Softwareentwicklung als verteiltes kooperatives Arbeiten

Michael Herczeg, Hansjürgen Paul

Einleitung

Nachdem mittlerweile die sogenannte Softwarekrise mehr als ein Vierteljahrhundert weite Teile der Wirtschaft weltweit nachhaltigst schädigt und so simple Ereignisse wie die Umstellung auf Sommer- bzw. Winterzeit oder das Jahr 2000 Anlaß für diverse Forschungsvorhaben und internationale Konferenzen werden, wird kaum jemand von diesem Beitrag ein Allheilmittel erwarten können. Dennoch aber soll der Anspruch erhoben werden, einige Probleme der Praxis der Softwareentwicklung aus einem unüblichen Blickwinkel zu betrachten und so zu neuen, angemesseneren Einschätzungen zu gelangen.

Wir wollen aufzeigen, daß einige der unangenehmen Probleme der Praxis ihre Ursache im defizitären Verständnis von Softwareentwicklung als verteiltes kooperatives Arbeiten haben, und wollen für diese Lösungswege vorschlagen. Daß diese Lösungen nicht zwangsläufig (software-) technischer Natur und oftmals so nicht zu lösen sind, soll in diesem Beitrag ebenfalls deutlich werden.

1. Die Situation

Trotz aller methodischen und technischen Fortschritte scheint sich die Entwicklung von Software einer organisatorischen und wirtschaftlichen Kontrolle mehr denn je zu entziehen. Sei es das Versagen des Abrechnungsmoduls eines Telefonsystems, das von Millionen von Teilnehmern benutzt wird oder das der Steuerung einer Trägerrakete, die milliardenschwere Satelliten in eine Umlaufbahn bringen soll, Softwarefehler stellen sich immer häufiger als die Ursache heraus – auch wenn der Öffentlichkeit oft andere Gründe genannt werden. In den seltensten Fällen sieht die Öffentlichkeit demzufolge auch in den Softwareherstellern die Schuldigen.

Die folgende Situtationsbeschreibung basiert auf Erfahrungen in realen Softwareprojekten. Derartige Situationen finden sich bei entsprechendem Einblick in praktisch allen größeren Softwareprojekten. Dies gilt auch für Institutionen, die erfolgreich anspruchsvollste Qualitätszertifizierungen – welcher Schule auch immer – nachzuweisen in der Lage sind. Beispiele dafür sind militärische Softwaresysteme oder Systeme der Luft- und Raumfahrttechnik, die nach den höchsten existierenden Qualitäts- und Prozeßstandards arbeiten.

Es soll im folgenden aufgezeigt werden, daß ein großer Teil der Probleme deutliche Spuren von

Kommunikationsdefiziten aufweist, die vor allem dann auftreten, wenn man sie sich am wenigsten leisten kann: in großen, komplexen und verteilten Projekten.

1.1 Größe und Komplexität

Während in den Kindertagen der Softwareentwicklung am Ende der 60er Jahre Softwareprojekte ein paar zehntausend Zeilen Programmcode von einigen Dutzend Programmierern zu erstellen hatten, müssen heute oft einige Millionen Zeilen von einigen tausend Softwareentwicklern hergestellt werden. Die Größe und Komplexität von Softwareprojekten hat einen Umfang erreicht, der sich in immer mehr Fällen weder als planbar noch als kontrollierbar erweist. Projekte ufern zeitlich und inhaltlich unkontrollierbar aus. Zeitverzug um Monate und Jahre ist der Normalfall. Nicht selten benötigt ein Projekt um den Faktor fünf bis zehn mehr an Aufwand, sprich eine Größenordnung mehr, als geplant. Daran hat sich seit den 80er Jahren (vgl. Brooks 1987) nichts verbessert.

Projekte werden nicht mehr fertiggestellt, sondern „inkrementell" oder „iterativ" in mehreren Releases geliefert und freigegeben. Die Betriebssysteme – egal, ob Personal Computer, Workstation oder Mainframe – liefern endlosen Stoff für diese zur Managementstrategie erhobene Misere. Dies gründet sich dabei nicht etwa auf einen von Kunden gewünschten Stufenplan, sondern auf eine notgedrungen schrittweise Entwicklung und Fehlerbeseitigung.

Trotz dieser alarmierenden Situation wachsen Softwareprojekte weiter und diffundieren in Anwendungsbereiche, die bislang noch einigermaßen durch klassische Ingenieurdisziplinen beherrscht wurden. Man denke nur an die zunehmende Computerisierung von Transportmitteln wie Autos, Bahnen und Flugzeugen. Wer glaubt, ein technisches System durch Software „sicherer" zu machen, handelt im Grunde genommen fahrlässig und verkennt die Tatsachen (vgl. z. B. Neumann 1995).

1.2 Verteilte Projekte

Mit der Größe von Softwareprojekten wächst offenbar die Notwendigkeit, die Software an verschiedenen Standorten zu entwickeln. Internationale Firmen unterhalten mehrere Entwicklungsabteilungen in verschiedenen Ländern, die gemeinsam an einem Softwareprojekt arbeiten sollen. Dies soll einerseits die Projektlaufzeit durch Parallelisierung verringern und andererseits den jeweiligen Anforderungen unterschiedlicher nationaler Märkte entgegenkommen. Nebenbei glaubt man, durch Outsourcing in Billiglohnländer auch noch Kosten sparen zu können (vgl. z. B. Cyranek et al. 1995).

Eine Verteilung von Projekten findet nicht nur geographisch statt. Selbst das Organisieren eines Projekts mit mehreren Dutzend Entwicklern an einem Standort führt zu einer Verteilung der Entwickler auf verschiedene Gebäude, Stockwerke und Räume. Es macht erfahrungsgemäß keinen großen Unterschied mehr, ob zwei Entwicklungsteam auf unterschiedlichen Stockwerken eines Gebäudes arbeiten, oder ob sie auf unterschiedlichen Kontinenten sitzen. Bereits bei 30 Metern beginnt die kritische Distanz (vgl. z. B. Springer et al. 1996).

Leider ist ein ständiges sich gegenseitig Besuchen und Abstimmen aus zeitlichen Gründen weder immer möglich, noch aus organisatorischen Gründen erwünscht. Trotz Zeitmangel und Kostendruck leisten sich manche Firmen den Luxus, Teams gegeneinander antreten zu lassen und Aufgaben mehrfach zu vergeben. Die vermeintlich beste, weil am schnellsten entwickelte, Lösung wird dann als einzige akzeptiert.

1.3 Zeitdruck

Softwareentwickler stehen immer unter Zeitdruck. Die Gründe dafür sind vielschichtiger Natur. Praktisch alle Softwareprojekte werden unter Annahmen und Randbedingungen gestartet, unter denen ein termingerechter Abschluß unmöglich ist und das Vorhaben im Grunde von vornherein zum Scheitern verurteilt ist. Der harte Markt erfordert fast jedes Zugeständnis. Die erstellten Systemanalysen sind zwangsläufig oberflächlich und unvollständig und die daraus folgenden Aufwandsabschätzungen viel zu niedrig. Wer seriös kalkuliert und die entsprechenden Zeiträume zur Grundlage seines Angebots macht, wird kaum jemals einen Auftrag erhalten.

Unabhängig davon unterschätzen die meisten Softwareentwickler bzw. ihre Projektleiter die erforderliche Entwicklungszeit, weil sie beispielsweise nicht in Betracht ziehen, wie häufig die Entwickler bei der Arbeit gestört (z. B. durch eigenen oder fremden Kommunikationsbedarf) oder behindert werden (z. B. durch Unzulänglichkeiten der Infrastruktur). Nicht zuletzt verspäten sich verspätete Projekte noch durch zusätzlichen Kommunikationsbedarf aufgrund ständiger Änderungen, aber auch durch Konzentrationsmangel und Streß.

1.4 Arbeitsteilung

In der Softwareentwicklung läßt sich mehr und mehr eine Arbeitsweise beobachten, die dem klassischen Modell der tayloristischen Arbeitsteilung entspricht. Dies spiegelt sich beispielsweise in dem zunehmenden Spezialistentum unter den Softwareentwicklern wieder. Während Datenbankprogrammierer die Strukturierung von und den Zugriff auf Datenbanken realisieren, entwerfen GUI-Entwickler einzig und allein grafische Benutzungsoberflächen. Die Liste der Spezialisierungen läßt sich beliebig fortsetzen. Softwareentwickler dieser Art kennen nur schmale

Schnittstellen, über die sich ihre Programme austauschen. Gewissermaßen ein Abbild dieser schmalen Programmschnittstellen bilden die menschlichen „Kommunikationsschnittstellen" zwischen den Entwicklern und Auftraggebern dieser Systeme.

Eine solche Fragmentierung von Arbeit ist eine Folge der zunehmenden Komplexität von Softwareentwicklung und gleichzeitig Ausdruck der Hoffnung von Projektleitern und (oftmals selbsternannter) Software-Gurus, eine Effizienzsteigerung bei der Softwareentwicklung zu erreichen. Inwieweit dieses pseudoindustrielle Verständnis von „Software-Engineering" tatsächlich trägt, sei im Moment dahingestellt.

1.5 Der integrierte Kunde

Die Spezifikation eines Softwaresystems läßt sich nicht in einem Wurf erledigen, schon gar nicht der eines komplexen Systems. Hierzu ist eine enge Zusammenarbeit zwischen Auftraggeber und Hersteller notwendig, die über die Verabschiedung eines juristisch definierten Pflichtenheftes hinausgeht. Kern einer solchen Zusammenarbeit ist die gemeinsame Erstellung eines problemadäquaten Anforderungskatalogs und eines darauf aufbauenden Systemkonzepts, das eine iterative Vorgehensweise nicht nur stillschweigend duldet, sondern auf ihr aufbaut.

Das Konzept einer solchen partizipativen Projektstruktur wird dadurch gestört, daß es kaum gemeinsame organisatorische oder technische Grundlagen für die Zusammenarbeit von Mitarbeitern unterschiedlicher Betriebe, wie hier zum Beispiel Auftraggeber und Auftragnehmer, gibt. Die Zusammenarbeit reduziert sich in der Praxis meist auf den Austausch und das Kommentieren von Dokumenten. Im günstigsten Fall werden einige wenige Mitarbeiter für ein paar Tage zusammengesetzt. Ein die gesamte Entwicklungs- und die spätere Nutzungsphase überspannendes Konzept einer dichten und wirkungsvollen Kommunikation ist praktisch nicht vorhanden. Die bewußte, konstruktive Auseinandersetzung mit den unterschiedlichen Interessenlagen von Anwendern, Benutzern und Softwareentwicklern findet nicht statt.

Ein Softwareprojekt ist nach seiner Auslieferung nicht abgeschlossen. Erst die reale Nutzung unter Betriebsbedingungen erlaubt relevante Rückschlüsse auf Schwächen und Unzulänglichkeiten. Nur durch intensive Kommunikation zwischen den Entwicklern, den Betreibern einer Software und den Endbenutzern läßt sich die Software aufgaben- und benutzergerecht weiterentwickeln. Nicht nur die in zahlreichen wissenschaftlichen Publikationen über partizipative Systementwicklung empfohlenen Strategien und Projektmodelle finden keinen Weg in die Praxis – selbst elementare, offensichtliche Erfahrungen werden systematisch ignoriert.

2. Kooperatives Entwickeln

Was in zahlreichen Fällen als tayloristisches Zerfasern von Arbeit niederschlägt, erscheint gleichzeitig als eine unumgängliche Notwendigkeit und Chance: arbeitsteiliges und kooperatives Entwickeln. Große Projekte, ob es der Bau der Pyramiden oder das Entwickeln eines großen Telekommunikationssystems ist, lassen sich nur durch Aufbau einer arbeitsteiligen Projektorganisation erreichen.

Ein solches Leitbild der Projektarbeit stellt aber durch die aktive, konstruktive Vorgehensweise andere Anforderungen als die reaktive und tendenziell destruktive Art des tayloristischen Zerteilens. Dabei dominieren die drei Dimensionen Inhalt, Zeit und Raum die Anforderungen an adäquate organisatorische Unterstützungen und technische Hilfsmittel.

2.1.Inhaltliche Verteilung

Nach den Erkenntnissen der Systemtechnik (Systems Engineering) lassen sich Systeme als hierarchische Gebilde aus Subsystemen und Komponenten bestehend beschreiben, wobei zwischen Handlungs- und Sachsystemen unterschieden wird. Handlungssysteme sind soziale Arbeitssysteme, wie z. B. eine Softwareentwicklungsabteilung, Sachsysteme sind technische Artefakte wie z. B. Softwaresysteme (vgl. Ropohl 1979).

Die Kunst der Softwareentwicklung besteht nun darin, eine möglichst gute Abbildung des Handlungssystems (Entwicklungsorganisation) auf das Sachsystem (zu entwickelndes Softwaresystem) herzustellen. Im Idealfall weisen beide Systeme ähnliche Strukturen auf, so daß bestimmte Softwaresubsysteme von bestimmten Softwareentwicklungsteams realisiert werden können, ohne daß dies zu einer technisch schlechteren Lösung aufgrund der Strukturierung führt. Auf diese Weise impliziert die Struktur des Softwaresystems die Struktur der Entwicklungsorganisation und umgekehrt und führt gleichzeitig zu einer transparenten Arbeitsteilung sowie zu einem transparenten und funktionsfähigen Softwaresystem.

Sieht man von den offenbar unvermeidlichen Stabsfunktionen einmal ab, so setzt sich eine solche Projektstruktur in der kommerziellen Softwareentwicklung aus einer ganzen Reihe von Entwicklungsteams zusammen, die ein Abbild der Systemstruktur sind. Jedes Entwicklungsteam ist dabei für ein oder mehrere Subsysteme oder Komponenten zuständig. Zwischen den Teams gibt es – entsprechend der technischen Schnittstellen der Systemteile – „Kommunikationsschnittstellen", die entweder im voraus organisatorisch geplant oder während der Arbeit zwangsläufig informell entstehen.

Die Kompetenz einer Firma kommt bei dieser problemorientierten, inhaltlichen Verteilung u. a. in der Fähigkeit zum Ausdruck, die zugrundeliegende Struktur richtig zu erkennen und eine ad-

äquate Umsetzung in der Projektstruktur zu finden. Erschwert wird eine solche Einschätzung durch den Umstand, daß diese Umsetzung über die Projektlaufzeit nicht zwangsläufig konstant ist und eine „optimale Lösung" bestenfalls rückbetrachtend gefunden werden kann.

2.2 Zeitliche Verteilung

Ein Softwareprojekt mit seinen zu erstellenden Systemteilen läßt sich gemäß der inhaltlichen Verteilung auf eine Zeitachse projizieren. In Abhängigkeit vom Umfang eines Projekts und der verfügbaren Entwicklungszeit entsteht dabei ein mehr oder weniger komplexes Bild paralleler oder sequentieller Arbeitspakete.

Der für die inhaltliche Verteilung notwendige Kommunikationsbedarf wird dabei mit einer zeitlichen Komponente verknüpft. Auf diese Art wird u. a. veranschaulicht, daß es notwendig werden kann, daß Kommunikationsvorgänge zeitlich versetzt zu erfolgen haben.

Parallelisierte Aktivitäten haben den Vorteil, daß die Kommunikation direkt stattfinden kann und lediglich die Ergebnisse – ggf. mit ihrer Herleitung – dokumentiert werden müssen. Sobald Abstimmungsprozesse wie bei sequentiellen Arbeitspaketen über die Zeit verteilt stattfinden müssen, sind zusätzliche Maßnahmen aufgrund der langen Zeiträume zwischen den Entwicklungsphasen notwendig, so etwa bei einer zyklischen Vorgehensweise.

2.3 Räumliche Verteilung

Neben der inhaltlichen und zeitlichen Verteilung von Projektaktivitäten arbeiten die Entwicklungsteams im allgemeinen räumlich getrennt voneinander. Die räumliche Verteilung wirkt sich bei direkt benachbarten Arbeitsplätzen ebenso auf die Art der Kooperation aus, wie im anderen Extrem bei der Verteilung auf unterschiedliche Kontinente.

Unterschiede in der individuellen Arbeitsweise verstärken sich dabei über die der Arbeitsgruppe und des jeweiligen Betriebsstandortes hinaus bis hin zu unterschiedlichen Arbeitskulturen in anderen Ländern und Kontinenten. Das Finden einer gemeinsamen Sprache ist dabei eher das kleinste der Probleme.

Auch hier müssen eine geeignete Kommunikationsinfrastruktur wie auch Projektorganisation das Zusammenwirken sicherstellen. Je weiter die räumliche – und damit kulturelle – Verteilung ausgeprägt ist, um so wichtiger ist die Beantwortung der Frage nach dem Warum der Trennung. Die räumliche Verteilung sollte sich – ebenso wie die inhaltliche und die zeitliche – aus der Aufgabenstellung ergeben und nicht ausschließlich einer gleichmäßigen Verteilung der Kapazitäten dienen. Rein wirtschaftliche Überlegungen können hier teuer zu stehen kommen.

Die kulturelle Verteilung der Projektarbeiten birgt stets die erhöhte Gefahr von Mißverständnissen, Fehlinterpretationen und vorsätzlich unterschiedlichen Ausprägungen von Arbeitsaufgaben – eben weil es signifikante Unterschiede in der Arbeits-, Kooperations- und Kommunikationskultur gibt. Dieses erhöhte Risiko einzugehen lohnt nur, wenn spezifische Kompetenzen der Teams oder aus der Aufgabenstellung heraus erwachsende Anforderungen dies erfordern. Soll ein Softwaresystem beispielsweise nicht weltweit ausschließlich in der US-Version vertrieben werden, sondern die japanische Version spezifisch japanischen Anforderungen Rechnung tragen, die finnische den finnischen Anforderungen usw., so wäre dies ein wichtiger Grund für eine interkulturelle Vorgehensweise.

3. Der Softwareentwicklungsprozeß

Die inhaltliche, zeitliche und organisatorische Struktur eines Softwareentwicklungsprojektes definiert einen Softwareentwicklungsprozeß. Im folgenden werden die wesentlichen Merkmale eines solchen Entwicklungsprozesses im einzelnen betrachtet, um daraus die entscheidenden Kooperations- und damit Kommunikationsbedürfnisse abzuleiten.

3.1 Phasen

Die übergeordneten Arbeitsschritte in der Softwareentwicklung werden allgemein als Phasen bezeichnet. Es gibt eine Vielzahl von Phasenkonzepten, die im Rahmen des Software-Engineerings entwickelt wurden (vgl. z. B. Sommerville 1995).

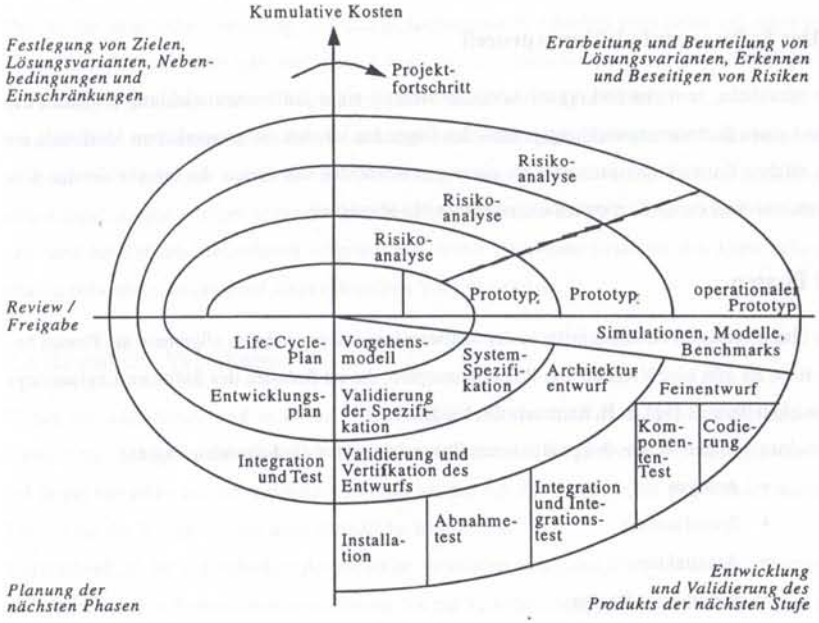
Gemeinsam finden sich in den praktizierten Phasenkonzepten die folgenden Phasen:

- Analyse
- Spezifikation
- Architektur
- detailliertes Design
- Implementierung
- Test
- Integration
- Installation
- Wartung

In der Regel ist in der kommerziellen Softwareentwicklung bei größeren Projekten kein Team phasenübergreifend tätig; die Teams wechseln in ihrer Zuständigkeit. Zwischen den Phasen werden die als relevant angesehenen Projektinformationen wie Spezifikationen, Softwaremodule

oder Testberichte – oft als Deliverables bezeichnet – für nachfolgende Phasen bereitgestellt. Dabei erfolgt die Übergabe an ein anderes Team nur selten direkt und persönlich, die Deliverables werden für anonyme Empfänger produziert.

Die Praxis zeigt aber, daß sich vor allem benachbarte Phasen zeitlich überlappen müssen. Dies spart letztendlich Gesamtentwicklungszeit, weil es den an benachbarten Phasen beteiligten Teams erlaubt, frühzeitig – auch informell – zu kommunizieren (Cleland 1994: 364). Man weiß, für wen man die Deliverables produziert und auf wessen Deliverables man aufbaut. Abstimmungsprozesse zwischen den Teams können ablaufen, bevor formale Deliverables geliefert werden.



Ursprüngliches Spiralmodell des Softwareentwicklungsprozesses (Boehm 1988:64; deutsche Bezeichnungen nach Pomberger/Blaschek 1993:27)

Scharfe formale Übergänge zwischen den einzelnen Phasen sind ohnehin nur bei hochstandardisierten Produktentwicklungen möglich bzw. sinnvoll. In allen anderen Fällen führt ein „blindes" Vertrauen in die definierten Schnittstellen erfahrungsgemäß zu unnötigen Verzögerungen und Fehlern.

Das hinreichend bekannte Wasserfallmodell (vgl. Royce 1970) mußte auch in weiten Teilen der kommerziellen Softwareentwicklung einem iterativen Prozeßverständnis weichen. Der gesamte Entwicklungszyklus, bestehend aus den zuvor beschriebenen Phasen, wird im Sinne eines iterativen Prozesses mehrfach durchlaufen, ähnlich dem Boehmschen Spiralprozeß (siehe Abb. 1 / vgl. Boehm 1988).

Entscheidend ist bei dieser Vorgehensweise, daß es so erst möglich wird, Korrekturen, Verfeinerungen oder neue Erkenntnisse in eine nachfolgende Iteration einzubringen. Dabei müssen unbedingt sämtliche in einer Iteration gemachten Erfahrungen und gewonnenen Erkenntnisse in die Folgeiterationen einfließen.

Hierbei sind in der Praxis oft erhebliche Defizite zu beobachten, die in der Regel auf wechselnde Teams, inadäquate organisatorische Rahmenbedingungen und einem Verharren in untauglichen Formalismen zurückzuführen sind.

3.2 Evolution des Produkts

Die iterative Vorgehensweise wird nicht nur auf ein sich in Entwicklung befindliches Softwaresystem angewandt. Auch ein bereits ausgeliefertes Produkt entwickelt sich über mehrere Releases bzw. Versionen evolutionär weiter. Dies ist ein Tribut, den man an die Komplexität der Aufgabe bei der Entwicklung großer Systeme zahlt. Auf diese Art schafft man eine Grundlage für den Kommunikationsprozeß zwischen Entwickler und Auftraggeber. Es zeigt sich oft, daß der Auftraggeber erst durch eine erste Lieferung in die Lage versetzt wird, seine Kritik am Vorhandenen so detailliert und nachvollziehbar zu artikulieren, um von den Entwicklern verstanden zu werden.

Hinzu kommt der Effekt, daß sich durch die ausgelieferten Systeme die Arbeitsaufgaben der Benutzer verändern – bis hin zu Veränderungen der Arbeitsorganisation. Von diesen Veränderungen sind auch die Anforderungen an die Softwaresysteme betroffen; was zuvor richtig und wichtig erschien, kann nun von den Benutzern gänzlich anders eingeschätzt werden.

Durch die evolutionäre Entwicklung in mehreren Releases ergibt sich die Möglichkeit, ein Produkt nachzuspezifizieren, zu überarbeiten, zu verfeinern. In der Praxis steht diesen Vorteilen ein klarer Nachteil gegenüber: die geringe Häufigkeit von Releases – typischerweise ein Release in ein bis zwei Jahren. Einerseits eignen sich solche Releaseabstände kaum für die systematische Erweiterung, Fehlerbeseitigung und Optimierung, andererseits paßt bei der Kurzlebigkeit heutiger Produkte diese Vorgehensweise immer weniger in die vom Markt diktierten Zeitraster.

3.3 Deliverables

Qualität und Verfügbarkeit von zwischen den Phasen ausgetauschten Informationen bestimmen maßgeblich die Qualität des Softwaresystems. Ob und inwieweit diese Deliverables für die Weiterbearbeitung bzw. Weiterverwendung durch andere Teams taugen, ist ebenfalls von entscheidender Bedeutung.

In den meisten Praxisprojekten werden Deliverables in Form von Papierdokumenten ausgetauscht. Das Verwenden von Papier hatte sich recht schnell als Darstellungsmedium aufgedrängt, da ein Großteil der Informationen für den Auftraggeber ohnehin in Form von Papierdokumenten zu liefern ist. Unabhängig davon, ob konventionelle Schriftstücke überhaupt dazu taugen, ein Softwaresystem zu dokumentieren, ist das Erstellen und Ändern solcher Dokumente ein sehr zeitraubender Vorgang. Der Grund dafür liegt neben dem eigentlichen Erstellungsaufwand für die Deliverables vor allem in den Reviewprozeduren. Jedes Deliverable muß üblicherweise sowohl von der Projektleitung, als auch vom potentiellen Nutzer des Deliverables in einem formalen Review geprüft und akzeptiert werden, bevor damit gearbeitet werden darf.

Diese Statik der Kommunikation – wenn man den Austausch von Dokumenten überhaupt als Kommunikation bezeichnen will – beeinflußt maßgeblich die (Un-)Flexibilität des ganzen Entwicklungsprozesses. Hier gilt es, nicht nur adäquatere Darstellungsformen für die zu übermittelnden Inhalte zu finden, auch der Umgang mit den Deliverables ist verbesserungsfähig. Die informellen Strukturen einer eingespielten Organisation und der Informationsaustausch in einer solchen kann hier auch für Großprojekte Vorbild sein. So sollten Deliverables beispielsweise kein juristisches Instrument zum firmeninternen Haftungsausschluß sein oder den „Dienst nach Vorschrift" propagieren; vielmehr geht es darum, alle wichtigen Informationen an jene weiterzuleiten, die diese dringend benötigen, um die richtigen Entscheidungen treffen zu können. Deliverables sind Hilfsmittel der Kommunikation, sie ersetzen sie nicht.

3.4 Evolution des Prozesses

Neben dem auftragsbezogenen Softwareentwicklungsprozeß läuft in den Betrieben teamübergreifend ein weiterer Prozeß ab. Dieser Metaprozeß ist der Prozeß des ständigen Zugewinns an Erfahrung, bezogen auf die Entwicklung der eigentlichen Software, aber auch auf die Interaktion mit den Auftraggebern. Die Summe der Erfahrungen aller an dem Projekt Beteiligten ermöglicht es der softwareentwickelnden Organisation, strategische Fehler zu vermeiden und Folgeprojekte effizienter und effektiver durchzuführen (Cleland 1994: 417).

Mit diesem Metaprozeß stößt man endgültig an die derzeitigen Grenzen organisierter Softwareentwicklung und deren Evolution. Die Einbeziehung aller am Projekt Beteiligten in einen Metaprozeß erschöpft sich in der Praxis in der Regel in Flurgesprächen, Formularen für Verbesserungsvorschläge und im besten Fall in ein oder zwei Tagen Manöverkritik nach Lieferung eines Releases. Eine solche Manöverkritik wird häufig allerdings einer vermeintlich motivationsfördernden gesellschaftlichen Veranstaltung geopfert, bei der man sich nach all den Anstrengungen gegenseitig auf die Schultern klopft oder in freundlicher Manier vorsichtig einige mögliche Defizite der Zusammenarbeit erwähnt.

Es gibt üblicherweise weder einen bewußt nachvollzogenen Metaprozeß noch dafür notwendige Kommunikationsmittel. Nicht nur die vielleicht gewonnenen guten Vorsätze für das kommende Projekt weichen schnell wieder der sogenannten Realität und ihren Zwängen, selbst das Erkennen des Metaprozesses wird beiseite geschoben.

Dabei ist das Erreichen eines Meilensteins die beste Gelegenheit, potentiell sich herauskristallisierende Erkenntnisse festzuhalten und für das eigene Handeln umzusetzen. Bereits das Anerkennen des Metaprozesses und seine offene Diskussion – nicht auf den Fluren und nicht nur unmittelbar auf den letzten Auftrag bezogen – verstärkt den Prozeß und vergrößert den sich daraus für die Gesamtorganisation ableitenden Gewinn enorm. Die Akzeptanz des Prozesses und die Auseinandersetzung mit seinen Inhalten ist eine wesentliche Voraussetzung für organisationales Lernen – und der beste Schutz vor sich ständig wiederholenden Fehlern.

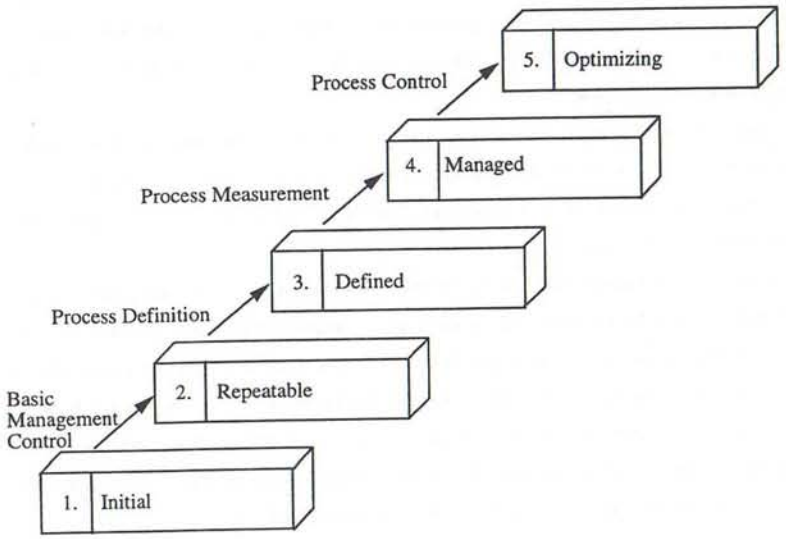
4. Auswege

Betrachtet man die Ausführungen einiger Vordenker des Software-Engineerings, so liegt offensichtlich der Schluß nahe, daß sich die Probleme der Softwareentwicklung durch weitere Formalisierung und Verfeinerung, sprich durch Verschärfung, des Softwareentwicklungsprozesses beseitigen lassen. Dabei sollen alle Projektphasen noch genauer beschrieben, die Kommunikationsbeziehungen und die jeweiligen Deliverables noch formal eindeutiger festgelegt werden.

Eine solche Strategie wird z. B. durch das CMM (Capability-Maturity-Model, „Potential-Reife-Modell") verfolgt (Humphrey 1990). Die Umsetzung dieses Modells führt zu einem immer formaleren und damit rigideren Prozeß, der im allgemeinen von einer stark zunehmenden Taylorisierung der Arbeit geprägt ist. Das CMM beschreibt fünf Stufen der Prozeßqualität, vom Initial bis zum Optimizing (vgl. Abb. 2).

Aus diversen Erfahrungen in der Anwendung von CMM in größeren Softwareentwicklungsprojekten ist zu vernehmen, daß allenfalls das Erreichen der dritten Stufe (*Defined*) erstrebenswert und nützlich ist. Die höheren Stufen führen zu einer nicht-linearen Zunahme des Entwicklungs-

aufwandes sowie zu erhöhter Inflexibilität und Demotivation – zusammen mit damit verbundener Leistungsreduktion bis hin zur Abwanderung der Mitarbeiter. Diese Stufen wurden bislang auch nur von einer Handvoll Unternehmen weltweit erreicht, was ebenfalls eine gewisse Skepsis rechtfertigt.[1]



Die fünf Qualitätsstufen (Process Maturity Levels) des Maturity-Model (CMM) nach Humphrey 1990

Ähnliche Erfahrungen sind im Umgang mit anderen Methoden gemacht worden, mit mehr oder weniger fatalen Konsequenzen für die jeweiligen Betriebe. Was sind aber die Alternativen zu solchen, sich offenbar kontraproduktiv auswirkenden Konzepten? Was können softwareentwikkelnde Betriebe tun, um einen Ausweg aus ihrer eigenen Softwarekrise zu finden?

Aus dem bisher dargestellten wird deutlich, daß bei Softwareentwicklungsprozessen eine ganze Reihe unnötiger organisatorischer und infrastruktureller Defizite zu beobachten sind. Strategien zur Behebung dieser Defizite haben die Aufgabe, die Kanalisierung, Stabilisierung und Kristallisierung von Kommunikationsprozessen zu leisten. Zu diesem Zweck sind neben einer verbes-

[1] Es soll hier nicht verschwiegen werden, daß CMM – unabhängig von jenen fünf Qualitätsstufen – eine Reihe von organisatorischen und sozialen Komponenten beinhaltet, die zweifellos sinnvolle Bereicherungen für Softwareentwicklungsprojekte darstellen. Dies ändert allerdings nichts an den Problemen, die der Einsatz der Methode in der Praxis mit sich bringt.

serten, weil problemadäquateren, Infrastruktur eine Reihe von organisatorischen und sozialen Maßnahmen gefordert.

Organisation und Infrastruktur von Softwareentwicklungsprozessen müssen grundsätzlich drei Formen von Kooperation unterstützen:

- inhaltliche Kooperation,
- räumliche Kooperation,
- zeitliche Kooperation.

4.1 Inhaltliche Kooperation

Eine der wesentlichsten Grundlagen für eine effiziente, zielgerichtete Softwareentwicklung ist das Sicherstellen möglichst optimaler Rahmenbedingungen für die Zusammenarbeit aller Personen, die substantielle Beiträge zur Gestaltung des Produkts liefern können. Man sollte annehmen, daß dies eine Selbstverständlichkeit ist. Leider scheitert das „Zusammenbringen der richtigen Leute" jedoch oft an firmenpolitischen Prinzipien, z. B. bezüglich der Kooperation zwischen Kunde und Hersteller, oder an organisatorischen Prinzipien, z. B. bezüglich der Zusammenarbeit der Mitarbeiter beim Hersteller (Cleland 1994: 378).

Inhaltliche Kooperation beginnt mit der Zusammenarbeit von Vertretern des Auftraggebers und des Softwareentwicklers beim Erarbeiten der Anforderungen im Bereich des Requirements Engineerings. Dabei muß eine alle Phasen umfassende und alle Iterationen überspannende, kontinuierliche Zusammenarbeit organisiert werden. Auf diese Weise kann der Kunde fortwährend kontrollieren, wie seine Anforderungen umgesetzt werden und inwiefern die vereinbarten Anforderungen wirklich die intendierten Ziele zu erreichen helfen. Grobe Fehlentwicklungen durch Fehlinterpretationen oder falsche Anforderungen können auf diese Weise frühestmöglich erkannt und korrigiert werden. Hierbei besteht allerdings die Gefahr eines ständigen Redefinierens und Infragestellens des Produkts, deshalb ist hier die Vereinbarung klarer Spielregeln für die Einflußnahme gefordert.

In der Praxis beschränkt sich die Zusammenarbeit auf die Definition bzw. Abnahme des Softwaresystems in den ersten bzw. letzten Phasen. Eine weitergehende Kooperation erfordert neben einer geeigneten Kommunikationsinfrastruktur auch eine solide Vertrauensbasis zwischen Auftraggeber und Hersteller, die nur selten anzutreffen ist. Exponentiell anwachsende Kosten könnten allerdings in Zukunft eine wirtschaftlich motivierte Kooperation der beschriebenen Form zustande kommen lassen.

Hinter dieser Vorstellung einer inhaltlichen Kooperation steht die Vorstellung eines virtuellen Teams, vergleichbar einem virtuellen Unternehmen. Solche Teams könnten z. B. in den Berei-

chen Architektur und Design arbeiten, etwa Systemingenieure, Systemarchitekten und Softwareentwickler. Ähnliches gilt für die Verifikation, wenn Integrationsmitarbeiter, Softwareentwickler und Systemarchitekten gemeinsam nach Fehlern suchen, oder für die Zusammenarbeit von QA-Beauftragten, Systemingenieuren und Kunden bei der Validierung. Diese Teams arbeiten bis zu einem bestimmten Ergebnis – weitestgehend unabhängig von Zeit und Ort – an gemeinsamen Inhalten. Ein Mitarbeiter kann dabei zu einem Zeitpunkt mehreren Teams angehören. Bislang bebzw. verhindern starre Organisationsstrukturen mehrfache Teamzugehörigkeiten. Dies führt zur Verarmung der Tätigkeitsprofile von Projektmitarbeitern auf allen organisatorischen Ebenen und wirkt der organisationsinternen Diffusion von Know-how entgegen.

Ebenso negativ wirkt sich die Trennung planender und ausführender Tätigkeiten – die Trennung von Kopf und Hand – aus. Wer ausschließlich die Tätigkeiten anderer plant, kommt zu genauso unangemessenen und ineffizienten Resultaten wie jene, die vorgeplante Tätigkeiten auszuführen haben, ohne verstehen zu können, was sie dort eigentlich machen. Im Fall der Softwareentwicklung hat eine solche Trennung noch fatalere, weil schneller greifende, Folgen als in anderen Branchen. Entwicklung und Herstellung sind bei Softwaresystemen nicht voneinander zu trennen. So verkennen viele CASE-Strategien die Tatsache, daß Konstruktion und Produktion – sofern man diese Begriffe in diesem Kontext überhaupt anwenden kann – untrennbare Entwicklungsabschnitte sind (vgl. Roth 1995: 149 ff.).

Die inhaltliche Kooperation bedarf nicht nur organisatorischer und sozialer Unterstützung; es gilt gleichzeitig, eine problemadäquate technische Infrastruktur bereitzustellen. Bisher verfügbare Systeme – insbesondere zahlreiche CASE-Werkzeuge – laborieren in der Regel am Symptom, nicht aber an der Ursache. So versucht man oft, der geforderten Dokumentationsflut nachzukommen, indem man halbautomatische Dokumentengenerierung betreibt – wodurch letztlich die Qualität des Produkts um keinen Deut besser wird.

Ein Blick in erfolgreiche softwareentwickelnde Betriebe vermittelt schnell einen Eindruck davon, welche Werkzeuge wirklich genutzt werden. So sind elektronische, moderierte Diskussionsforen, die oftmals mit Hilfe von „umfunktionierten" Electronic-Mail-Systemen realisiert werden, um ein vielfaches erfolgreicher und damit produktiver als monolithische „Softwareentwicklungsumgebungen".

Inhaltliche Kooperation erfordert in erster Linie die Information der am Prozeß Beteiligten – so banal dies auch klingen mag. Nicht der Projektmanager an der Spitze der Hierarchie muß über alle Hindernisse, Probleme, Lösungen und Absprachen genauestens informiert sein, sondern jene, die die Software entwickeln und deren Handlungsgrundlage diese Information darstellt. Gemeinschaftlich aufgebaute Requirements-, Design- oder Problemdatenbanken entlasten nicht nur

den einzelnen Entwickler, sondern das gesamte Team und letztlich die gesamte Organisation. Wenn klar ist, wie man an die relevanten Daten herankommt, wie diese zu interpretieren sind und wenn feststeht, daß diese Information dann auch noch gültig ist, so trägt dies in erheblichem Umfang zur Qualität der Software und zur Steigerung der Termintreue bei.

Die Hilfsmittel des Softwareentwicklers müssen ihm nicht nur Einsicht in den aktuellen Stand des Projekts gewähren, sondern auch dessen Historie transparent machen. Dazu dienen gemeinsam genutzte „Projektverfolgungswerkzeuge", die neben einer getroffenen Entscheidung auch Auskunft über den Weg zu dieser Entscheidung geben. Ein Beispiel hierzu findet sich in Fischer et al. 1996; dort wird eine Datenbank beschrieben, die die ohnehin von den Entwicklern ausgetauschten EMails sammelt und gemäß ihrer Thematik sortiert. Wer wissen will, wie man zu einer Entscheidung über ein Entwicklungsdetail gekommen ist und es beispielsweise aufgrund der räumlichen oder zeitlichen Verteilung nicht möglich ist, einen Beteiligten zu Rate zu ziehen, kann sich mit diesem Werkzeug schnell einen Überblick verschaffen. Die dort zu findenden Aufzeichnungen sind zwar äußerst knapp gehalten, aber gerade dadurch für den Insider effizienter als endlose Prosatexte. Eine solche Datenbank ist insbesondere bei Großprojekten wesentlich besser geeignet als ein Hypertextsystem, da das Auffinden der relevanten Einträge dort zuviel Zeit in Anspruch nimmt und die Gesamtstruktur für eine zielgerichtete Navigation schnell zu komplex wird.

Eine ähnliche Funktion haben projektöffentliche „lebende" Vorhabenshandbücher zur kontinuierlichen und transparenten Prozeßoptimierung. Diese Handbücher werden nicht nach Projektende verfaßt, sondern durchleben den gesamten Prozeß der Entwicklung. Sie geben in Teilgruppen oder von einzelnen Entwicklern gemachte Erfahrungen weiter und steigern so die Effizienz und Effektivität des Gesamtprozesses.

4.2 Räumliche Kooperation

Neben der üblichen, unvermeidbaren räumlichen Trennung von Auftraggeber und Softwareentwickler erzwingen große Projekte die Zusammenarbeit von Entwicklungsteams, die räumlich verteilt sind. Die räumliche Verteilung nimmt dabei – in Abhängigkeit von der Unternehmensstruktur und der Größe des Projekts – unterschiedliche Ausmaße an, in der Regel weit mehr als sich aus der Problemstellung heraus begründen ließe. Dies beginnt mit einer Verteilung auf unterschiedliche Räume, Stockwerke und Gebäude und steigert sich schnell bis hin zu einer Verteilung auf unterschiedliche Standorte in mehreren Ländern, oftmals sogar auf verschiedenen Kontinenten. Die Verteilung auf mehrere Standorte ist bei Projekten, wie sie beispielsweise im Bereich der Telekommunikation zu finden sind, wo mehrere hundert oder tausend Entwickler an

einem Produkt arbeiten, inzwischen Alltag. Erschwert wird eine solche Verteilung von Projekten zusätzlich durch die zunehmend sich verbreitenden Outsourcing-Strategien, wobei die Entwicklung ganzer Subsysteme an andere Unternehmen vergeben wird, als ob es eine Zulieferindustrie wie etwa im Automobilbau gäbe.

Räumlich verteilte Kooperationen werden auf Basis einer sehr dünnen technologischen Grundlage realisiert, der Strukturierung eines Systems in viele Subsysteme und der Definition von möglichst engen Schnittstellen zwischen diesen Subsystemen. Solche Strukturierungen erzeugen oft sehr künstliche und dem Produkt nicht sehr förderliche Systemstrukturen, die zur Projektlaufzeit nicht mehr veränderbar sind.

Damit aber eine der Problemstellung gerecht werdende Systemstruktur in der Praxis überhaupt gefunden und umgesetzt werden kann, muß es möglich sein, Teams unabhängig von ihrem Standort eine sehr enge und – erfahrungsgemäß – auch eine sehr persönliche Zusammenarbeit zu ermöglichen. Hier müssen allerdings noch Grundlagen einer inter- und multikulturellen Zusammenarbeit geschaffen werden. Erfahrungen im europäischen Raum deuten bereits auf große Auswirkungen durch unterschiedliche Arbeitskulturen und Mentalitäten hin. Hinzu kommt das Problem von sehr unterschiedlichen Qualifikationen durch stark variierende Berufausbildungen. Im außereuropäischen bzw. interkontinentalen Kontext gestaltet sich dies noch schwieriger.

Ohne geeignete Qualifizierungsmaßnahmen sind solche Kooperationsformen von vornherein zum Scheitern verurteilt. Eine frühzeitige Vorbereitung der einzelnen Mitarbeiter und die Vermittlung einer erweiterten Kommunikations- und Kooperationskompetenz ist unumgänglich, insbesondere im Kontext internationaler Kooperationsvorhaben.

Die räumliche Verteilung eines größeren Entwicklungsvorhabens darf nicht zum Selbstzweck oder Politikum werden. So haben beispielsweise Firmen, die in zwei verschiedenen Ländern Niederlassungen unterhalten, hinreichend schlechte Erfahrungen mit alternierend verteilten Hierarchiestufen gemacht. Man hatte versucht, die hierarchisch übergeordnete Organisationsebene im jeweils anderen Land zu etablieren. Diese Vorgehensweise bewirkt, daß sich nach Ländern getrennt zwei Organisationen herausbilden, die jede für sich zwar bedingt handlungsfähig ist, namentlich durch Überspringen einer Hierarchiestufe, miteinander aber fast gar nicht kooperieren. Auf diese Weise schadet man nicht nur dem zu entwickelnden Produkt und demotiviert die Mitarbeiter, man gefährdet sogar die Existenz der Gesamtorganisation.

Über qualifikatorische und arbeitsorganisatorische Maßnahmen hinaus erfordert die räumliche Kooperation auch eine adäquate Unterstützung durch infrastrukturelle Maßnahmen. Diese Infrastruktur hat in erster Linie die Aufgabe, die geographische Distanz zwischen den Beteiligten überwinden zu helfen. Ziel ist es, Informationen auszutauschen, Aufgaben und Probleme zu diskutieren, den anderen zu überzeugen – sprich: zu kommunizieren, obwohl die anderen Team-Mitglieder möglicherweise tausende von Kilometern voneinander getrennt sind.

Die einfachste Möglichkeit, das Treffen der Beteiligten an einem Ort zu einem bestimmten Zeitpunkt, wird bei größeren Distanzen offenbar nur selten bzw. erst ab der Managementebene als gleichberechtigte Lösung zu technischen Kommunikationshilfsmitteln angesehen. Die entstehenden Kosten und der vermeintliche Zeitverlust werden als zu hoch eingestuft. Unmittelbare, persönliche Treffen zwischen den Beteiligten sind für den Erfolg bzw. für die Effizienz eines Teams wesentlich. Kennen sich die Team-Mitglieder bereits von vorherigen gemeinsamen Projekten, so verlieren diese Treffen an Bedeutung. Ergibt sich aus der Aufgabenstellung die Notwendigkeit von beispielsweise täglichem Informationsaustausch, so erfordert dies technische Kommunikationshilfsmittel.

Das verbreitetste und am häufigsten genutzte Kommunikationshilfsmittel ist sicherlich das Telefon; nach einer Repräsentativuntersuchung des Instituts Arbeit und Technik verfügen 92,5 % aller Beschäftigten in Deutschland an ihrem Arbeitsplatz über ein leitungsgebundenes Telefon, 23,7 % haben ein Funktelefon bzw. Handy. Über den Verbreitungsgrad hinaus sind die Vorteile dieser Technik nicht unerheblich. So lassen sie sich problemlos durch Freisprech- und Mithöreinrichtungen in Gesprächsrunden einbinden oder erlauben den Mitschnitt. Wichtig ist ferner, daß alle Beteiligten mit diesem Werkzeug umgehen können und es nahezu weltweit ein selbstverständliches Element der Gesprächskultur ist. Sicherlich unterscheidet sich die Nutzung beispielsweise durch einen Börsianer in New York von der durch einen japanischen Handelsvertreter, verglichen aber mit den Anforderungen an den Nutzer eines Videokonferenzsystems sind diese gering (vgl. Dutke / Paul 1996, Dutke et al. 1996).

Videokonferenzsysteme spielten bisher in der Praxis eine eher untergeordnete Rolle. Die Gründe dafür können aber nur teilweise in den oft horrenden Kosten oder in technischen Unzulänglichkeiten sehen. Solange die Nutzung nicht vom Arbeitsplatz aus möglich war und spezielle Studios notwendig wurden, nutzte kaum ein Softwareentwickler dieses Hilfsmittel. Mit der Verfügbarkeit von Desktop-Videokonferenzsystemen auf Intra- bzw. Internetbasis tun sich hier neue Möglichkeiten auf (vgl. Dutke et al. 1996, Foks 1996).

Kombiniert man beispielsweise ein Desktop-Videokonferenzsystem mit einer Application-Sharing-Umgebung, so können damit Team-Mitglieder gemeinsam Entwicklungswerkzeuge wie Compiler oder Debugger benutzen oder Softwaremodule integrieren und testen, als ob sie zusammen an einem Computer sitzen, obwohl sie u. U. tausende von Kilometer trennen. Whiteboards dienen dabei als gemeinsame elektronische Notizblöcke.

Die Möglichkeiten solcher Werkzeuge mögen verlockend klingen. Man sollte aber keinesfalls

übersehen, daß der Metakommunikationsaufwand, beispielsweise für die optimale Positionierung von Kamera und Mikrofon, die richtige Beleuchtung, die adäquate Konfigurierung der Application-Sharing-Umgebung oder gar die Etablierung eines sozialen Protokolls oder den Aufbau eines gemeinsamen mentalen Modells, erheblich ist. So bedarf es viel Geduld und längerer Eingewöhnungsphasen, bis man auf diese Art und Weise wirklich produktiv ist (Dutke et al. 1996, Foks 1996). Hinzu kommt, daß Application-Sharing-Umgebungen noch erkennbare gestalterische Defizite aufweisen. So wird der Übergang der subjektiven Räume, etwa die Trennung von Privatem, Gemeinschaftlichem und Öffentlichem, dem Benutzer nicht oder nur unzureichend transparent gemacht (Dutke / Paul 1996, Dutke et al. 1996).

Bei interkontinental verteilten Teams gibt es neben der geographischen und kulturellen Distanz das Problem der Zeitzonen. So werden sich zumindest die regulären Arbeitszeiten bei Teams mit Mitgliedern z. B. aus den USA, Deutschland und Japan nur sehr kurz überlappen. Intensive Kooperation erfordert hier asynchron arbeitende Hilfsmittel.

Im einfachsten Fall greift man zur universell einsetzbaren EMail, die im Bereich der Softwareentwicklung das Telefax weitestgehend verdrängt hat. Ein besonderer Vorteil liegt in der Möglichkeit, Dokumente, Programmcode und Programme so zu versenden, daß mit ihnen ohne Medienbruch weitergearbeitet werden kann.

Die Anforderungen bei der kooperativen Softwareentwicklung gehen über den vergleichsweise simplen Austausch von Dateien hinaus. So ist eine gemeinsam geführte und allen Team-Mitgliedern jederzeit zugängige Datenbank, die aktuell und verläßlich Auskunft über Funktionsbeschreibungen, Organigramme und Strukturdefinitionen gibt, ebenso wichtig, wie eine Sammlung von Kommunikationsadressen. Eine solche Sammlung umfaßt sinnvollerweise sämtliche Telefon- und Telefaxnummer, EMail-Adressen der Team-Mitglieder, aber auch die Adressen der Mitglieder angrenzender Teams. Dazu gehört auch eine Beschreibung der unmittelbaren Aufgabenfelder, Zuständigkeiten und Kompetenzen. Dort muß erfaßt sein, wer bei welchen Entscheidungen zu konsultieren ist, z. B. wenn eine Design- oder Implementierungsentscheidung auch außerhalb des eigenen Teams Auswirkungen hat bzw. haben könnte.

Die Infrastruktur für solche Systeme ist vorhanden und zugrundeliegende Werkzeuge sind ebenfalls verfügbar. So wurde beispielsweise unter der Bezeichnung BSCW (Basic Support for Cooperative Work) ein auf dem http-Protokoll aufsetzendes Werkzeug zur Realisierung von gemeinsamen Arbeitsbereichen entwickelt (Appelt / Hinrichs 1997). Das BSCW-System stellt u. a. Zugriffsmechanismen zur Verfügung und koordiniert den Zugang zu sich ständig verändernden Dateisammlungen. Es handelt sich dabei nicht um ein monolithisches System, das den Anspruch erhebt, alles und jedes zu unterstützen – es stellt vielmehr einen Basisdienst zur Verfügung, der

vom Team entsprechend genutzt werden kann.

Bei Abstimmungsprozessen im Team ist es wichtig, trotz räumlicher Trennung gemeinsam z. B. eine Spezifikation erarbeiten oder ergänzen zu können. Handhabbare spezielle Datenbanken, die die gemeinsame Spezifikation von Schnittstellen unterstützen, darüber hinaus die gemeinschaftlich festgelegte Semantik der Spezifikation verständlich dokumentieren und ggf. auch anderen Teams im Rahmen der Übergabe von Deliverables in kurzer Zeit transparent machen, sind zwar noch Zukunftsmusik, werden aber nichtsdestoweniger bereits heute dringend benötigt. „Repositories" unterstützen zwar das Bearbeiten von Designdaten, über Lockingmechanismen zur Synchronisation von Schreib-Lese-Operationen hinaus helfen sie bei kooperativem, gemeinschaftlich kreativem Arbeiten aber in keinster Weise.

4.3 Zeitliche Kooperation

Für die zeitliche Verteilung bei der kooperativen Entwicklung von Softwaresystemen gibt es neben den aus der Aufgabenstellung erwachsenden Sequentialisierungen von Arbeitspaketen noch einen weiteren Grund, nämlich den der verschiedenen Arbeitszeiten. Flexible Arbeitszeitmodelle bieten zwar für den einzelnen Mitarbeiter zahlreiche Vorteile, sie erschweren aber die Organisation des Arbeitens im Team. Hinzu kommt, daß Team-Mitglieder möglicherweise zeitweilig nicht zur Verfügung stehen, weil sie durch zusätzliche Aufgaben in anderen Projekten gebunden sind.

Diese Situationen ähneln dem schon beschriebenen Fall der Kooperation über mehrere Zeitzonen hinweg und sind grundsätzlich von den Anforderungen zu unterscheiden, die aus der zeitlichen Verteilung infolge sequentieller Entwicklungsphasen erwachsen. Zwar kann man versuchen, die mangelnde Verfügbarkeit von Team-Mitgliedern durch technische Hilfsmittel zu kompensieren, dennoch handelt es sich in erster Linie um ein organisatorisches Problem. So darf der Druck des Marktes ein softwareentwickelndes Unternehmen nicht dazu verleiten, jeden sich bietenden Auftrag anzunehmen und seine Kapazitäten gleich mehrfach zu verplanen. Sicherlich ist es extrem schwierig, wenn nicht gar unmöglich, die „Produktionskapazitäten" eines Softwareunternehmens verläßlich zu ermitteln – erst recht, wenn es gilt, freie bzw. frei werdende Kapazitäten abzuschätzen (vgl. Roth 1995).

Softwareentwicklung findet in mehreren Phasen statt, die in der Regel mehrfach durchlaufen werden müssen. Sowohl den aus den verhältnismäßig großen Zeitabständen zwischen den Phasen, als auch die aus den noch längeren Zeiträumen zwischen den verschiedenen Versionen und Releases erwachsenden Schwierigkeiten kann man ebenfalls auf organisatorischer Ebene begegnen. So kann es sinnvoll und effektiv sein, Teams sich mit ihren Produkten identifizieren zu las-

sen und nicht in jedem Zyklus ein anderes Team auf die Weiterentwicklung anzusetzen. Grundsätzlich wirkt sich eine konsistente Zuständigkeitsverteilung positiv auf Effektivität und Effizienz aus.

Die Abhängigkeiten einzelner Phasen im Entwicklungszyklus können so komplex werden, daß arbeitsorganisatorische Unterstützung allein nicht genügt. So können wichtige Abstimmungsprozesse, die üblicherweise möglichst früh stattfinden, sich signifikant verzögern, weil erst Prototypen die Umsetzbarkeit einer softwaretechnischen Lösung erproben müssen. In dieser Situation alle relevanten Erkenntnisse adäquat an die Entwicklungsteams weiterzuleiten, ist keine triviale Aufgabe. So kommt es ebenfalls vor, daß Kooperationen bis zu späteren Iterationen eines Produkts ruhen, weil zunächst in anderen Teilbereichen konstruktive Verbesserungen umgesetzt werden müssen.

Diese zeitlich völlig unterschiedlichen Kooperations- und Kommunikationsbedürfnisse – zeitliche Kooperation im Team, zwischen verschiedenen Teams und über lange Entwicklungszeiträume hinweg – gilt es, durch entsprechend unterschiedliche Hilfsmittel zu unterstützen.

Sind Team-Mitglieder nur vorrübergehend oder kurzzeitig nicht zugegen, arbeiten aber am gleichen Ort, so ist das einfachste Hilfsmittel sicherlich die handschriftliche Notiz auf konventionellem Papier. Aufgrund der spezifischen Anforderungen der Softwareentwicklung wird dieses elementare Hilfsmittel sehr schnell von EMail-Systemen verdrängt. So kann man mit diesem nahezu universell einsetzbaren Allheilmittel nicht nur jeden beliebigen Dateityp zusammen mit der passenden Nachricht übermitteln, die EMail wird für den Empfänger aufbewahrt und ihm nch seiner Rückkehr direkt am Arbeitsplatz vorgelegt.

EMail erfreut sich unter Softwareentwicklern einer außerordentlichen Beliebtheit. Problematisch ist bei großen Projekten, die oft über Jahre andauern, sich in dem Sammelsurium von tausenden von EMails nicht zu verlieren. Nicht ohne Grund bewahren Entwickler einen Großteil ihrer EMails auf, vermittelt doch die Abfolge der Nachrichten ein verläßliches Bild des Entwicklungsprozesses. Gefragt sind Ablage- und Suchsysteme, die das Wiederfinden der gesuchten Informationen ohne großen Aufwand ermöglichen. Das schon zuvor beschriebene System der University of Colorado in Boulder ist eines der ersten (Fischer et al. 1996). Es stellt beispielsweise Suchmechanismen zur Verfügung, die Ähnlichkeitsbeziehungen zwischen den einzelnen EMails ausnutzen.

Während Voice-Mailboxen, d. h. elektronische, in die Telefonanlage integrierte Anrufbeantworter in größeren Betrieben durchaus zum Einsatz kommen, beispielsweise um Terminänderungen bekannt zu machen, sind Video-Mailboxen nur in Kombination mit Desktop-Videoconferencing sinnvoll. Ist ein Teilnehmer nicht erreichbar, so teilt man ihm eine Nachricht per Video mit. Solange technische Probleme wie Frame-Rate und Datenvolumen nicht gelöst sind (vgl. Foks 1996) und Desktop-Videoconferencing-Systeme keine weite Verbreitung gefunden haben, sind Video-Mailboxen ohne praktischen Wert.

Betrachtet man die phasenübergreifende zeitliche Kooperation in der Softwareentwicklung, so sind die Deliverables die am häufigsten zum Einsatz kommenden technischen Hilfsmittel. Sie vermitteln Inhalte an spätere Phasen bzw. sind für das Team die Grundlage der eigenen Arbeiten. Dies ist zumindest der Grund, aus dem man Deliverables eingeführt hat – oft dienen sie aber aus Fehlern im organisatorischen Gesamtkonzept als juristische Absicherung gegenüber dem Restprojekt. Die Folge ist, daß ihr Charakter viel zu rigide ist, um zeitliche Kooperation angemessen zu unterstützen und stattdessen von den Teams informelle Strukturen geschaffen werden.

Während Deliverables der Weitergabe von gefundenen Lösungen und erledigten Arbeiten dienen, werden für noch zu findende Lösungen und noch unerledigte Arbeiten „Problemdatenbanken" benötigt. Einträge in diesen Datenbanken beinhalten freien Text, sind aber in sich strukturiert. Problemdatenbanken mit ihrem technischen Charakter sind sowohl phasen- als auch iterationenübergreifend von Bedeutung. Systeme dieser Art benötigen unbedingt einen zeitlichen Wiedervorlagemechanismus, der sicherstellt, daß die entdeckten Fehler entweder zeitgerecht behoben oder über deren Nichtbehebung definiert entschieden wird, das betreffende Modul etwa nicht mit ausgeliefert wird. Ist der Aufwand abgeschätzt und wurde entschieden, daß der Fehler behoben wird, so muß ein systemgestützter Übergang auf das Projektplanungssystem erfolgen. Ist die Aufgabe erledigt, muß ein Feedback sowohl in das Projektplanungssystem als auch in die Problemdatenbank erfolgen. Aus dem Problem wurde ein Deliverable.

Ähnliches gilt für die im Jargon der Praxis als „Action Points" oder Aktionspunkte bezeichneten Entscheidungen auf über- bzw. vorgelagerten, nicht-technischen Ebenen. Diese Probleme werden in Entscheidungsagendas gesammelt und benennen beispielsweise einzelne Kundenwünsche oder Probleme mit der Infrastruktur. Auch bei dieser Klasse von Aufgaben kann eine Problemdatenbank weiterhelfen, wenn der Umgang mit diesen tendenziell nicht vollständig präzisierten und unstrukturierten Arbeitsaufgaben noch anspruchsvoller ist. Außer einer eher intuitiven Vorstellung von dem, was zu tun ist, einem möglichen Verantwortlichen und einem Eckdatum, bis zu dem das Problem gelöst sein muß, steht in der Regel wenig fest.

Verteilt sich die zeitliche Kooperation auf noch größere Zeiträume, so nimmt die Bedeutung der Kontextinformation im Vergleich zur eigentlichen Problemstellung immer mehr zu. Es wird immer aufwendiger, zu beschreiben, in welchem Zusammenhang das Problem auftrat und welche Lösungscharakteristika von Bedeutung sind. Insbesondere die Übermittlung von Informationen an zukünftige „Entwicklergenerationen" über Produkt, Prozeß und Metaprozeß wird immer

schwieriger. Dieses Problem der Wissensvermittlung bzw. -weitergabe ist in seiner Bedeutung nicht zu unterschätzen, da die Mitarbeiterfluktuation eine der wichtigsten Ursachen für Qualitätsdefizite bei der Softwareentwicklung darstellt. Vor dem Problem der Wissensweitergabe steht die Branche im Grunde relativ hilflos. Daß dieses Problem eher durch organisatorische, denn durch technische Hilfsmittel wie etwa eine „Wissensdatenbank" zu lösen ist, gehört hier zu den wenigen gemeinsamen Einschätzungen.

Wesentlich konkreterer Natur sind die Anforderungen, die man aus der zeitlichen Kooperation heraus an einheitliche Referenzierungssysteme stellt. Referenzierungssysteme werden benötigt, um sich eindeutig auf Dokumente, Software, Briefe, EMails, Telefonate, Problemberichte, Aktionspunkte, Entscheidungen und andere wichtige gegenständliche oder organisatorische Elemente der Softwareentwicklung beziehen zu können. Dieses gilt auch für den Austausch dieser Dokumente innerhalb bzw. außerhalb des Teams. Im Gegensatz zu anderen Branchen ist es bei der Softwareentwicklung nicht möglich, Arbeitsaufgaben mit Laufmappen oder ähnlichem zu übergeben, inklusiver aller relevanter Dokumente, da Dateien, Programme und Module imaginärer Natur sind. Fast alle Diskussionen im Entwicklungsprozeß beziehen sich aber auf diese oder jene Datei, dieses oder jenes Deliverable; ein Referenzierungssystem würde es erlauben, hier eindeutige Bezüge herzustellen.

Es existieren technische Hilfsmittel, die eine solche Referenzierung zu unterstützen versuchen, so etwa das BSCW-System als basales Hilfsmittel (Appelt / Hinrichs 1997) oder kommerzielle Configuration-Managementsysteme. Diese sind zwar im Ansatz tauglich und ausbaufähig, als problematisch erweisen sich in der Praxis vor allem die Brüche im Umgang mit verschiedenen Dokumenttypen – die einheitlichen Referenzierungssysteme sind dann längst nicht mehr so einheitlich, spätestens bei dem Umgang mit Action Points oder EMails. Ebenso unklar ist, wie es sich etwa mit der Mobilität der Team-Mitglieder verhält. Verlassen diese nämlich ihren Arbeitsplatz, so gehen sie buchstäblich „offline" und haben keinen Zugang mehr zum System. Das BSCW-System bietet durch das http-Protokoll immerhin die Möglichkeit, sich von Internet-Anschluß zu Internet-Anschluß zu bewegen.

Abschließend sei auf zwei weitere Problemstellungen hingewiesen, die allgemein zeitliche Kooperation unterstützen. So sollten alle Mitarbeiter Einblick in den für sie relevanten Projektstand haben. Weder die gültige Meilensteinplanung noch der Rückstand des realen Projekts gegenüber der Meilensteinplanung dürfen ausschließlich für das Management bestimmt sein. Ansonsten ist die Eigenkontrolle unnötig erschwert und eine frühzeitige Warnung vor Verzögerungen ist unmöglich.

In diesen Kontext gehört auch die Transparenz des Gesamtablaufs. Requirementsdatenbanken,

die die Anforderungen an das zu entwickelnde System beinhalten, sollten über Zeitmechanismen verfügen, die berücksichtigen, daß in bestimmten Iterationen bestimmte Anforderungen gelten, in anderen aber nicht, z. B. bei internen Iterationen, deren Resultate nicht an den Kunden weitergegeben werden. Nur wenn der zeitliche Gültigkeitsbereich von funktionalen und qualitativen Anforderungen jedem Beteiligten im Projekt klar ist, kann mit erwartungskonformen Ergebnissen zum festgelegten Zeitpunkt gerechnet werden.

5. Ausblick

Die Softwareentwicklungsbranche hat den Status einer Industrie erreicht: nicht weil deterministisch und im großen Maßstab ein Produkt geplant und produziert wird, nicht weil es etablierte Produktionsverfahren und -techniken gibt, die zum Einsatz kommen, sondern weil die Unternehmen dieser Branche die gleichen Probleme haben, wie jene der Automobilindustrie, des Maschinenbaus oder irgend eines anderen Produktionsbereichs – und weil die gleichen Fehler wie anderenorts gemacht werden. So glaubt man, durch straffe, hochgradig formale Organisation, durch Beseitigung informeller Strukturen und die gesamte Palette tayloristischer Arbeitsorganisation Produktivität, Effizienz und Effektivität steigern zu können. Man glaubt, durch immer aberwitzigere Zertifizierungen Qualität durch Qualitätskontrollen erreichen zu können, ohne zu verstehen, daß diese produziert und nicht geprüft werden will. Man glaubt, Kopf und Hand voneinander trennen zu können, und stellt dabei Kostenrechnungen auf, die an jene beim Import von Bananen erinnern.

Daß man in die gleichen Fallen wie andere Branchen läuft, ist allerdings nicht typisch für die Softwareentwicklung – diesen Fehler haben schon wesentlich etabliertere Industrien begangen. Die Unfähigkeit aber, aus diesem Sachverhalt die Konsequenzen zu ziehen, etwa über Jahrzehnte am Symptom zu laborieren, anstatt bekannte und in anderen Industrien erprobte Lösungskonzepte, wie etwa die der Gruppenarbeit, teilautonomen Gruppen und dezentralen Entscheidungsstrukturen, auf das eigene Handlungsfeld zu übertragen, hat schon eine spezifische Qualität. Sicherlich hat dazu auch beigetragen, daß die Kunden dieser Branche sich bis auf den heutigen Tag mehr oder weniger laut mit den Zähnen knirschend mit minderwertigen Resultaten zufrieden geben. Softwaresysteme sind trotz zahlreicher Pannen und echter Katastrophen offenbar noch immer „good enough" (Yourdon 1996) – es reicht, um die Ergebnisse auszuliefern, er reicht vor allem, um dafür kassieren zu können. Noch.

Will die Softwareentwicklung wirklich einen Weg aus ihrer hausgemachten Endloskrise finden, so tun ihre Vertreter gut daran, sich von ihrer Technikzentrierung zu verabschieden. Wer wie die Softwareentwickler tagtäglich mit Software und softwaretechnischen Unzulänglichkeiten zu

kämpfen hat, bei dem sollte es doch erstaunen, wenn er versucht, Softwareprobleme mit Software zu lösen. Ein Weg aus der Krise kann nicht gefunden werden, indem man sich auf die Entwicklung praxisuntauglicher Instrumente konzentriert. Vielmehr ist eine Orientierung an den Aufgaben und Problemen kooperativer Softwareentwicklung gefragt. Die Arbeitsaufgaben des Entwicklers im Team, die Aufgabe des Teams in einem Verbund von Teams und die Aufgabe des softwareentwickelnden Betriebs bei der Unterstützung der die Systeme anwendenden Organisationen sind die entscheidenden Maßstäbe für die Zukunft.

Sicherlich ist die Unterstützung der Entwickler durch geeignete Werkzeuge von zentraler Bedeutung, sind virtuelle Teams, die unabhängig von Zeit und Ort die komplexen Strukturen von Softwareentwicklungsprozessen überschauen und beherrschen, von adäquaten Infrastrukturen wie LANs, WANs oder funktionierender Hardware abhängig. Sicherlich tragen verteilte und kooperativ erzeugte und genutzte Informationsbestände für Requirements, Spezifikationen, Designinformationen, Teile des Produkts, Verifikation und Validierung entscheidend dazu bei, mit bezahlbarem Aufwand das zu entwickeln, was der Anwender für seine Organisation wünscht. Ohne einen adäquaten organisatorischen Gesamtrahmen, ohne die Loslösung von überkommenen statischen Strukturen und Verhaltensmustern, verpufft die Wirkung dieser Hilfsmittel, bleibt der Qualitätsgewinn auf der Strecke.

Die Entwicklung großer Softwaresysteme leidet unter eindeutigen Defiziten bei der kooperativen Arbeit. Ein Großteil der Probleme ist organisatorischer Natur und kann demzufolge nur durch organisatorische Maßnahmen gelöst werden. So trauen die Managementetagen der softwareentwickelnden Betriebe ihren Mitarbeitern offenbar genauso wenig, wie beispielsweise jene des Maschinenbaus oder der Automobilindustrie. Mit tayloristischen Methoden wird ein Dienst nach Vorschrift gefördert und gefordert, der die dringend benötigten kooperativ arbeitenden, flexiblen, sich selbst organisierenden und teilautonomen Entwicklerteams zunichte macht. Hier ist dringend Einhalt und Umkehr anzuraten; die Chancen dafür sind durchaus vorhanden, Ansatzpunkte zahlreich gegeben. So darf etwa die Distanz zum Kunden nicht systematisch maximiert werden: eine enge, über die ganze Projektlaufzeit praktizierte Kooperation zwischen Auftraggeber und Auftragnehmer ist kein Zeichen von Orientierungslosigkeit – vielmehr ist dies der erste Schritt aus den pseudoindustriellen Organisationskonzepten, in die sich die Softwareentwickler von Kostenüberschreitungen und defizitärer Termintreue haben treiben lassen. Die Schaffung einer Kommunikationskultur mit formellen und informellen Kommunikationsmöglichkeiten zur Kristallisation und Abstimmung von Entscheidungen muß dabei nicht von außen künstlich im softwareentwickelnden Betrieb implantiert werden; es gilt, diese noch vorhandene Kultur wieder freizulegen und zu beleben.

6. Literatur

[ApHi97] Appelt, W. / Hinrichs, E., 1997: Erstellung und Wartung von WWW-Seiten mit dem BSCW-System. EMISA-Forum (1 / 97).

[Boeh88] Boehm, B. W., 1988: A spiral model of software development and enhancement. IEEE Computer (21 / 5). 61-72.

[Broo87] Brooks, F. P., 1987: No silver bullet. Essence and accidents of software engineering. IEEE Computer (20 / 4). 10-19.

[Cle94] Cleland, D. I., 1994: Project Management. New York: McGraw-Hill.

[Cyra95] Cyranek, G. et al., 1995: Internationale Arbeitsteilung und globale Vernetzung: Intergration der Schwellen- und Entwicklungsländer in weltweite Netzwerke. Fachgespräch. In: Huber-Wäschle, F. et al., GISI 95 – Herausforderungen eines globalen Informationsverbundes für die Informatik. Berlin: Springer. 581-599.

[DuPa96] Dutke, S. / Paul, H., 1996: Privatheit, Gruppenhandeln und mentale Modelle: Schlüsselkonzepte multimedial unterstützter Arbeit. In: Brödner, P. et al. (Hg.), Kooperative Konstruktion und Entwicklung. München: Rainer Hampp. 147-181.

[Dut96] Dutke, S. et al., 1996: Privatheit, Gruppenhandeln und mentale Modelle: Psychologische Schlüsselkonzepte in der Gestaltung multimedial unterstützter kooperativer Arbeit. Graue Reihe des IAT (96 / 2). Gelsenkirchen: Institut Arbeit und Technik.

[Hum90] Humphrey, W. S., 1990: Managing the Software Process. Reading: Addison-Wesley.

[Fisch96] Fischer, G. et al., 1996: Informing System Design Through Organizational Learning. In: Edelson, D. C. / Domeshek, E. A., Proceedings of the International Conference on the Learning Sciences, July 1996, Evanston, IL. Charlottesville: Association for the Advancement of Computers in Education (AACE). 52-59.

[Foks96] Foks, T., 1996: Telekooperation – Stand der Dinge 1995. Graue Reihe des IAT (96 / 3). Gelsenkirchen: Institut Arbeit und Technik.

[Neu95] Neumann, P. G., 1995: Computer-Related Risks. Reading: ACM Press / Addison Wesley.

[PoBl93] Pomberger, G. / Blaschek, G., 1993: Software Engineering. München: Carl Hanser.

[Rop79] Ropohl, G., 1979: Eine Systemtheorie der Technik. München: Carl Hanser.

[Roth95] Roth, C., 1995: Software-Werkzeuge zwischen Werkstatt- und Fabrikmetapher. Dissertation. Bremen: Universität Bremen, Fachbereich Mathematik und Informatik.

[Roy70] Royce, W. W., 1970: Managing the development of large software systems: concepts and techniques. In: Riddle, W. E., IEEE Computer Society – Proceedings of the 9th International Conference.

[Somm95] Sommerville, I., 1995: Software Engineering. Wokingham: Addison-Wesley.

[Spr96] Springer, J. et al., 1996: Persönliche Kommunikation und Telekooperation – Anforderungen an telekooperative CAD-Systeme. In: Brödner, P. et al. (Hg.), Kooperative Konstruktion und Entwicklung. München: Rainer Hampp. 183-214.

[Your96] Yourdon, E., 1996: Rise and Resurrection of the American Programmer. Englewood Cliffs: Edward Yourdon Press.

GABLER EDITION WISSENSCHAFT

Information Engineering und
IV-Controlling

Herausgegeben von Professor Dr. Franz Lehner

Die Schriftenreihe präsentiert aktuelle Forschungsergebnisse der Wirtschaftsinformatik sowie interdisziplinäre Ansätze aus Informatik und Betriebswirtschaftslehre. Ein zentrales Anliegen ist dabei die Pflege der Verbindung zwischen Theorie und Praxis durch eine anwendungsorientierte Darstellung sowie durch die Aktualität der Beiträge. Mit der inhaltlichen Orientierung an Fragen des Information Engineerings und des IV-Controllings soll insbesondere ein Beitrag zur theoretischen Fundierung und Weiterentwicklung eines wichtigen Teilbereichs der Wirtschaftsinformatik geleistet werden.

Franz Lehner/Schahram Dustdar (Hrsg.)

Telekooperation in Unternehmen

DeutscherUniversitätsVerlag

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Telekooperation in Unternehmen / Hrsg.: Franz Lehner ;
Schahram Dustdar. - Wiesbaden : Dt. Univ.-Verl. ; Wiesbaden : Gabler, 1997
(Gabler Edition Wissenschaft : Information Engineering und IV-Controlling)
ISBN 3-8244-6433-0

Der Deutsche Universitäts-Verlag und der Gabler Verlag sind Unternehmen der Bertelsmann Fachinformation.

Gabler Verlag, Deutscher Universitäts-Verlag, Wiesbaden

Lektorat: Claudia Splittgerber

Druck und Buchbinder: Strauss Offsetdruck, Mörlenbach
Printed in Germany

ISBN 3-8244-6433-0