifikationsmethoden mit hohem kommunikativem Wert einzusetzen (Beck / Ziegler 1991:83) (siehe z.B. "RFA"-Netze von Oberquelle 1987).

(1) Die Spezifikation der Organisationsschnittstellen

Als erstes muß bestimmt werden, "ob" und "wo" ein sinnvoller Einsatz moderner Technologie möglich ist (Malone 1985). "Die Vorstellung, man könne mit Hilfe der Technik die Defizite einer Organisation beheben, ohne die Strukturen der Gesamtorganisation in Frage zu stellen, ist zwar noch immer weitverbreitet, aber zumeist ein Trugschluß" (Klotz 1991:108). Wichtig ist, das Arbeitssystem als eine lebendige Organisation, als einen sich selbst erhaltenden Organismus zu begreifen, der sich entwickeln und verändern muß, um die Organisationsziele zu erreichen. Unter dieser Perspektive geht es bei der Festlegung der Organisationsschnittstelle primär darum, die Lebensfähigkeit der Organisation durch Einsatz moderner Technologie zu fördern; dabei ist es unumgänglich, die notwendigen Gestaltungsmaßnahmen so zu treffen, daß eine größtmögliche Orientierbarkeit aller Individuen gewährleistet ist. "Nur durch eine verknüpfte Optimierung des technischen und sozialen Teilsystems kann das ganze System optimal gestaltet werden" (Baitsch et al. 1989:32).

Die Auswirkungen der umgesetzten Gestaltungsmaßnahmen lassen sich z.B. mit dem "Tätigkeitsbewertungssystem (TBS)" (Hacker / Iwanowa / Richter 1983), bzw. "Tätigkeitsbewertungssystem für «geistige Arbeit»" (Rudolph / Schönfelder/ Hacker 1988) testen.

(2) Die Spezifikation der Werkzeug-Schnittstellen

Bei der Spezifikation der Werkzeugsschnittstelle wird die intendierte Festlegung zur Mensch-Maschine-Funktionsteilung vorgenommen (Beck / Ziegler 1991). Die Aufgaben, welche in Menschenhand verbleiben, müssen sich durch folgende Merkmale auszeichnen (Zölch / Dunckel 1991):

1. ausreichender Handlungs- und Entscheidungsspielraum;
2. angemessener zeitlicher Spielraum;
3. ausreichende körperliche Aktivitäten;
4. konkreter Kontakt zu materiellen und sozialen Bedingungen des Arbeitshandelns;
5. tatsächliche Beanspruchung vielfältiger Sinnesmodalitäten;
6. Variationsmöglichkeiten bei der Erledigung der Arbeitsaufgaben;
7. aufgabenbezogene Kommunikation und unmittelbare zwischenmenschliche Kontakte.

Die Auswirkungen der umgesetzten Gestaltungsmaßnahmen lassen sich z.B. mit Hilfe der "Kontrastiven Aufgabenanalyse (KABA)" (Zölch / Dunckel 1991) testen. Bei Hacker, Müller-Rudolph und Schwarzer-Schönfelder (1989) wird ein Verfahrenspaket vorgestellt, das eine kooperative Aufgabenanalyse beim Einführen oder Verbessern rechnerunterstützter geistiger Arbeit beschreibt.

(3) Die Spezifikation der Ein- / Ausgabe-Schnittstellen

Nachdem einigermaßen Klarheit unter allen Beteiligten darüber besteht, welche Funktionen automatisiert werden können, empfiehlt es sich, zunächst mit den End-Benutzern das Bildschirm-Layout mittels Handskizzenentwurf (sehr preiswerte "Papier & Bleistift"-Methode; Wulff / Evenson / Rheinfrank 1990) auszutesten. Bei sehr umfangreichen Mengen unterschiedlicher Masken kann eine Bilddatenbank die mittels Graphik-Editor erstellten Masken verwalten (Martin 1988:79). Der Einsatz von Prototyping-Tools für diesen Zweck ist oftmals deshalb unangemessen, weil die "tool"-spezifischen Darstellungsmöglichkeiten zu begrenzt sind (Pomberger et al. 1987:63).

Die Auswirkungen der umgesetzten Gestaltungsmaßnahmen lassen sich durch Diskussionen mit den End-Benutzern, bzw. mittels Check-Listen (Baitsch / Katz / Spinas / Ulich 1989:172) austesten.

(4) Die Spezifikation der Dialog-Schnittstellen

Zur Spezifikation der Dialog-Schnittstelle ist es unabdingbar, Prototypen zur Veranschaulichung der dynamischen, interaktiven Aspekte des zu entwickelnden Werkzeuges einzusetzen (Melzer 1989). Prototypen sollten nur ganz gezielt und zweckgebunden zur Abklärung spezieller Spezifikationsaspekte eingesetzt werden, weil ansonsten die unausweichliche Gefahr besteht, zuviel Aufwand in den Ausbau und den Erhalt von "Anschauungsprodukten" zu stecken (Kieback et al. 1991). Bei der Auswahl von "Interface-Development-Tools" kann die Checkliste von Hix und Schulman (1991) behilflich sein. Eine sehr effiziente und zudem preiswerte Variante ist die Verwendung von Simulationsstudien, z.B. mittels handgemalter Folien, Karten, etc., welche dem Benutzer in Abhängigkeit von seinen Aktionen vom Testleiter vorgelegt werden (Karat 1990).

Die Auswirkungen der umgesetzten Gestaltungsmaßnahmen lassen sich z.B. mit dem Verfahren VERA / B-Teil 4 (Rödiger 1987), mittels spezifischer Performanztests (van der Schaaf 1989) oder mit Checklisten (Baitsch et al. 1989:166-171) austesten.

3.4 Die Implementationsphase

Nachdem in der Analyse- und Spezifikationsphase der notwendige Optimierungsaufwand investiert wurde, kann man nun zuversichtlich in die Implementationsphase einsteigen. Die Implementationsphase gliedert sich in die folgenden drei Schritte (Boehm 1981):

1. Entwurf der Programm-Architektur;
2. Entwurf der einzelnen Programm-Module (-Objektklassen, etc.);
3. Kodierung und "Debugging".

Es ist wichtig, den Entwurf von der Spezifikation zu unterscheiden (Sommerville 1989). Während in der Spezifikation möglichst präzise alle relevanten Eigenschaften des technischen Teilsystems festgelegt werden, muß in der Implementationsphase dafür Sorge getragen werden, daß das zu entwickelnde technische Teilsystem möglichst alle diese Eigenschaften aufweist. Hier kommen primär rein software-technische Kenntnisse zum Tragen: einfache und saubere Systemstruktur, Entkopplung von Bausteinen, Minimalität und Abstraktheit von Schnittstellen, Vermeidung globaler Daten, Bildung abstrakter Systemsichten, etc. (Sommerville 1989).

Vor der eigentlichen Kodierungsphase ist zu prüfen, inwieweit sich nicht schon vorhandene Software wiederverwenden läßt (Jones 1987; Prieto-Diaz 1991). Sollte dies nur beschränkt oder garnicht möglich sein, ist zu testen, welche Program-

mier-, bzw. Entwicklungsumgebung ("CASE workbench") die größtmögliche Effizienz verspricht (Schönthaler / Nemeth 1990). Zu einer Entwicklungsumgebung gehört zunächst im Zentrum ein Sammelbehälter ("information repository"), in den durch die folgenden Werkzeuge Dokumente und Produkte abgelegt, bzw. entnommen werden: das "data dictionary", Werkzeuge zum Suchen im Sammelbehälter (der "Browser" bei Schmidt / Pomberger / Bauknecht 1989), Werkzeuge zur Erzeugung strukturierter Diagramme, Werkzeuge zur Entwurfsanalyse und -testung, entwurfsgesteuerte Programmgeneratoren, Maskengeneratoren, Reportgeneratoren, Im- & Exportmöglichkeiten. Es lassen sich durch den Einsatz derartiger Entwicklungsumgebungen Produktivitätssteigerungen bis zu 40% erreichen (Chikofsky / Rubenstein 1988). In wie weit objekt-orientierte Programmierung den Implementationsaufwand verringert, bleibt vorerst abzuwarten (Wirfs-Brock / Johnson 1990).

3.5 Die Benutzungsphase

Nach Erstellung einer lauffähigen Version kann diese zu Benutzbarkeitsstudien ("benutzungsorientierte Benchmarktests", Spencer 1985; Karat in: Helander 1988:891ff; Rauterberg 1991) im konkreten Arbeitskontext eingesetzt werden. Erst zu diesem Zeitpunkt lassen sich alle Probleme mit dem aktuellen organisatorischen und technischen Umfeld abklären. Auf die Notwendigkeit von empirischen Evaluationstechniken im konkreten Arbeitskontext - im Gegensatz zu Labor-Tests - weisen Whiteside, Bennett und Holtzblatt (in: Helander 1988:805ff) hin. Diese Feldstudien tragen im Unterschied zu Labor-Studien dem Aspekt der "ökologischen Validität" Rechnung (Karat 1990:352). Voraussetzung für den Einsatz dieser Methoden ist eine lauffähige Systemversion, welche nur im Rahmen eines Versionen-Konzeptes erstellbar ist.

Durch die konkrete Benutzung im realen Aufgabenkontext - sozusagen unter "Echtzeit" Bedingungen - kann der erreichte Anpassungsgrad an den geplanten organisatorischen Soll-Zustand mittels Interview, Fragebogen, Benchmarktest, etc. überprüft und getestet werden. Daten über die Benutzung lassen sich auf Video aufnehmen (Vossen 1991), durch den Testleiter protokollarisch festhalten oder automatisch in Logfiles abspeichern (Müller-Holz et al. 1991). Obwohl Videoaufaufnahmen die informationsreichsten Datenaufzeichnungen sind, hat sich eine Kombination aus Logfile und direkter Protokollierung als ein guter "Kompromiß zwischen Leistungsfähigkeit und Ökonomie" herausgestellt (Müller-Holz et al. 1991:418).

4. Ein partizipatives Softwareentwicklungskonzept

In der Literatur kann man eine Reihe von Vorschlägen zur Einbettung von Optimierungszyklen in den Softwareentwicklungsprozeß finden (Floyd / Keil 1983; Iivari 1984; Budde et al. 1986; Mambrey / Oppermann / Tepper 1986; Eason / Harker 1987; Grudin / Ehrlich / Shriner 1987; Krüger 1987; Pomberger et al. 1987; Rödiger 1987; Boehm 1988; Rüesch 1989; Sommerville 1989; Macaulay et al. 1990; Schönthaler / Nemeth 1990:283ff). Je nach Erfahrungshintergrund werden von den einzelnen Autoren in ihren Konzepten unterschiedliche Schwerpunkte gelegt.

4.1 Der Einstieg in den globalen Optimierungszyklus

Die beim Start und beim Durchlauf einzelner Optimierungszyklen zu beachtenden Aspekte werden im folgenden diskutiert. Als eine wesentliche Rahmenbedingung von Softwareentwicklung hat sich der Typ der zu entwicklenden Software herausgestellt (Waeber 1990). Es lassen sich die folgenden vier Typen unterscheiden:

Typ-A: Spezifische Anwendung für firmen-interne Fachabteilung; Fachabteilung und Softwareentwicklungsabteilung gehören derselben Firma an.

Typ-B: Spezifische Anwendung für externe Anwender; Fachabteilung und Softwareentwicklungsabteilung gehören unterschiedlichen Firmen an.

Typ-C: Standardbranchenlösung für externe Anwender; dieser Typ geht oftmals aus Projekten des Typs A oder B hervor, indem spezifische Anpassungen von Individualsoftwarelösungen (Typ-A, B) für weitere Anwender vorgenommen werden.

Typ-D: Standardsoftware für weitgehend anonymen Anwenderkreis.

Der Einstieg in den globalen Optimierungszyklus (siehe Abb. 2) bei einer Neuentwicklung erfolgt über den Start-A, wohingegen bei einer Weiterentwicklung der Einstieg bei Start-B erfolgt. Je nach Projekttyp kommen kontextspezifisch unterschiedliche lokale Optimierungszyklen zur Optimierung spezifischer Arbeitsergebnisse zum Einsatz. Die Entscheidung über die jeweils aktuelle Vorgehensweise gehört zur Aufgabe des Projektmanagements und schlägt sich in der gewählten Aufbauorganisation nieder.

4.2 Globale und lokale Optimierungszyklen

Der Einsatz von Optimierungszyklen im Softwareentwicklungsprozeß ist unter anderem an folgende Voraussetzungen gebunden (Peschke 1986:149):

" 1. *Ein geändertes Projekt-Management-Modell, das vor allem die Kommunikation zwischen Betroffenen und Entwicklern sicherstellt.*
2. *Eine rechnergestützte Versions- und Dokumentationsverwaltung, die auch Ergebnisse der Evaluation und aktuelle Kritik aufnimmt.*
3. *Die Information aller Beteilgten über Projektziele und Besonderheiten der Vorgehensweise, sowie eine Schulung der Betroffenen.*
4. *Der grundsätzlichen Bereitschaft der Entwickler, unvollständige Software zu produzieren und Kritik dazu entgegenzunehmen.*[2)]
5. *Der Erweiterung der Kenntnisse der Entwickler über rein DV-technisches Wissen hinaus bzgl. Maßnahmen der Arbeitsstrukturierung.*
6. *Der Einsatz einer weitgehend integrierten Software-Tool-Umgebung, die den Entwickler in der mehrmaligen Erstellung und Änderung der Software unterstützt.*
7. *Der Bereitschaft aller Beteiligten, während des Projektes zu lernen."*

Wenn wir davon ausgehen, daß alle genannten Voraussetzungen weitgehend erfüllt sind, so bleibt dennoch offen, wie das Softwareentwicklungsprojekt konkret durchzuführen ist. Um die Ziele eines arbeitsorientierten Gestaltungskonzeptes (Ulich 1991:215ff) zu erreichen, empfiehlt es sich, die ersten Projekt-Phasen ("requirements analysis and definition") mit einer Vielzahl von Optimierungszyklen anzufüllen. Nach einer möglichst umfassenden Abklärung der Anforderungen des Auftraggebers (Arbeits- und Aufgaben-Analysen, Mensch-Computer-Funktionsteilung, etc.) wird das System-Design festgelegt. Zu fragen bleibt, welche Design-Spezifikationen noch durch weitere Optimierungszyklen abgeklärt werden müssen. So konnte Melzer (1989) zeigen, daß zwar der größte Teil an Mängeln bezüglich Ein- / Ausgabe-Schnittstelle und Anwendungskomponente während der Design-Phase erkannt und beseitigt werden können, aber die Mängel der Dialog-Komponente erst an einer ablauffähigen Version, bzw. an einem entsprechenden Prototypen festgestellt werden können.

Einfachere und schnellere Techniken zum Benutzer-Einbezug sind: Diskussionsgruppen unter zu Hilfenahme verschiedener kommunikativer Techniken (Meta-Plan, Handskizzen-Entwürfe, "Screen-Dumps", Szenarien, etc.; Spinas/ Ackermann 1989), Fragebögen zu Einstellungen, Meinungen, Anforderungen der Benutzer, die "walk-through"-Technik zur systematischen Abklärung aller möglichen Arbeitsschritte, sowie gezielte Interviews zur konkreten Analyse des Arbeits-

2) siehe hierzu "egoless programming" (Weinberg 1971)

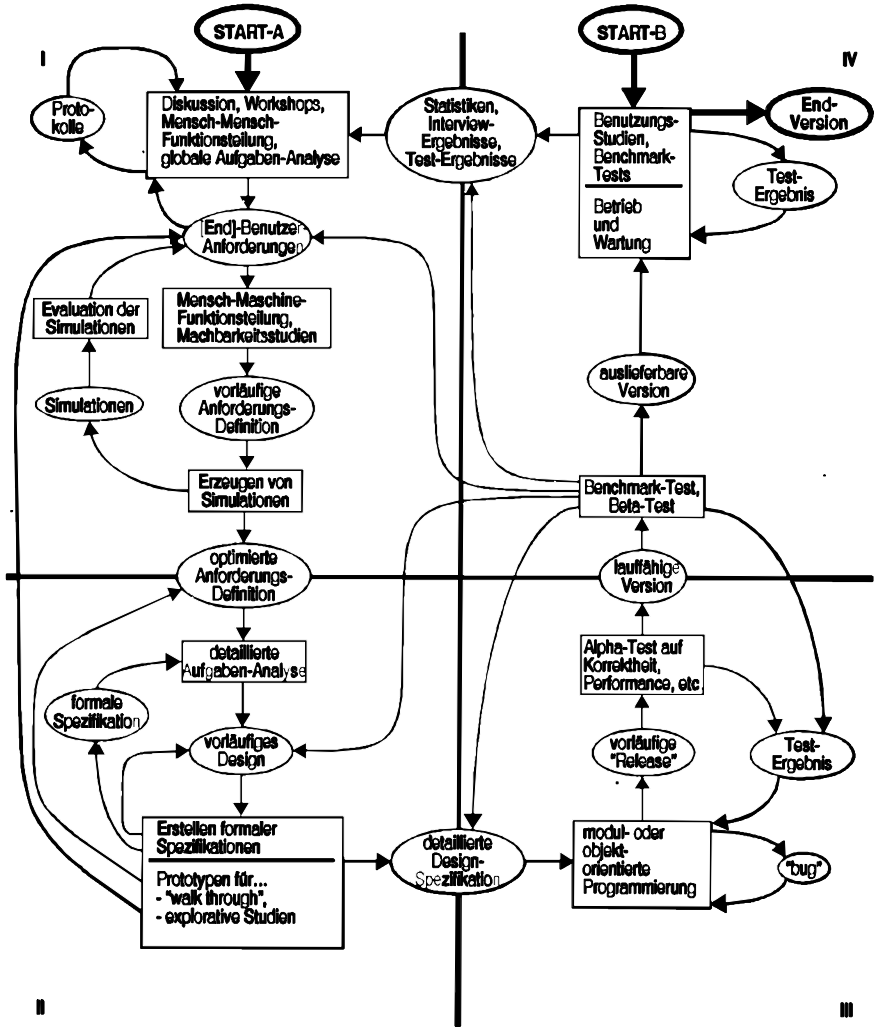

Abb. 2:Ablaufmodell eines partizipativen Softwareentwicklungskonzeptes mit einer Übersicht über die verschiedenen lokalen Optimierungszyklen innerhalb und zwischen den einzelnen Quadranten. Die systematische Einbeziehung der Anwendungs- und Wartungsphase mit Rückkopplung zur Planungsphase beinhaltet somit das Versionen-Konzept (in Anlehnung an Grudin / Ehrlich / Shriner 1987, sowie Boehm 1988)

kontextes (Grudin / Ehrlich / Shriner 1987; Macaulay et al. 1990). Bei neu zu entwickelnden System lassen sich sinnvoll Simulationsmethoden (z.B. Szenarien, "wizard of oz" Studien) einsetzen, welche keine spezielle Hard- / Software erfordern. Einen Überblick über Techniken zur Analyse und Evaluation interaktiver Computersysteme geben Spencer (1985) (siehe auch: Crellin / Horn / Preece 1990; Vainio-Larsson / Orring 1990). Beim Versionen-Konzept, bei dem nach zweimaligem Durchlauf mindestens zwei verschiedene Systemversionen vorliegen, lassen sich vergleichende Studien durchführen: z.B. benutzungsorientierte Benchmark-Tests (Lewis / Henry / Mack 1990; Rauterberg 1991).

4.3 Die vier Optimierungsquadranten

Der globale Optimierungszyklus mit seinen eingebetteten lokalen Zyklen läßt sich in vier Bereiche unterteilen (Quadrant I bis IV, siehe Abb. 2). Quadrant-I beinhaltet die Analyse und Grobspezifikation. Hierbei werden überwiegend kommunikative, informale Methoden verwendet. Im Quadrant-II wird die Detailspezifikation unter Einsatz von Prototypen optimiert. Im Quadrant-III wird die spezifizierte Hard / Software implementiert und an Testdaten getestet. Der Quadrant-IV umfaßt die Testung und Optimierung des entwickelten Systems in der realen Einsatzumgebung.

Je nach Projekt-Typ und -Auftrag wird der Optimierungsaufwand in den einzelnen Quadranten unterschiedlich ausfallen. Alle bisher durchgeführten Analysen von Software-Entwicklungsprojekten zeigen jedoch, daß mit zunehmendem Optimierungsaufwand in Quadrant-I der Wartungsaufwand in Quadrant-IV abnimmt und Kosten gespart werden (Macaulay et al. 1990). Das traditionelle "Wasserfall"-Modell ist als Spezialfall in dem hier vorgestellten Modell enthalten: wenig Aufwand in Quadrant-I, viel formaler Aufwand in Quadrant-II, entsprechender Implementationsaufwand in Quadrant-III und viel Wartungsaufwand in Quadrant-IV.

5. Methoden partizipativer Softwareentwicklung

Wie wir eingangs auf dem Hintergrund systemtheoretischer Betrachtungen aufgezeigt haben, setzt sich jeder einzelne Optimierungszyklus aus einer Test- und einer Aktivitätskomponente mit entsprechender Verkopplung (Abb. 1) zusammen. Beide Komponenten können recht unterschiedlicher Art sein. In der Tabelle 1 geben wir eine Übersicht über den Einsatzschwerpunkt, Art der Aktivität und des Tests, das jeweilige Arbeitsergebnis, sowie den geschätzten Bereich der Zyklus-Dauer. Je kürzer die Zyklus-Dauer ausfällt, desto schneller und damit öfter kann

Methode	Aktivität	Test	Ergebnis	Zyklus-Dauer
Diskussion-I	verbale Kommunikation	rein verbale Interpretation	globale Design-Entscheidungen	Sekunden - Minuten
Diskussion-II	Meta-Plan, Flip-Charts, etc.	visuelle + verbale Interpretation	spezifische Design-Entscheidungen	Minuten - Stunden
Simulation-I	Handskizzen, Szenarien, "wizard of oz", etc.	visuelle + verbale Interpretation	Spezifikation der Ein- / Ausgabe-Schnittstelle	Minuten - Tage
Simulation-II	Erstellung von Ablaufplänen, etc. mittels (semi)-formaler Methoden	visuelle + verbale Interpretation bei entsprechender Qualifikation	(semi)-formale Beschreibungsdokumente	Stunden - Wochen
Prototyping-I	horizontales Prototyping	"lautes Denken", "walk-through"	Spezifikation der Dialog-Komponente	Tage - Wochen
Prototyping-II	partiell vertikales Prototyping	heuristische Evaluation	Spezifikation von Teilen der Anwendungskomponente	Tage - Wochen
Prototyping-III	vollständig vertikales Prototyping	aufgabenorientierte Benchmark-Tests	Spezifikation der Anwendungskomponente	Wochen - Monate
Versionen-I	Durchlauf des gesamten Entwicklungszyklus	induktive Benchmark-Tests	erste weitgehend vollständige Version	Monate - Jahre
Versionen-II	Durchlauf des gesamten Entwicklungszyklus	deduktive Benchmark-Tests	mehrere weitgehend vollständige Versionen	Monate - Jahre

Tab. 1: Übersicht über die verschiedenen Methoden zur Benutzer-Partizipation

dieser Optimierungszyklus durchlaufen werden, um zu einem optimalen Ergebnis zu gelangen.

Ein wesentliches Problem der adäquaten Verzahnung der verschiedenen Optimierungszyklen besteht in ihrer *Synchronisation*. Sind an verschiedenen Stellen im partizipativen Softwareentwicklungskonzept (Abb. 2) gleichzeitig mehrere Optimierungszyklen aktiv, so müssen diese adäquat synchronisiert werden. Dieser Aspekt ist deshalb besonders wichtig, weil nur so das Ausmaß an Inkonsistenzen innerhalb des gesamten Entwicklungsprozesses minimiert werden kann. Wenn z.B. parallel zur Implementationsphase weitere Rücksprachen und Anforderungsanalysen mit dem Anwender stattfinden, passiert es leicht, daß die Entwickler gemäß der stets veralteten Spezifikationen oftmals für den "Papierkorb" programmieren. Die Ursache hierfür ist bei fehlender oder mangelnder Synchronisation dieser beiden Optimierungszyklen in ihrer unterschiedlichen Zyklus-Dauer zu sehen.

Unter dem funktionalen *Synchronisationsprinzip* verstehen wir, die Festlegung der zeitlichen Abfolge der einzelnen Optimierungs-"Quadranten" im Sinne einer funktionalen Phasen-Einteilung. Dieses Prinzip wird primär im "Wasserfall"-Konzept angewendet (das "Meilenstein"-Konzept) und führt notwendigerweise bei einer Übertragung - ausschließlich dieses Synchronisationsprinzips - auf das Versionen-Konzept zu einer drastischen Erhöhung der gesamten Zyklus-Dauer.

Um diesen Nachteil zumindest teilweise zu vermeiden, kann das *informationelle Synchronisationsprinzip* zum Einsatz kommen. Hierbei wird eine entsprechende informationelle Kopplung zwischen den einzelnen Optimierungszyklen aufgebaut, sodaß alle Personen in den unterschiedlichen Zyklen über den aktuellen Stand in den parallel aktiven Zyklen informiert sind (Mambrey / Oppermann / Tepper 1986:79ff). Dies kann durch einfache Hilfsmittel wie Aktenordner an einem definierten Standort, durch regelmäßige Besprechungstermine, aber auch mittels technischer Unterstützung (mail-box, Versionen-Datenbanken, "information repository", etc.) realisiert werden (Jones 1987:193). Hier sind auch und insbesondere die neueren Ergebnisse aus dem Forschungsbereich zur "computer-supported-cooperative-work" (CSCW) bezogen auf den Arbeitskontext der Entwickler anwendbar.

Ein weiteres, wesentliches Synchronisationsprinzip besteht darin, daß der Personenkreis, welcher an den unterschiedlichen Optimierungszyklen teilnimmt, *derselbe* ist. Dieses Prinzip bricht sich jedoch oftmals an der arbeitsteilig orientierten Organisationsform, welche bei vielen Softwarehäusern vorhandenen ist. Hier müßte eine Re-Organisation dieser Softwareentwicklungsabteilungen gemäß dem arbeitspsychologischen Kriterium der "Ganzheitlichkeit" von Arbeitsaufgaben vorgenommen werden (Ulich 1991). Da eine Person also nicht zur gleichen Zeit an räumlich zwei verschiedenen Orten sein kann, werden wir dieses Prinzip das *personelle Synchronisationsprinzip* nennen.

5.1 Stärken und Schwächen partizipativer Methoden

Im folgenden seien noch einige, über ihren Anteil an der Zyklus-Dauer hinausgehende, zu beachtende Aspekte beim Einsatz der verschiedenen partizipativen Methoden aufgezählt.

(1) Diskussionsmethoden I, II

Die Diskussionsmethode ist die am meisten eingesetzte Methode, weil sie schnell[3], geläufig und bis zu einem gewissen Grade (siehe die "Kommunikationsbarriere") informativ ist. Da sie jedoch im wesentlichen auf rein verbaler Kommunikation beruht, kommt es zu einer Reihe von Mißverständissen, welche oftmals nicht, bzw. zu spät entdeckt werden. Es ist daher unumgänglich, die Diskussionsmethode um Methoden mit visuellen Kommunikationstechniken zu ergänzen.

(2) Simulationsmethoden I, II

Unter den Simulationsmethoden sollen alle Techniken zusammengefaßt werden, welche das zu optimierende Arbeitssystem möglichst realistisch, visuell wahrnehmbar abbilden. Dies kann von einfachen, schnell entworfenen Handskizzen über ein Masken-Layout bis hin zu formalisierten Beschreibungstechniken gehen (SADT: Ross 1977; RFA-Netze: Oberquelle 1987; SA / SD: Yourdon 1989; siehe auch Boose 1989).

Der eindeutige Vorteil der formalen Analyse- und Beschreibungsmittel liegt darin, daß man durch die Verwendung dieser Hilfsmittel zu einer detaillierten Durchdringung des zu beschreibenden Gegenstandsbereiches angehalten wird. Je nach Verfahren liegen die Analyseschwerpunkte auf unterschiedlichen Aspekten. Der konkrete Arbeitskontext der End-Benutzer bleibt bei den meisten Beschreibungstechniken jedoch fast vollständig unberücksichtigt. Einschränkend kommt hinzu: "Je detaillierter diese Spezifikation ist, desto unverständlicher wird sie (nicht nur für DV-Laien)" (Budde et al. 1986:201). Je formaler die Darstellungsmethode ist, desto mehr Zeit wird auch zu ihrer Erstellung benötigt. Dies ist unter anderem dadurch bedingt daß bei Benutzer-Beteiligung diese erst eine entsprechende Qualifizierung in der Handhabung und Interpretation durchlaufen müssen. Dieser Umstand wird oftmals leider zum Anlaß genommen, bei Einsatz formaler Darstellungsmittel von einer Benutzer-Beteiligung Abstand zu nehmen.

[3] im Sinne der Zyklus-Dauer; ein Optimierungszyklus bei der Diskussionsmethode ist begrenzt auf die kommunikative Einheit von "Rede-Gegenrede", "Frage-Antwort", etc.

(3) Prototyping-Methoden I, II, III

Wie schon erwähnt, bieten die Prototyping-Methoden die Möglichkeit, den prozessualen Charakter des zu entwickelnden Systems den End-Benutzern nahe zu bringen (Melzer 1989). "Beim Prototyping handelt es sich darum, Teile oder die Gesamtheit eines Anwendungssystems in einem Arbeitsmodell soweit abzubilden, daß für den künftigen Benutzer die Arbeitsweise des geplanten Systems erfaßbar wird" (Rüesch 1989:49). In diesem Sinne dient Prototyping als besonders wirksames Mittel der Kommunikation zwischen dem Anwender und dem Entwickler. "Die wirtschaftliche Möglichkeit, Prototyping in der Anwendungsentwicklung einzusetzen, erfordert allerdings drei Hilfsmittel:

- eine nichtprozedurale Sprache,
- ein Datenbankverwaltungssystem und
- ein Data Dictionary" (Rüesch 1989:49).

Da Prototypen stets im Kontext der Test-Komponente eines entsprechenden Optimierungszyklus eingesetzt werden, müssen sie leicht änderbar sein. "So soll der zeitliche Abstand zwischen einem Änderungsvorschlag des Anwenders und der Überprüfung des geänderten Prototyps durch ihn möglichst gering sein, da sonst Motivationsprobleme auftreten" (Kreplin 1985:75).

Es lassen sich zwei Arten von Prototypen unterscheiden: vertikaler Prototyp und horizontaler Prototyp. Beim *horizontalen Prototypen*, welcher nur eine sehr geringe Anzahl von anwendungsbezogenen Funktionen des Endproduktes enthält, liegt der Gestaltungsschwerpunkt primär auf der Darbietung der Maskenabfolge eingebettet in eine Dialogstruktur. Der *vertikale Prototyp* demgegenüber geht verstärkt in die Tiefe; beim partiellen vertikalen Prototypen werden nur einige Anwendungsfunktionen in einer eher rudimentären Weise implementiert, während beim vollständigen vertikalen Prototypen nahezu alle Anwendungsfunktionen ansatzweise implementiert sind. Diese letzte Vorgehensweise entspricht am ehesten dem traditionellen Bild des Prototypen (siehe auch Pommberger et al. 1987:20-21). Pomberger et al. (1987) sprechen daher auch bei einem vollständigen vertikalen Prototypen von einem *Pilotsystem*. Mit zunehmender Vollständigkeit des vertikalen Prototypen verwischen die Grenzen zu einer ersten *Version*.

Die Nachteile der Prototyping-Methode liegen darin, daß sich die Voraussetzung, der Entwickler möge unvollständige Software produzieren ("rapid prototyping") und dann sich damit der Kritik des Anwenders aussetzen, nur schwer - wenn überhaupt - erfüllen läßt (Weinberg 1971). Ein anderer Aspekt ist, daß die traditionelle Semantik des Begriffs "Prototyp" im industriellen Bereich ein voll funktionsfähiges Produkt umfaßt. Dieser Begriff des Prototypen wäre aber im Bereich der Softwareentwicklung bereits das End-Produkt und eben nicht eine vorläufige Variante. "The sad truth is that as an industry, data processing routinely delivers a

prototyp under the guise of a finished product" (Boar 1984). Beide Aspekte können beim Einsatz der Prototyping-Methode dazu führen, daß "the best prototype is often a failed project" (Curtis et al. in: Grudin et al. 1987). "The fundamental idea of prototypes is to iterate the design, not to FREEZE it" (Jörgensen 1984:287). Um diese Gefahr zu vermeiden, legen einige Autoren verstärkt Wert auf einfachere und schnellere Techniken zur Partizipation (Grudin / Ehrlich / Shriner 1987; Nielson 1989, 1990).

Die Prototyping-Methode im Rahmen des entsprechenden Optimierungszyklus kann Gefahr laufen, ein inadäquates Optimum anzustreben. Dies kann dadurch zustande kommen, daß der konkrete Prototyp den Blick für grundsätzlich andere Alternativen verstellt (Floyd 1984:15). Dieses Problem läßt sich dadurch entschärfen, daß entsprechende, primär auf den Anwendungskontext ausgerichtete Optimierungszyklen vor- und übergelagert sind (siehe Abb. 2). Dennoch gibt es keine Garantie dafür, daß der Benutzer auch ein guter Software-Designer ist. Dies kann zur Folge haben, daß lediglich sub-optimale Lösungen iterativ erarbeitet werden (Jörgensen 1984:287). Hier kann gegebenenfalls der Einsatz von Standards und Kriterien zur Gestaltung interaktiver Software Abhilfe schaffen (Smith / Mosier 1986; Ulich et al. 1991).

(4) Versionen-Methoden I, II

Wenn die Methode des vertikalen Prototyping ausgebaut wird in Richtung auf zunehmende Anreicherung des Prototypen mit ausprogrammierter Funktionalität, so geht die Prototyping-Methode in die Versionen-Methode fließend über. Da dieses Vorgehen offenbar am ehesten noch mit dem "Wasserfall"-Modell im Rahmen eines Software-Lebenszyklus verträglich ist, hat es in den 80iger Jahren zunehmend an Bedeutung gewonnen. Der Versionszyklus als globaler Optimierungszyklus findet seinen Niederschlag in dem hier vorgeschlagenen partizipativen Entwicklungskonzept durch die Rückkopplung zwischen Anwendungsphase und Anforderungsermittlungsphase (Quadrant-IV & Quadrant-I; Abb. 2). Ein entsprechend prozeß-orientiertes Vorgehen ist bei Floyd und Keil (1983) beschrieben. Manche Autoren sprechen in diesem Zusammenhang auch von "evolutionärer" Softwareentwicklung (Lehman / Belady 1985).

Der wesentliche Vorteil des globalen Optimierungszyklus liegt eindeutig darin, daß erstmals hierbei alle Wechselwirkungen der Brauchbarkeit und Benutzbarkeit der jeweiligen Version im Rahmen der konkreten Arbeitsumgebung erfaßt und getestet werden können. Je nach Komplexität des zu entwickelnden Systems lassen sich bestimmte Design-Fehler grundsätzlich erst in der realen Anwendungsphase entdecken. Um den dann notwendigen Änderungsaufwand so klein wie möglich zu halten, muß das System von vornherein nach modernen Programmierkonzepten entwickelt werden (Dokumentation; modularer Aufbau; objekt-orientierte Programmierung; etc.).

Da man jedoch bei mehrmaligem Durchlauf durch den globalen Optimierungszyklus um ein gewisses Maß an Aufwärtskompatibilität nicht umhin kommt, müssen notwendigerweise vorgelagerte Optimierungszyklen in den Quadranten-I & II eingeplant werden, um das Risiko von grundlegenden Design-Fehlern zu minimieren (Peschke 1986:161). Peschke weist ebenfalls zu recht darauf hin, daß durch die recht große Zyklus-Dauer der Versionen-Methode der Kontakt zwischen Entwickler und Benutzer abbrechen kann (Peschke 1986:161).

5.2 Die Benutzer-Entwickler-Distanz bei partizipativen Methoden

Um Benutzer im Rahmen partizipativer Softwareentwicklung adäquat einbeziehen zu können, müssen eine Reihe von Aspekten berücksichtigt werden. Die im folgenden diskutierten Aspekte lassen sich zu dem sechs-dimensionalen Konzept der "Benutzer-Entwickler-Distanz" zusammenfassen. Je nach Projekttyp ergibt sich eine unterschiedliche Benutzer-Entwickler-Distanz, sodaß in Abhängigkeit davon unterschiedliche Methoden zur Partizipation zum Tragen kommen müssen.

Die anwendungsbezogene Distanz: Hiermit ist gemeint, in wie weit der konkrete Anwendungskontext des zu entwickelnden Systems bekannt ist. So werden z.B. Benutzer aus Fachabteilungen zur Partizipation "abgestellt", welche entweder in der Fachabteilung auf Grund ihrer unzureichenden fachlichen Qualifikation entbehrlich sind, oder sich sogar erst während der "Partizipationszeit" die nötigen fachlichen Qualifikationen aneignen sollen. Ein schwerwiegenderes Problem kommt jedoch dadurch zustande, daß der Anwendungskontext noch weitgehend unbekannt ist, bzw. sehr heterogen sein kann; dieser Umstand trifft insbesondere bei weitgehenden Neuentwicklungen, bzw. der Entwicklung von Standardsoftware zu.

Die qualifikatorische Distanz: Werden Analyse- und Beschreibungstechniken eingesetzt, deren Anwendung eine spezielle Ausbildung erfordert, so müssen alle im Umgang mit diesen Techniken Betroffene entsprechend qualifiziert werden. Für eine fruchtbare Kommunikation zwischen Benutzer und Entwicklern müssen sich ebenfalls die Entwickler soweit in die Arbeitstätigkeit der Benutzer einarbeiten, daß sie zumindest die richtigen Fragen stellen können. Werden Benutzer unzureichend beteiligt, ist der Entwickler vor die Aufgabe gestellt, sich selbst zum Fachexperten auszubilden. Diese Aufgabe kann aber im Rahmen eines Entwicklungsprojektes nur suboptimal bewältigt werden.

Die organisationale Distanz: Wenn die Entwicklungsabteilung und die Fachabteilung zu einer gemeinsamen Organisation zugeordnet sind, dann lassen sich oftmals freiere vertragliche Abkommen treffen, als wenn Entwicklungsabteilung und Fachabteilung vollständig unterschiedlichen Organisationen angehören. Je größer

die organisationale Distanz ist, desto umfangreicher sind meistens die vertraglichen Vereinbarungen für die rechtliche Absicherung des Projektes.

Die motivationale Distanz: Wenn Benutzer nur unzureichend über den aktuellen Entwicklungsstand informiert werden, sie keine unmittelbaren Einflußmöglichkeiten haben oder ihnen dazu die notwendige Qualifikation fehlt, kann es zu ernsten motivationalen Störungen kommen, welche die weitere Zusammenarbeit behindern. Insbesondere kann eine große motivationale Distanz zustande kommen, wenn die Benutzer nicht den Eindruck haben, daß ihre konkrete Arbeit erleichtert werden soll, sondern sie das Opfer von Rationalisierungsmaßnahmen werden könnten.

Die räumliche Distanz: Dieser Aspekt kommt immer dann besonders zum Tragen, wenn die Arbeitsstätte der Benutzer räumlich deutlich von dem Ort der Softwareentwicklungsabteilung getrennt ist. So kann es sich als besonders nützlich erweisen, wenn möglichst große Teile der Entwicklung am Ort der Arbeitsstätte durchgeführt werden können, um somit eine möglichst schnelle Rückkopplung zwischen Benutzern und Entwicklern zuzulassen.

Die zeitliche Distanz: Hierunter fallen alle die Probleme, welche dadurch bedingt sind, daß die Arbeitszeit der Benutzer kostbar ist. Selbst bei unmittelbarer räumlicher Nähe kann eine intensive Rückkopplung mit Benutzern dadurch erschwert werden, daß diese kaum Zeit finden, sich an intensiven Anforderungsermittlungen zu beteiligen. Dieses Problem taucht oftmals bei Projekten auf, welche die Arbeit von fachlich hoch qualifizierten Benutzern EDV-technisch unterstützen sollen.

6. Fazit

Eines der Hauptprobleme traditioneller Software-Entwicklung liegt darin, daß alle bisher primär an Software-Entwicklungen beteiligten Personengruppen nicht wahrhaben wollen, daß Software-Entwicklung in den meisten Fällen primär Arbeits- und Organisationsgestaltung ist. Um mit dieser Perspektive an Software-Entwicklung heranzugehen, hieße, von vornherein Experten für Arbeits- und Organisationsgestaltungsmaßnahmen mit in den Prozeß der Software-Entwicklung einzuplanen. Dies erfordert jedoch notwendigerweise eine interdisziplinäre Zusammenarbeit zwischen Arbeits- und Organisationsexperten einerseits und Software-Entwicklungsexperten andererseits (Waeber 1991). Wegen der umfangreichen Qualifikation in dem jeweiligen Fachgebiet ist nur sehr begrenzt möglich, auf eine interdisziplinäre Zusammenarbeit zu verzichten.

Es wird ein Software-Entwicklungskonzept vorgestellt, welches die verschiedenen, bisher entwickelten Ansätze zur Überwindung der Spezifikations-, Kommunikations- und Optimierungsbarriere integriert. Die Grundlage bildet der Optimierungszyklus, welcher aus einer Test- und einer Aktivitätskomponente besteht; bei-

de Komponenten sind miteinander rückgekoppelt. Die an verschiedenen Stellen in der Literatur vorgeschlagenen Rückkopplungsschleifen werden als lokale Optimierungszyklen in einen globalen Optimierungszyklus eingebettet. Der globale Optimierungszyklus läßt sich in vier Bereiche unterteilen: der Bereich der Anforderungsermittlung (Quadrant-I), der Bereich der Spezifikation (Quadrant-II), der Bereich der Implementation (Quadrant-III) und der Bereich der Benutzung (Quadrant-IV).

Von Quadrant zu Quadrant fortschreitend lassen sich unterschiedliche Aspekte des zu gestaltenden Arbeitssystems optimieren. Stufenweise werden alle Sichten auf das Soll-Konzept zunehmend konkreter, wobei die Einbeziehung aller Beteiligten und Betroffenen als Experten ihrer Arbeit und als Sachverwalter ihrer Interessen unabdingbar ist. Es hat sich gezeigt, daß ein entsprechender Optimierungsaufwand im Quandrant-I und -II nicht nur die Gesamtkosten (Entwicklungskosten plus Nutzungskosten) reduzieren hilft, sondern auch eine an das Arbeitssystem optimal angepaßten Hard- / Software-Lösung ermöglicht. Dies wird darauf zurückgeführt, daß alle an der späteren Nutzung beteiligten Personengruppen repräsentativ beteiligt wurden, und somit das relevante Wissen in die Gestaltung des Arbeitssystems einfließen konnte.

Mit zunehmendem Optimierungsaufwand im ersten Quadranten verringert sich der Aufwand im vierten Quadranten. Der Optimierungsaufwand im zweiten und dritten Quadranten hängt im wesentlichen von der Komplexität des zu gestaltenden Arbeitssystems ab. Eine Aufwandsminimierung im zweiten Quadranten läßt sich z.B. durch die Verwendung von modernen Prototypingwerkzeugen und dem Anwender verständlichen Spezifikationsmethoden erreichen. Im dritten Quadranten wird eine Aufwandsminimierung durch den Einsatz mächtiger Entwicklungsumgebungen und durch eine entsprechende Qualifizierung der Softwareentwickler erreicht.

Am wichtigsten ist jedoch, daß wir anfangen zu lernen, Technik, Organisation und den Einsatz menschlicher Qualifikation gemeinsam zu planen. Betrachten wir also die Technik als eine Option, welche es uns gestattet, unsere Lebens- und Arbeitsräume menschengerecht und lebenswert zu gestalten.

7. Literaturverzeichnis

Baitsch, C. / Katz, C. / Spinas, P. / Ulich E., 1989: Computerunterstützte Büroarbeit - Ein Leitfaden für Organisation und Gestaltung. Zürich: Verlag der Fachvereine

Baroudi, J. / Olson, M. / Ives, B., 1986: An Empirical Study of the Impact of User Involvement on System Usage and Information Satisfaction. Communications of the ACM 29(3):232-238

Beck, A. / Ziegler, J., 1991: Methoden und Werkzeuge für die frühen Phasen der Software-Entwicklung. In: Ackermann, D. / Ulich, E. (Hg.) Software-Ergonomie '91. (Berichte des German Chapter of the ACM, Vol. 33). Stuttgart: Teubner. 76-85

Bewley, W. L. / Roberts, T. L. / Schroit, D. / Verplank, W. L., 1983: Human factors testing in the design of XEROXs' 8010 Star office workstation. Proceedings of CHI '83, Human Factors in Computing Systems (Boston, Mass., 12th - 15th December), New York: ACM. 72-77

Boar, B. H., 1984: Application Prototyping: A Requirements Definition Strategy for the 80s'. New York: John Wiley.

Boehm, B. W., 1981: Software Engineering Economics. Englewood Cliffs: Prentice Hall.

Boehm, B. W., 1988: A spiral model of software development and enhancement. Computer (May 1988): 61-72

Boehm, B. W. / Gray, T. / Seewaldt, T., 1981: Prototyping versus specifying: a multiproject experiment. IEEE Transactions on Software Engineering, SE-10(3): 224-236

Boose, J. H., 1989: A survey of knowledge acquisition techniques and tools. Knowledge Acquisition 1: 3-37

Budde, R. / Kuhlenkamp, K. / Sylla, K.-H. / Züllighoven, H., 1986: Prototypenbau bei der Systemkonstruktion - Konzepte der Systementwicklung. Angewandte Informatik 5: 198-204

Buhr, M. / Klaus, G., 1975: Philosophisches Wörterbuch. Berlin: Das Europäische Buch

Carroll, J., 1988: Integrating Human Factors and Software Development. In: Proceedings of CHI '88 (Washington, 15th - 19th May 1988). New York: ACM. 157-159

C.E.C. Commission of the European Communities, 1989: Science, Technology and Societies: European Priorities. Results and Recommendations of the FAST II Programme. Summary Report. Directorate-General Science, Research and Development, Brussels

Chikofsky, E. J. / Rubenstein, B. L., 1988: CASE: Reliability engineering for information systems. IEEE Software 5(2): 11-17

Crellin, J. / Horn, T. / Preece, J., 1990: Evaluating Evaluation: A Case Study of the Use of Novel and Conventional Evaluation Techniques in a Small Company. In: Diaper D. et al. (Hg.) Human-Computer Interaction - INTERACT '90. Amsterdam: Elsevier Science. 329-335

Cummings, T. / Blumberg, M., 1987: Advanced manufacturing technology and work design. in: Wall, T. / Clegg, C. / Kemp, N. (Hg.) The Human Side of Advanced Manufacturing Technology. Chichester: Wiley. 37-60

Debusmann, E. / 1984: Das VAB-Verfahren zur Analyse und Gestaltung von Bürotätigkeiten. (Europäische Hochschulschriften, Reihe V, Vol. 569).Frankfurt /Bern/New York: Lang

Dzida, W., 1987: On tools and interfaces. In: Frese, M. / Ulich, E. / Dzida, W. (Hg.) Psychological Issues of Human Computer Interaction in the Work Place. Amsterdam: North-Holland. 339-355

Eason, K. D. / Harker, S. D. P., 1987: A User Centered Aapproach to the Design of a Knwoledge Based System. In: Bullinger, H.-J. / Shackel, B. (Hg.) Human-Computer Interaction - INTERACT '87. Amsterdam: Elsevier Science. 341-346

Floyd, C., 1984: A Systematic Look at Prototyping. In: Budde, R. / Kuhlenkamp, K. / Mathiassen, L. / Züllighoven, H. (Hg.) Approaches to Prototyping. Berlin: Springer. 1-15

Floyd, C. / Keil, R., 1983: Adapting Software Development for Systems Design with User. In: Briefs, U. / Ciborra, C. / Schneider, L. (Hg.) System Design for, with, and by the Users. Amsterdam: North-Holland. 163-172

Foidl, H. / Hillebrand, K. / Tavolato, P., 1986: Prototyping: die Methode - das Werkzeug - die Erfahrungen. Angewandte Informatik 3:95-100

Gomaa, H., 1983: The impact of rapid prototyping on specifying user requirements. ACM SIGSOFT Software Engineering Notes 8(2): 17-28

Grudin, J. / Ehrlich, S. F. / Shriner, R., 1987: Positioning Human Factors in the User Interface Development Chain. In: Proceedings of CHI + GI (Toronto, 5th - 9th April 1987). New York: ACM. 125-131

Hacker, W. / Iwanowa, A. / Richter, P., 1983: Tätigkeits-Bewertungs-System. (Schriftenreihe zur Arbeitspsychologie, Vol. 20; Hg.: E Ulich). Bern: Huber

Hacker, W. / Matern, B., 1980: Methoden zum Ermitteln tätigkeitsregulierender kognitiver Prozesse und Repräsentationen bei industriellen Arbeitstätigkeiten. In: Volpert, W. (Hg.) Beiträge zur psychologischen Handlungstheorie. (Schriften zur Arbeitspsychologie, Vol. 28; Hg.: E. Ulich). Bern: Huber. 29-49

Hacker, W. / Müller-Rudolph, E. / Schwarzer- Schönfelder, E., 1989: Hilfsmittel für die kooperative Aufgabenanalyse - eine Voraussetzung aufgabenorientierter Systemgestaltung. In: Maass, S. / Oberquelle, H. (Hg.) Software-Ergonomie '89. (Berichte des German Chapter of the ACM, Volume 32). Stuttgart: Teubner. 89-99

Hackman, J. R. / Oldham, G. R., 1975: Development of the Job Diagnostic Survey. Journal of Applied Psychology 60: 159-170

Helander, M., 1988 (Hg.): Handbook of Human-Computer Interaction. Amsterdam: Elsevier Science

Hix, D. / Schulman, R. S., 1991: Human-Computer Interface Development Tools: a methodology for their evaluation. Communications of the ACM 34(3): 75-87

Iivari, J., 1984: Prototyping in the Context of Information System Design. In: Budde, R. / Kuhlenlamp, K. / Mathiassen, L. / Züllighoven, H. (Hg.) Approaches to Prototyping. Berlin: Springer

Jörgensen, A. H., 1984: On the Psychology of Prototyping. In: Budde, R. / Kuhlenkamp, K. / Mathiassen, L. / Züllighoven, H. (Hg.) Approaches to Prototyping. Berlin: Springer. 278-289

Jones, T. C., 1987: Effektive Programmentwicklung. Hamburg / New York: McGraw Hill

Karat, C.-M., 1990: Cost-Benefit Analysis of Iterative Usability Testing. In: Diaper, D. et al. (Hg.) Human-Computer Interaction - INTERACT '90. Amsterdam: Elsevier Science. 351-356

Keil-Slawik, R., 1989: Systemgestaltung mit Aufgabennetzen. In: Maass, S. / Oberquelle, H. (Hg.) Software-Ergonomie '89. (Berichte des German Chapter of the ACM, Vol. 32). Stuttgart: Teubner. 123-133

Kieback, A. / Lichter, H. / Schneider-Hufschmidt, M. / Züllighoven, H., 1991: Prototyping in industriellen Software-Projekten - Erfahrungen und Analysen. GMD-Report No. 184, Gesellschaft für Mathematik und Datenverarbeitung, P.O. 1240, DW-5205 Sankt Augustin 1

Klotz, U., 1991: Die zweite Ära der Informationstechnik. Harvard Manager 13(2): 101-112

Kraut, R. E. / Streeter, L. A., 1991: Coordination in Large Scale Software Development. Communications of the ACM:(in press)

Kreplin, K.-D., 1985: Prototyping - Softwareentwicklung für den und mit dem Anwender. Handbuch der Modernen Datenverarbeitung 22(126): 73-126

Krüger, W., 1987: Problemangepasstes Management von Projekten. Zeitschrift Führung und Organisation 4: 207-216

Lehman, M. M. / Belady, L. A., 1985: Program Evolution - Processes of Software Change. London: Academic

Lewis, J. R. / Henry, S. C. / Mack, R. L., 1990: Integrated Office Software Benchmarks: A Case Study. In: Diaper, D. et al. (Hg.) Human-Computer Interaction - INTERACT '90. Amsterdam: Elsevier Science. 337-343

Macaulay, L. / Fowler, C. / Kirby, M. / Hutt, A., 1990: USTM: a new approach to requirements specification. Interacting with Computers 2(1): 92-118

Mai, M., 1990: Sprache und Technik. Zeitschrift des Vereins Deutscher Ingenieure für Maschinenbau und Metallbearbeitung 132(7): 10-13

Malone, T. W., 1985: Designing organizational interfaces. In: Borman, L. / Curtis, B. (Hg.) Proceedings of CHI '85 (San Francisco, Special Issue of the SIGCHI Bulletin). 66-71

Mambrey, P. / Oppermann, R. / Tepper, A., 1986: Computer und Partizipation. Opladen: Westdeutscher Verlag

Mantei, M. M. / Teorey, T. J., 1988: Cost / Benefit Anaylsis for Incorporating Human Factors in the Software Lifecycle. Communications of the ACM 31(4):428-439

Martin, C. F., 1988: User-Centered Requirements Analysis. Englewood Cliffs: Prentice Hall

Melzer, W., 1989: User Participation in Software Design - Problems and Recommendations. Psychological Task Analysis, Design and Training in Computerized Technologies. Technical Report No. 113. Helsinki University of Technology. Otakaari 4 A., SF-02150 Espoo. 109-119

Müller-Holz auf der Heide, B. / Aschersleben, G. / Hacker, S. / Bartsch, T., 1991: Methoden zur empirischen Bewertung der Benutzerfreundlichkeit von Bürosoftware im Rahmen von Prototyping. In: Frese, M. / Kasten, C. / Skarpelis, C. / Zang-Scheucher, B. (Hg.) Software für die Arbeit von morgen. Berlin / Heidelberg / New York: Springer. 409-420

Mumford, E. / Welter, G., 1984: Benutzerbeteiligung bei der Entwicklung von Computersystemen - Verfahren zur Steigerung der Akzeptanz und Effizienz des EDV-Einsatzes. Berlin: Schmidt

Nielson, J., 1989: Usability Engineering at a Discount. In: Salvendy, G. / Smith, M. J. (Hg.) Designing and Using Human-Computer Interfaces and Knowledge Based Systems. Amsterdam: Elsevier Science. 394-401

Nielson, J., 1990: Big paybacks from 'discount' usability engineering. IEEE Software 7(3):107-108

Oberquelle, H., 1987: Sprachkonzepte für benutzergerechte Systeme. (Informatik-Fachberichte 144). Berlin / Heidelberg / New York: Springer

Peschke, H., 1986: Betroffenenorientierte Systementwicklung. Europäische Hochschulschriften Reihe XLI Informatik Bd./Vol.1. Frankfurt / Bern / New York: Lang

Pomberger, G. / Bischofsberger, W. / Keller, R. / Schmidt, D., 1987: Prototypingorientierte Softwareentwicklung. Teil I. Technical Report No 87.05, Institut für Informatik, Winterthurerstrasse 190, CH-8057 Zürich

Prieto-Diaz, R., 1991: Implementing Faceted Classification for Software Reuse. Communications of the ACM 34(5): 88-97

Rauterberg, M., 1991: Benutzungsorientierte Benchmark-Tests: eine Methode zur Benutzerbeteiligung bei Standardsoftwareentwicklungen. In: Ackermann, D. / Ulich, E. (Hg.) Software-Ergonomie '91. (Berichte des German Chapter of the ACM, Vol. 33). Stuttgart: Teubner. 96-107

Rettig, M., 1991: Testing made palatable. Communications of the ACM 34(5):25-29

Rödiger, K.-H., 1987: Arbeitsorientierte Gestaltung von Dialogsystemen im Büro- und Verwaltungsbereich. unveröffentlichte Doktorarbeit. Fachbereich Informatik, Technische Universität Berlin, Franklinstrasse 28/29, DW-1000 Berlin 10

Ross, D. T., 1977: Structured Analysis (SA): a language for communicating ideas. IEEE Transactions on Software Engineering SE-3(1): 16-34

Rudolph, E. / Schönfelder, E. / Hacker, W., 1987: Tätigkeits-Bewertungs-System für Geistige Arbeit. Psychodiagnostisches Zentrum an der Humboldt-Universität, Oranienburger Strasse 18, DO-1020 Berlin

Rüesch, P., 1989: Entwicklungsumgebung. Output 11: 45-51

Schiemenz, B., 1979: Kybernetik. In: Kern, W. (Hg.) Handwörterbuch der Produktionswissenschaft. Stuttgart: Poeschel. 1022-1028

Schmidt, D. / Pomberger, G. / Bauknecht, K., 1989: Prototypingorientierte Softwareentwicklung. Teil II. Technical Report No 89.08, Institut für Informatik, Winterthurerstrasse 190, CH-8057 Zürich

Schönthaler, F. / Nemeth, T., 1990: Softwareentwicklungswerkzeuge: methodische Grundlagen. Stuttgart: Teubner

Smith, S. L. / Mosier, J. N., 1986: Guidelines for Designing User Interface Software. Technical Report ESD-TR-86-278, U.S.A.F. (NTIS No. AD-A177 198). Electronic Systems Division, Hanscom Air Force Base, Massachusetts U.S.A.

Sommerville, I., 1989: Software Engineering. 3rd edition. Wokingham: Addison-Wesley

Spencer, R. H., 1985: Computer usability testing and evaluation. Englewood Cliffs: Prentice Hall

Spinas, P. / Ackermann, D., 1989: Methods and Tools for Software Development: Results of Case Studies. In: Klix, F. / Streitz, N. / Waern, Y. / Wandke, H.

(Hg.) Man-Computer Interaction Research MACINTER-II. Amsterdam: Elsevier Science. 511-521

Spinas, P. / Waeber, D., 1991: Benutzerbeteiligung aus der Sicht von Endbenutzern, Softwareentwicklern und Führungskräften. In: Ackermann, D. / Ulich, E. (Hg.) Software-Ergonomie '91. (Berichte des German Chapter of the ACM, Vol 33). Stuttgart: Teubner. 36-45

Strohm, O., 1991a: Arbeitsorganisation, Methodik und Benutzerorientierung bei der Software-Entwicklung. In: Frese, M. / Kasten, C. / Skarpelis, C. / Zang-Scheucher, B. (Hg.) Software für die Arbeit von morgen. Berlin / Heidelberg / New York: Springer. 431-441

Strohm, O., 1991b: Projektmanagement bei der Software- Entwicklung. In: Ackermann D. / Ulich E. (Hg.) Software-Ergonomie '91. (Berichte des German Chapter of the ACM, Vol. 33). Stuttgart: Teubner. 46-58

Ulich, E., 1989: Arbeitspsychologische Konzepte der Aufgabengestaltung. In: Maass, S. / Oberquelle, H. (Hg.) Software-Ergonomie '89. (Berichte des German Chapter of the ACM, Vol. 32). Stuttgart: Teubner. 51-65

Ulich, E., 1991: Arbeitspsychologie. Stuttgart: Poeschel

Ulich, E. / Rauterberg, M. / Moll, T. / Greutmann, T. / Strohm, O., 1991: Task Orientation and User-Oriented Dialogue Design. International Journal of Human Computer Interaction 3(2): 117-144

Upmann, R., 1989: Aufgaben- und nutzerorientierte Gestaltung rechnergestützter, kooperativer Arbeitssysteme in den indirekten Produktionsbereichen mittelständischer Maschinenbauunternehmen. In: Maass, S. / Oberquelle, H. (Hg.) Software-Ergonomie '89. (Berichte des German Chapter of the ACM, Vol. 32). Stuttgart: Teubner. 110-122

Vainio-Larsson, A. / Orring, R., 1990: Evaluating the Usability of User Interfaces: Research in Practice. In: Diaper, D. (Hg.) Human-Computer Interaction - INTERACT'90. Amsterdam: Elsevier Science. 323-328

van der Schaaf, T., 1989: Redesigning and Evaluating VDU Graphics for Process Control: cognitive ergonomics applied to the operator interface. In: Salvendy, G. / Smith, M. J. (Hg.) Designing and Using Human-Computer Interfaces and Knowledge Based Systems. Amsterdam: Elsevier Science. 263-270

Volpert, W., 1987: Psychische Regulation von Arbeitstätigkeiten. In: Kleinbeck, U. / Rutenfranz, J. (Hg.) Arbeitspsychologie. (Enzyklopädie der Psychologie, Themenbereich D, Serie III, Vol. I). Göttingen: Hogrefe. 1-42

Vossen, P., 1991: Rechnerunterstützte Verhaltensprotokollierung und Protokollanalyse. In: Rauterberg, M. / Ulich, E. (Hg.) Posterband zur Software-Ergonomie '91. Institut für Arbeitspsychologie, Eidgenössische Technische Hochschule (ETH), Nelkenstrasse 11, CH-8092 Zürich. 181-188

Waeber, D., 1990: Entwicklung und Umsetzung von Modellen partizipativer Softwareentwicklung. In: Spinas, P. / Rauterberg, M. / Strohm, O. / Waeber, D. / Ulich, E. (Hg.) Projektberichte zum Forschungsprojekt BOSS - Benutzer-orientierte Softwareentwicklung und Schnittstellengestaltung Nr. 4. Institut für Arbeitspsychologie, Eidgenössische Technische Hochschule (ETH), Nelkenstrasse 11, CH-8092 Zürich

Waeber, D., 1991: Aufgabenanalyse und Anforderungsermittlung in der Softwareentwicklung: zur Konzeption einer integrierten Entwurfsstrategie. In: Frese, M. / Kasten, C. / Skarpelis, C. / Zang-Scheucher, B. (Hg.) Software für die Arbeit von morgen. (Ergänzungsband zum Tagungsband). Krefeld: Vennekel. 35-45

Weinberg, G. M., 1971: The psychology of computer programming. Van Nostrand, New York

Weltz, F. / Lullies, V. / Ortmann, R. G., 1991: Softwareentwicklung als Prozeß der Arbeitsstrukturierung. In: Ackermann, D. / Ulich, E. (Hg.) Software-Ergonomie '91. (Berichte des German Chapter of the ACM, Vol. 33). Stuttgart: Teubner. 70-75

Wirfs-Brock, R. J. / Johnson, R. E., 1990: Surveying current research in object-oriented design. Communications of the ACM 33(9):105-124

Wulff, W. / Evenson, S. / Rheinfrank, J., 1990: Animating Interfaces. In: Proceedings of the Conference on Computer-Supported Cooperative Work (Los Angeles, 7th - 10th October, 1990). 241-254

Yaverbaum, G. J. / Culpan, O., 1990: Exploring the Dynamics of the End-User Environment: The Impact of Education and Task Differences on Change. Human Relations 43(5):439-454

Yourdon, E., 1989: Modern Structured Analysis. Englewood Cliffs: Prentice-Hall.

Zölch, M. / Dunckel, H., 1991: Erste Ergebnisse des Einsatzes der "Kontrastiven Aufgabenanalyse". In: Ackermann, D. / Ulich, E. (Hg.) Software-Ergonomie '91. (Berichte des German Chapter of the ACM, Vol. 33). Stuttgart: Teubner. 363-372