Inhaltsverzeichnis

Liebe Leserinnen und Leser,

die sowohl von Wissenschaftlern als auch Praktikern übereinstimmend konstatierte Softwarekrise führte in der Informatik zur Entstehung des Fachgebietes Software Engineering. Die gleichnamige Konferenz 1968 in Garmisch wird als Gründungsereignis für diesen Forschungszweig angesehen. Die methodischen und organisatorischen Probleme bei der Durchführung großer und komplexer Softwareprojekte waren damals für alle an der Konferenz Beteiligten nachvollziehbar. Das Ziel der Forschungen auf diesem Gebiet sollte deshalb die Entwicklung von geeigneten Methoden und Werkzeugen zur geplanten und kontrollierten Entwicklung von Software - auch in größerem Maßstab - sein. Allerdings bestand zwischen den Teilnehmern keine Einigkeit darüber, wie dieses Ziel zu erreichen sei, auf welche Art und Weise hilfreiche Erkenntnisse über den Softwareentwicklungsprozess und Software entstehen, und wie brauchbare Methoden und Werkzeuge entwickelt werden können.

Heute gibt eine Vielzahl von Methoden und Werkzeugen. Die einen zielen auf die Verbesserung der technischen Parameter von Software ab, andere haben den Ablauf der Entwicklung, den Prozess der Herstellung von Software im Blick. Es gibt jedoch eine beachtenswerte Anzahl von empirischen Untersuchungen, die belegen, dass viele dieser Methoden und Werkzeuge in der Praxis gar nicht erst zum Einsatz kommen. Die Vorstellungen der Wissenschaftler und Praktiker darüber, was im Alltag eines Softwareunternehmens notwendig und brauchbar ist, scheinen nicht überein zu stimmen. Die Vorstellungen, wie Entwicklungsprozesse verbessert und erleichtert werden können, und wie gute und brauchbare Software entsteht, sind offenbar im Wissenschaftsbereich andere als im Entwicklungsalltag.

Im Rahmen eines empirischen Forschungsprojekts[1] an der Fakultät für Informatik der TU Chemnitz haben die OrganisatorInnen des in diesem Heft dokumentierten Workshops diese Differenzen zum Anlass genommen, Austauschprozesse zwischen Wissenschaft und Praxis in Gang zu bringen, um Unterschiede in den Sichtweisen und Bewertungen zwischen und innerhalb dieser Personengruppen sichtbar und diskutierbar zu machen. Mit diesem Ziel veranstalteten sie am 31. Oktober und 1. November 2001 in Chemnitz einen Workshop mit dem Titel „Software Engineering meets Practice“. Auf dem Workshop stellten empirisch arbeitende WissenschaftlerInnen ihre Ergebnisse im Hinblick auf Softwareunternehmen, auf Softwareentwickler und auf den Softwareentwicklungsprozess vor. Praktiker kommentierten die Ergebnisse aus Sicht ihres Alltages: Sie stellten ihren Standpunkt zu den Themen vor dem Hintergrund ihrer Erfahrungen in der Branche - als Entwickler, Projektleiter oder Geschäftsführer - dar. Am Abend des ersten Tages formulierten die Praktiker ihre Anforderungen an die Informatik bzw. ein hilfreiches Software Engineering. Ein ehemaliger Student fügte diesen Wünschen konkrete Vorschläge zur Verbesserung des Chemnitzer Informatik-Studiums hinzu. Am Ende des zweiten Veranstaltungstages entwarfen zwei VertreterInnen des Fachgebietes Software Engineering ihre Erklärungsmodelle im Hinblick auf die vorhandenen Differenzen zwischen Theorie und Praxis.

Wir freuen uns, dass wir Ihnen in diesem Heft die Mehrzahl der Workshop-Beiträge präsentieren können. Die Praktiker nahmen in ihren Kommentaren teilweise nur auf ausgewählte Argumente aus den wissenschaftlichen Beiträgen Bezug. Das liegt zum Teil in der Entstehungsgeschichte der

[1] Dieses Projekt „Software entwickeln in der Praxis im Kulturvergleich“ wurde im Rahmen der Chemnitzer Forschergruppe “Neue Medien im Alltag: Von individueller Nutzung zu soziokulturellem Wandel” (Bo 929/13-1) gefördert. Werner Dilger war der Leiter des Projektes, Annette Henninger und Andrea Sieber waren Mitarbeiterinnen, Yvonne Magwas und Martin Söns waren als studentische MitarbeiterInnen beteiligt.

Beiträge begründet. Die WissenschaftlerInnen stellten ihre Beiträge im Vorfeld den Praktikern zur Verfügung. Der Vortrag auf dem Workshop bezog sich jedoch nicht zwangsläufig auf alle Elemente des hier abgedruckten Textes bzw. Argumente wurden im Vortrag auf Nachfrage hervorgehoben, die im Text nicht so prominent erscheinen. Demgegenüber sind die Beiträge der Praktiker aus den Transkripten der von ihnen auf dem Workshop gehaltenen Vorträge entstanden. Sie spiegeln also ausschließlich das wieder, was auf dem Workshop thematisiert wurde. Nichtsdestotrotz sind wir der Meinung, dass dadurch zumindest Teile der Diskussion während des Workshops auch für diejenigen nachvollziehbar werden, die an der Veranstaltung selbst nicht teilnehmen konnten.

Gerd Paul aus dem Sozialwissenschaftlichen Institut in Göttingen eröffnete den Diskurs aus industriesoziologischer Perspektive. Er legte auf der Basis seiner empirischen Untersuchungen dar, inwieweit virtuelle Organisationsformen für Kleinunternehmen in der Softwarebranche von Bedeutung sein können. Günter Tröger, Geschäftsführer der Systemhaus Chemnitz GmbH, formulierte in seinem Kommentar Fragen nach den Erfolgsfaktoren für den Bestand kleiner Unternehmen. Er verwies auf rechtliche Unsicherheiten bei der Zusammenarbeit in virtuellen Netzwerken und legte aufgrund seiner Erfahrungen dar, welche Rahmenbedingungen für ihn erfüllt sein müssen, damit er mit anderen Firmen kooperiert.

Sabine Sonnentag vom Institut für Psychologie in Braunschweig stellte in ihrem Beitrag Merkmale und Vorgehensweisen von besonders leistungsstarken Softwareentwicklern vor. Sie wurden aus mehreren empirischen Studien abgeleitet und verdeutlichen den Zusammenhang zwischen Kooperation, Kommunikation und Leistungsstärke. Frank Stockmann, Vorstand und Projektleiter der IVS-Solutions AG in Chemnitz, interpretierte die Ergebnisse anders. Er war der Meinung, dass leistungsstarke Entwickler nur kommunizieren und kooperieren, weil sie es aufgrund der Nachfrage von Kollegen und der Einbindung in das Unternehmen müssen. Er zeigte zugleich die Problematik der Zusammenarbeit von mehreren leistungsstarken Personen in einem Team auf.

Die Softwareentwickler ab 40 standen im Mittelpunkt eines Projektes aus dem Forschungsverbund zum demografischen Wandel. Uwe Lünstroth vom Lehrstuhl Technikphilosophie in Cottbus zeigte gängige Vorurteile gegen ältere Softwareentwickler auf und hob im Gegensatz dazu ihre spezifischen Qualifikationen und Erfahrungen hervor, die insbesondere bei der Entwicklung kundenorientierter Technik von Vorteil seien. Wolfgang Häcker, Offering Leiter der IT-Services & Solutions GmbH wies anhand seiner Alltagserfahrungen nach, dass die Einbindung älterer Softwareentwickler nicht nur moralisch geboten ist, sondern aufgrund der Fortexistenz von älteren Softwaresystemen lukrative Marktnischen in der Wartung und Pflege solcher Systeme eröffnet. In seinem Unternehmen werden aufgrund der großen Nachfrage nach solchen Dienstleistungen aktuell sogar Interessierte auf älteren technologischen Standards ausgebildet.

Mit einer anwendungsorientierten Form der Methodenentwicklung beschäftigte sich Yvonne Dittrich vom Blekinge Institute of Technology in Ronneby (Schweden). Sie berichtet von verschiedenen Projekten, in denen Werkzeuge für die Softwareentwickler mit ihnen gemeinsam angepasst wurden. Friedrich Strauss, Technischer Chefdesigner bei der Firma sd&m in München beschreibt, welche Bedingungen neue Methoden in seinem Unternehmen erfüllen müssen, um überhaupt eingesetzt zu werden. Er ist der Meinung, dass wissenschaftliche Neuentwicklungen auf methodischem Gebiet durchaus in der Praxis eine Rolle spielen. Sie sind jedoch nicht in Reinform zu finden und halten oft Einzug aufgrund bisheriger verbesserungswürdiger Arbeitserfahrungen.

Der Beitrag von Katharina Bluhm vom Lehrstuhl für Arbeits-, Industrie- und Wirtschaftssoziologie in Jena zur Internationalisierung im Mittelstand und der Beitrag von Teade Punter vom Fraunhofer Institut für Experimentelles Software Engineering zur Veröffentlichung von Erfah-

rungen mit Methoden und Werkzeugen im Internet können aus forschungsstrategischen Überlegungen in diesen Projekten leider nicht veröffentlicht werden. Da beim Fehlen des wissenschaftlichen Beitrages die Ausführungen der Praktiker ohne Bezug bleiben würden, haben wir uns entschlossen, auch den Beitrag von Stefan Kahlert, Gruppenleiter Entwicklung bei der T-Systems, Debis Systemhaus GEI in Chemnitz nicht mit in die Dokumentation aufzunehmen. Wir bitten dafür um Verständnis.

Die letzten vier Beiträge dieses Bandes dienten im Workshop dazu, die Debatte um die Widersprüche zwischen Theorie und Praxis noch einmal auf einer abstrakteren Ebene und aus anderen Perspektiven weiterzuführen. Der Beitrag von Andrea Sieber und Annette Henninger versucht, die Wünsche der Praktiker an die Informatik im allgemeinen und das Software Engineering im speziellen zusammenzufassen, die im Abschlussgespräch am 31.Oktober geäußert wurden. Sie ergänzen dabei die in der Diskussion formulierten Anforderungen durch Ergebnisse ihrer empirischen Untersuchungen. René Stöckel als ehemaliger Student der Informatik kann diese Anforderungen durch konkrete Gestaltungsvorschläge für das Informatik-Studium in Chemnitz bereichern. Der Beitrag von Petr Kroha, Lehrstuhl Informationssysteme und Softwaretechnik in Chemnitz zeigt auf, dass wir genaugenommen gar nicht von einer Theorie im Software Engineering sprechen können, und dass die Methoden und Werkzeuge, die in diesem Gebiet entwikkelt werden, letztendlich nur dann Bestand haben, wenn die Unternehmen damit Geld verdienen können. Christiane Floyd vom Lehrstuhl Softwaretechnik in Hamburg bezweifelt die allen anderen Zielen übergeordnete Priorität der ökonomischen Logik und zeigt auf, warum die Entwicklung von brauchbarer Software und brauchbaren Methoden und Werkzeugen so ein schwieriges Unterfangen ist. Die beiden letztgenannten Beiträge bildeten den Abschluss der Veranstaltung.

Wir möchten uns noch einmal ganz herzlich bei allen ReferentInnen, TeilnehmerInnen und OrganisatorInnen für ihr Mitwirken am Gelingen dieser Veranstaltung bedanken. Ein besonderer Dank gilt der Forschergruppe „Neue Medien im Alltag“ an der TU Chemnitz, ohne deren finanzielle Unterstützung die Durchführung des Workshops in dieser Form nicht möglich gewesen wäre. Ebenso geht ein herzliches Dankeschön an die Studierenden der Angewandten Informatik mit Spezialisierung Medieninformatik für die Vorführungen ihrer Seminararbeiten sowie an unsere Mitarbeiterin Annette Henninger und unsere studentischen MitarbeiterInnen Yvonne Magwas und Martin Söns für die Unterstützung bei der Organisation des Workshops und der Erstellung dieser Dokumentation. Möge der interdisziplinäre Austausch zwischen Theorie und Praxis weitere Fortsetzungen finden.

Andrea Sieber & Werner Dilger — Chemnitz, Oktober/November 2001

Hightech Kleinunternehmen auf dem Weg in die Virtualität?

Gerd Paul
Universität Göttingen
Sozialwissenschaftliches Forschungsinstitut
Friedländer Weg 31
37085 Göttingen
gpaul@gwdg.de

Ich wurde gebeten, Ihnen über meine Arbeit in einem aktuellen Projekt über Telekooperation in den Bereichen Multimedia, Softwareentwicklung und Internetservices zu berichten. Ich werde es unter dem Blickwinkel meiner wissenschaftlichen Disziplin – der Industrie- und Techniksoziologie – tun und mich dem Thema in mehreren Schritten annähern, indem ich a) auf Hintergründe der Diskussion um Virtualisierung eingehe, b) eine Reihe von Problemen benenne, die mit dem Modell des virtuellen Unternehmens zu tun haben und Ihnen c) aus der eigenen aktuellen Forschung über Firmen im Multimedia-, Software- und Internetsektor einige Einblicke in die Arbeitsrealität der postmodernen New Economy gebe.

1 Hintergründe der Diskussion um die Virtualisierung

Den Hintergrund meiner Ausführungen bildet die Rede von der knowledge economy. Es geht, wie man postmodern sagen würde, um die „große Erzählung" eines gänzlich anderen, wissensbasierten Kapitalismus, in dem Unternehmen und Arbeit nicht durch materielle, sondern durch immaterielle Eigenschaften ihren Wert gewinnen. Erst durch den Siegeszug der Informationstechnologien in den 80er und dann zunehmend in den 90er Jahren in Büros und Fabriken ist eine informationstechnologiegestützte Rationalisierung und Reorganisation möglich geworden, die die Außen- und Binnenbeziehungen der Unternehmen neu ordnete und zugleich verflüssigte. Dies betrifft nach außen die Beziehungen zu Kunden, Geschäftspartnern und Lieferanten und nach innen alle Tendenzen einer Abkehr vom hierarchischen, fordistischen Organisationsmodell mit seinen vertikalen Abteilungen, den festgelegten Laufbahnen, den festen Aufgabengebieten, Arbeitszeiten usw. Lean Management, Business Process-Reenginering, „Generierung von Wettbewerbsstrategien" (Porter), Dezentralisierung, Errichtung von Profitzentren und andere magische Worte für umfassende Veränderung aus dem Vokabular der Managementberater stehen für eine neue, flexible Organisation von Produktion und Dienstleistung, wobei sich vielfach starre Unternehmensgrenzen zugunsten von Allianzen und Netzwerken auflösen. In diesem Zusammenhang gewinnt das Wort von der „virtuellen Organisation" an Bedeutung, auf das ich gleich noch einmal eingehen werde, denn es hat vielfältige Bedeutungen und ist - wie alle Modewörter und Visionen - selber modischen Veränderungen unterworfen.

Der Mythos eines ganz anderen Kapitalismus bekommt seine Kraft nicht nur durch eine schöne Vision. Er braucht Protagonisten, Helden, die Unmögliches möglich machen und - obgleich durch vielfältige Schicksalsschläge stets vom Scheitern bedroht - sich gegen die bestehende Ordnung der Dinge durchsetzen. Heutige mythologische Gestalten sind Bill Gates oder der deutsche Intershop-Gründer Stephan Schambach. Es sind außerordentliche Menschen, die die technologi-

schen Möglichkeiten mit ihrer Geschäftsvision profitabel verbunden haben und aus dem Nichts gestartet sind. Was zählt sind Ideen, Brainware und der individuelle Wille sich durchzusetzen. So lautet die Message, die der Mythos transportiert. Und er enthält eine zweite Message: Jeder kann es nachmachen, ihm müssen nicht mehr die Götter gewogen sein, sondern er muss nur entweder die nächste technologische Killerapplikation entwickeln – oder etwas bescheidener – durch Firmengründung in der Internetökonomie mit ihren unendlichen Möglichkeiten zu Reichtum und Anerkennung gelangen. Dass dies nun doch nicht jedem gegeben ist, sondern der Reichtums-Mehrung einer Reihe von Filtern wie Herkunft, Geschlecht, Interesse vorgeschaltet sind, wird Sie nicht überraschen. Aus meiner eigenen Beschäftigung mit Softwareentwicklung und neuen Berufen weiß ich, dass trotz einer Vielzahl von nicht institutionalisierten Wegen zum DV-Fachmann oder Programmierer in der Regel das Abitur die Voraussetzung für eine solche Arbeit darstellt, und sehr oft auch ein naturwissenschaftliches oder ingenieurwissenschaftliches Studium. Und auch sicher nicht alle der 47 Prozent deutscher Jugendlicher, die laut Shell Studie „wahrscheinlich" oder „sicher" Unternehmer werden wollen (Shell 2000, Vol. 1, S. 50), diejenigen, die dann ein Hightech Start-up Unternehmen in jungen Jahren aufmachen. Die dafür in Frage Kommenden sind eher im Umkreis der kleinen „Heavy User" Gruppe der jungen, eher männlichen und gut ausgebildeten Computerbesitzer, die auch sonst technologisch sehr gut ausgestattet sind, zu finden (Glott/Paul 2001).

Das Silicon Valley Modell ist nur sehr begrenzt auf Deutschland übertragbar. Diverse Beiträge in Lee et al. (2000) zur sozialen und ökonomischen Bedeutung des Silicon Valley machen plausibel, dass es eine ganze Reihe von sich gegenseitig verstärkenden Faktoren sind, die die Einzigartigkeit des Silicon Valley Modells bekräftigen. Dies sind die rechtlichen und staatlichen Rahmenbedingungen, die Finanzierungsmöglichkeiten für Start-ups, der ständige Vergleich zu anderen erfolgreichen innovativen Unternehmen der Region, eine relativ breite Basis gemeinsam geteilten Wissens innerhalb einer kooperationsfreudigen Business Community, die Existenz einer breiten Schicht hochqualifizierter, mobiler Arbeitnehmer, die Nähe zu exzellenten Universitäten und vielfacher Wissenstransfer zwischen staatlichen und privaten Akteuren, eine ausgezeichnete Serviceinfrastruktur, eine hohe Lebensqualität, ein soziales Klima, das Risiko belohnt und Scheitern toleriert und in dem auch neu eingewanderte Spezialisten eine Chance haben. Informalität und „fun" bei der Arbeit, die start-up Entrepreneurkultur und die Möglichkeit, an „cutting edge" Technologien zu arbeiten, wirken wie ein Magnet auf Scharen von jungen, ehrgeizigen Hightech Enthusiasten[1]. Lee (2000) argumentiert, dass der in der Regel junge und erfolgshungrige „Silicon Valley Unternehmer" sich sowohl vom traditionellen Kleinunternehmer als auch vom am Markt etablierten Unternehmer unterscheidet, etwa durch hohe Risikobereitschaft, die Zentralität von Schnelligkeit, starke Marketingaktivitäten von Anfang an, eine Internationalisierungs- und Marktführerschaftsstrategie, die Konzentration auf neue, revolutionäre Technologien, die durch kontinuierlich hohe F&E Aufwendungen hervorgebracht werden, eine flexible, hierarchiearme Organisation mit einem hochqualifizierten, extrem motivierten Team technologisch versierter,

[1] Bronson, der die Kultur des Silicon Valleys nachzeichnet, beschreibt die Motive der Newcomer so: „They just show up. They've given up their lives elsewhere to come here. They come for the tremendous opportunity, believing that in no other place in the world right now can one person accomplish with talent, initiative, and a good idea. It's a region where who you know and how much money you have, have never been relevant to success. They come because it does not matter that they are young or left college without a degree or have dark skin or speak with an accent. They come even if its illegal to do so. They come because they feel that they will regret it the rest of their lives if they do not at least give it a try. They come to be a part of history, to build the technology that will reshape how people will live and work five or ten years from now. They come for the excitement, just to be part of it. They come because they are competitive by instinct and can't stand to see others succeed more than they. They come to make enough money so they will never have to think about money again." (Bronson 1999, S. 3).

mobiler Spezialisten mit einer gemeinsam geteilten Vision und einem weiten Netzwerk an Zulieferern und Freelancern. Dieser Turbo Start-up Typ ist auch in Amerika in der Minderheit. „Most start-ups derive from individuals seeking self-employment, rather than conduct of an entrepreneurial effort to develop new products, markets, technologies, and so on. ... The typical business apparently starts small and stays small", kommentiert Bhidé (2000, S 12), der eine umfassende Längsschnittstudie über die Entwicklung neuer Firmen gemacht hat. Diese Aussagen gelten auch für Deutschland und sogar auch für viele Firmen der Internet-Branche, besonders die große Mehrheit derer, die ohne Beteiligungskapital arbeiten (siehe z.B. die Ergebnisse der Erhebungen von Klandt/Kraft aus dem Jahr 2000, www.e-startup.org).

Webdesigner oder Javaprogrammierer haben ja in diesem Sinne mit Sachen, die nicht real sind, aber als Möglichkeit existieren, zu tun, mit „virtuellen" Produkten, also mit medialen Repräsentationen, die sie sich erst ausdenken und dann in Webseiten oder Hilfsprogramme umsetzen und die jederzeit am Computerbildschirm veränderbar sind. Hier setzt schon eine engere Definition des Virtualitätsbegriffes an: „physisch nicht existent, aber durch Software geschaffen". Das mit CAD konstruierte Automodell, der digitale Produktprototype des Markenartikels sind Objekte, die durch Medienwechsel im Computer anders repräsentiert werden und jederzeit zu einer Vielzahl ähnlicher, aber anderer Produkte transformiert werden können. Virtualität, so die Wirtschaftswissenschaftler Littmann und Jansen in ihrem Buch „Oszillodox" ist die „Einheit von Aktualität und Potentialität." Sie fahren fort „Die Virtualität hält demnach immer mehr bereit, als derzeit aktualisierbar ist. Das Virtuelle kann aber nur das Potentielle bereithalten, das durch Experimentieren der Sinne wahrnehmbar ist. Es ist nur das potentiell, was als Potential gedacht werden kann." (Littman/Jansen 2000, S. 35).

Der Virtualitätsbegriff kam ursprünglich aus der Informatik, wo es um die Auslagerung von Datenblöcken im Computer bzw. die Simulation eines zusätzlichen, virtuellen Speichers ging. Mit Howard Rheingolds Buch „Virtual reality" aus dem Jahre 1991 war eine breite Diskussion eröffnet, wie sich dreidimensionale Objekte im Computer simulieren und mit Datenhandschuh und Videohelm erfahrbar gemacht werden können. Mitte der 90er Jahre, mit der Ausbreitung des Internets, wurde der Virtualitätsbegriff ausgeweitet. Es gab seit Mitte der 90er das Internet, und mit ihm eine Ausweitung des Attributs „virtuell", das zum Beispiel auf den e-commerce mit seinen virtuellen Märkten und virtuellen Produkten angewandt wurde, etwa der auf individuelle Wünsche und Interessen abgestellte, per E-Mail versandte Newsletter. In bezug auf das virtuelle Unternehmen gab es eine ähnliche, zunächst eher begrenzte, dann sich erweiternde und ausufernde Begriffsverwendung. Zunächst ging es um die Übertragung der Metapher der virtuellen Speicherverwaltung auf Wirtschaftsorganisationen, etwa in der Vorstellung, virtuelle Unternehmen seien solche, die zu einer „elektronischen Wirtschaftseinheit" (Littman/Jansen 2000, S. 44) verschmelzen. Anfang der 90er Jahre popularisierten William Davidow und Michael Malone die Idee des virtuellen Unternehmens, indem sie - wie viele nach ihnen - von einem temporären Netzwerk unabhängiger Unternehmen ausgingen, die durch Informationstechnologien verbunden, ihre Fähigkeiten und Marktzugänge gemeinsam nutzen, und die zu dem gemeinsamen Unternehmen ihre Kernkompetenz beisteuern. Bereits darin klangen schon die Vorteile der Virtualität an, dass diese Unternehmen nicht mehr an einem Standort räumlich integriert sind, sondern sich in virtuellen Räumen bewegen, dass sie durch zeitliche Asynchronisation rund um die Uhr arbeiten können und entsprechend der Ablaufstruktur Ressourcen flexibel zugeteilt werden können.

Unternehmen, die Telearbeit oder virtuelle Büros anboten, nutzten diese Vorteile. Auch der virtuelle Handel, der wie AMAZON mit Büchern handelt oder die Advance Bank (Electronic Finance) sind Beispiele für eine intraorganisationale Virtualisierung, bei der es um eine multime-

dia-gestützte Modularisierung von Wertschöpfungssegmenten und eine zeitliche Asynchronisation und Delokalisierung von Geschäftsprozessen geht.

Bereits schon in den 80er Jahren wurde mit den Stichworten „industrial districts" und „das zweite Italien" das Augenmerk der Soziologen auf regionale Netzwerke, zum Beispiel in der Emilia Romagna oder in der Textilindustrie um Prato gerichtet. Hier, oder auch bei Entwicklungsnetzwerken (Linux User Group) handelt es sich um interorganisationale Virtualisierung, also eine zeitlich begrenzte, nicht institutionalisierte Kooperation rechtlich unabhängiger Unternehmen. Auf eine zentrale Steuerung wird verzichtet.

Ein anderer, kundenbezogener Typ virtueller Organisation ist der, bei dem eine Produktindividualisierung durch den Kunden vorgenommen werden kann, der sich seine eigenen Brillen (Paris Miki), Schuhe (Custom Foot) oder Jeans (Levis) entwirft und diese individuellen Produktionsparameter dann über Internet an den Hersteller gehen.

Praktische Beispiele einer „hollow organisation", wie der Getränkehersteller ohne Produktion „Red Bull" oder die Spielzeughersteller „Toys are us" oder „Galoop toys", die mit einem sehr geringem Personal im Hauptquartier letztlich nur noch die externe Wertschöpfungskette optimierten bzw. die Verteilung der verschiedenen Entwurfs-, Marketing-, Produktions- und Logistikfunktionen an Subauftragnehmer informationell organisierten, popularisierten den Virtualisierungsgedanken. Diese Steuermänner, die fokalen Unternehmen, die externe Wertschöpfungsnetzwerke zentral koordinierten, aber noch Formen institutionalisierter Kooperation eingingen, haben die Industriesoziologie in den 90er Jahren stark beschäftigt, weil sie sozusagen Sinnbilder der Globalisierung waren, man denke nur an Nike Sportschuhe oder den US Fashion Shop GAP, die ihre Produkte zu Minimallöhnen in Südostasien produzieren lassen. Die Soziologen bedachten das fokale Kernkompetenznetzwerk mit hohem Oursourcing-Anteil mit Stichworten wie „Arbeit an der Kette" und wiesen dabei auf Machtasymmetrien zwischen beherrschendem Fokalunternehmen und abhängigen Zulieferern hin. Das Interesse der Industriesoziologen galt dem „extraorganisationalen Typen" virtueller Organisation, wobei das organisierende Unternehmen sozusagen fest bleibt, aber die Außenbeziehungen dynamisiert, meistens um die eigene Wertschöpfung zu reduzieren oder zeitlich begrenzte Leistungen einzukaufen. Sie kommen meistens mit einer geringen Zahl von Stamm-Mitarbeitern aus und greifen bei Bedarf auf freie Mitarbeiter, die sogenannten Freelancer zurück. Dazu gehören auch Markenunternehmen wie der Sportschuhhersteller Nike, die als Broker externer Produktionskapazitäten auftreten, temporäre Massenproduktionsnetzwerke, wie bei der SMART Herstellung, Forschungskooperationen u.a.

Mit den oben vorgestellten vier Grundtypen einer virtuellen Organisation, der intraorganisationalen, der interorganisationellen, der kundenbezogenen und der extraorganisationalen Virtualisierung ist nur ein klassifikatorischer Rahmen für die ständig zunehmenden Felder einer internetgestützten Tätigkeit benannt. Was die Phantasie anregt und ganz neue Einsatzmöglichkeiten ausdenken lässt, ist genau das Potentielle, das technisch Mögliche des Internets. Vorstellungen des virtuellen Klassenzimmers, der virtuellen Hochschule, der virtuellen Beratung, des virtuellen Arztbesuchs schöpfen ihre Überzeugungskraft nicht nur aus Kostengesichtspunkten, der Möglichkeit, qualifizierte Dienstleistungen weltweit zu erstellen, zu transportieren, zu verkaufen und aus anderen rationalen Erwägungen. Sehr wahrscheinlich wird mit dem Attribut „cyber" oder „virtuell" der Mythos der Maschine in anderer Form wiederbelebt, also die Vorstellung, mit maschineller Regelmäßigkeit, Eindeutigkeit, Verlässlichkeit die chaotische Realität in den Griff zu bekommen (vgl. Bamme et al. 1983).

2 Probleme des virtuellen Unternehmens

Wie kommt es nun, dass es immer dieselben Unternehmen sind, die in den Veröffentlichungen über virtuelle Unternehmen auftauchen, und warum kommen Wirtschaftswissenschaftler zu dem Schluss, dass „aus der Unternehmenspraxis bisher nur wenig erfolgreiche Netzwerkprojekte bekannt sind“ (Beyhs/Hirsch 1999)? Sind es pure Akzeptanzprobleme unmoderner Unternehmer? Wohl kaum. Es sind eine Reihe von ökonomischen und strategischen Kalkülen, die sich auf die Gefahren des virtuellen Unternehmens beziehen.

Zunächst ist nicht jedes Unternehmen gezwungen, virtuell zu werden, das heißt in diesem Zusammenhang, einen Teil der Aktivitäten auf das Internet zu verlagern. Der Kioskbesitzer um die Ecke hat keinen Zugewinn, wenn er der ganzen Welt auf seiner Homepage seine überall erhältlichen Waren anbietet. Wie wir aus der Vernetzungsdiskussion wissen, sind es nur bestimmte innovative wirtschaftliche Aktivitäten, die strategisch auf Wertschöpfungsoptimierung zielen, die Unternehmen dazu bringen, auf selbststeuernder Projektorganisation mit den Partnern zu setzen und freiwillig auf traditionelle Kontroll- und Koordinationsmodi zu verzichten.

Die Errichtung eines neuen Vertriebskanals wie das 24-Stunden-Banking via Internet kostet erst einmal sehr viel Geld und ist eine Investition in eine ungewisse Zukunft. Fast keine deutsche Internet Bank arbeitet zum Beispiel profitabel. Es ist auch nicht gesichert, dass neue Kundensegmente gewonnen werden können, oder ob bloß bereits bestehende Kunden auf das neue Medium umsteigen. Zudem müssen bei einer solchen intraorganisationellen Virtualisierung Kommunikations- und Entscheidungsstrukturen, Informations- und Wissensmanagementsysteme und auch die Kultur und strategische Ausrichtung integriert werden, was keine leichte organisatorische Aufgabe ist.

Bei den Organisatoren strategischer Netzwerke sind es mehrere Probleme, die eine deutliche Zurückhaltung vieler Unternehmer hervorrufen: Zum einen kann man die Zulieferer der atomisierten Wertschöpfungskette, die alle hochspezialisiert sind und Weltmarktstandards setzen, nicht beliebig wechseln, man denke nur an Bosch, den Produzenten von Komponenten für Einspritzpumpen für Dieselmotoren. Mit der Länge der Kette erhöht sich die Abhängigkeit. Zum anderen besteht die Gefahr eines immer weiteren Abgebens von Kompetenzen an externe Unternehmen darin, dass irgendwann die eigene Kernkompetenz ausgehöhlt ist, und damit substituierbar oder imitierbar wird. Mit einer immer kleiner werdenden Stammbelegschaft und einem Kranz von im Bedarfsfall herangezogenen Mitarbeitern auf Werkvertragsbasis wird der Wert jedes Stamm-Mitarbeiters erhöht, und das Unternehmen muss sich etwas einfallen lassen, damit dieses wertvolle Gut ihm nicht den Rücken kehrt.

3 Arbeitsrealität der post-modernen New Economy

Überhaupt ist eine Kooperation mehrerer Firmen in einem virtuellen Unternehmen sehr fragil. Das wissen wir aus unserer eigenen Forschung über Telekooperation, d.h. über eine zeitweilige Zusammenarbeit lokal unterschiedlich verteilter Firmen, die als Medium der Zusammenarbeit das Internet nutzen.

Bisher ist es nicht gelungen, einen eindeutigen und einheitlichen Begriff von Telekooperation zu entwickeln und durchzusetzen. Krcmar (1996) unterscheidet zwischen Computer Supported Cooperative Work (CSCW) im weiteren und Telekooperation im engeren Sinne. Unter CSCW werden alle Formen computerunterstützter kooperativer Arbeit vom Austausch von Informationen via E-Mail über anspruchsvollere Message-Conferencing-Formen bis hin zu Workflow Ma-

nagement Systems und Decision Support Tools zusammengefasst. Telekooperation ist ein wesentliches Teilgebiet der CSCW und bezeichnet „die mediengestützte arbeitsteilige Leistungserstellung zwischen verteilten Aufgabenträgern, Organisationseinheiten und/oder Organisationen" (Reichwald et al. 1998: 65). Es lassen sich drei Formen oder „Dimensionen" (Reichwald/Möslein 1996; Reichwald et al. 1998) von Telekooperation ausmachen: Telearbeit (i. e. die mediengestützte verteilte Aufgabenbewältigung), Telemanagement (i. e. die mediengestützte verteilte Aufgabenkoordination) und Teleleistung (i. e. die mediengestützte verteilte Dienstleistung). Telekooperation ist in diesem Sinne Oberbegriff für alle Formen der durch Informations- und Kommunikationstechnologien unterstützten Arbeitskooperation, von der Sachbearbeitung an häuslichen Arbeitsplätzen, der Projektabwicklung in dezentralen Telezentren und verteilten Unternehmenseinheiten oder der mobilen Erbringung von Vertriebs-, Wartungs- oder Instandhaltungsdienstleistungen am Standort des Kunden bis zur internationalen Zusammenarbeit von Entwicklerteams (Reichwald/Möslein 1998: 71 f.).

Von unseren virtuell zusammenarbeitenden Kleinunternehmern im Multimedia- Internet- und Softwaresektor wissen wir, dass diese Kooperationen mit Firmen an anderen Standorten nicht unbedingt deshalb eingehen, weil sie von dem Kooperationsmodell so überzeugt sind, sondern weil sie sich Partner mit komplementären Fähigkeiten suchen müssen, um den Kunden oder Auftrag nicht zu verlieren. Von einer dynamischen, sich ständig ändernden Organisation haben sie wenig. Sie ähneln viel mehr dem Bild des klassischen Kleinbetriebs, mit dem patriarchalischen Chef und seinen loyalen Mitarbeitern, das von den Gründern meist im Fußballbild der Mannschaft und des Trainers beschrieben wird. Es bleibt bei einer Kooperation mit anderen, die man kaum kennt oder über das Internet gefunden hat, das Grundproblem des Vertrauens, wie man bei relativ offenen gemeinsamen Projekten Aufgaben und spätere Erlöse verteilt. Die Gefahr opportunistischen Verhaltens eines der Partner, dass er sich vom individuellen Handeln mehr verspricht als von den Vorteilen der Kooperation, indem er zum Beispiel während der Kooperation den anderen Partnern Kunden abwirbt, lauert - besonders in Krisenzeiten - allenthalben und ist auch durch noch so gute Verträge schwer zu bannen. Ebenso schwierig ist es, bei unbekannten Partnern ihre wirklichen Kompetenzen festzustellen. Man unterstellt in einer Art „swift trust", dass die anderen auch Professionelle sind und ihre Arbeit aufgrund sehr ähnlicher Regeln und Werte machen, was aber noch lange nicht heißt, dass sie - auch wenn sie wirklich professionell sind - einen ähnlichen Arbeitsstil oder -rhythmus haben, eine passende technische Ausstattung usw. Dies sind alles potentielle Enttäuschungsfaktoren. Es führt vielfach dazu, dass man Netzwerke auf relativ enttäuschungsfeste „old boy networks" und andere persönliche Beziehungen aufbaut, was nicht immer eine optimale fachliche und personelle Ausstattung der Projekte gewährleistet. Dies mag auch einer der Gründe dafür sein, dass in dem insgesamt sehr schmalen Segment der telekooperativ arbeitenden Betriebe die allermeisten ihre Kooperationsaktivitäten auf das lokale Umfeld beschränkten und nur sehr selten tatsächlich „global sourcing" betrieben. Überhaupt bleibt, um noch einmal Littmann und Jansen zu zitieren, das „Paradoxon der Kooperation" zentral. Es besteht darin, „ dass das Ziel der Kooperation in der Erweiterung des Handlungsspielraums für die einzelnen Unternehmen liegt. Gleichzeitig werden jedoch durch eine zunehmende Bindungsintensität die Handlungsspielräume, z.B. aufgrund einer entsprechenden Berücksichtigung der Partnerinteressen oder wegen einer steigenden wirtschaftlichen Abhängigkeit eingeengt" (Littmann/Jansen 2000, S. 75).

Die amerikanischen Forscher Malone und Laubacher haben Ende der 90er Jahre die Vision entworfen, dass Großunternehmen obsolet geworden sind und die Zukunft den viel flexibleren Klein- und Kleinstunternehmen gehört, die in ständig neuen Figurationen punktuell zusammenarbeiten. Sie tun dies, indem sie Telekooperation betreiben, also internetgestützt mit mehreren Partnern an verschiedenen Standorten an einem Projekt arbeiten. Sie weisen auf den US Trend

hin, dass vor 25 Jahren noch jeder fünfte Arbeitnehmer bei einem der 500 größten Unternehmen beschäftigt war, es aber heute nicht einmal mehr jeder 10te sei und der größte Arbeitgeber der USA die Zeitarbeitsfirma Manpower sei (2 Mio. Menschen unter Vertrag). Die These ist, dass durch elektronische Netze eine Rückkehr zum „vorindustriellen Modell winziger, autonomer Firmen möglich ist. Originalton Malone/Laubacher: „Elektronische Vernetzung erlaubt diesen neuen Mikrounternehmen, weltweit Reservoirs für Informationen, Fachkenntnisse und Finanzierungen anzuzapfen, die früher alleine Großunternehmen zur Verfügung standen. Damit erfreuen sich die Kleinfirmen vieler Vorteile der Großen, ohne dafür ihre Schlankheit, Flexibilität und Kreativität preisgeben zu müssen. Der Trend zu diesen E-Lancern könnte sich in Zukunft in Einklang mit den Fortschritten der Kommunikationstechnik und der höheren Effizienz von Netzen sehr verstärken. Sollte das tatsächlich eintreten, dann könnte die vorherrschende Organisationsform der Zukunft nicht die dauerhaft bestehende Kapitalgesellschaft sein, sondern das elastische Netz von Akteuren, das gelegentlich nur für ein oder zwei Tage existiert. Taucht eine konkretes Vorhaben auf, werden Vorschläge erbeten oder elektronische Suchanzeigen veröffentlicht. Einzelpersonen oder kleine Teams können reagieren, ein Netz bildet sich heraus. Und weitere Beteiligte werden eingebunden, sobald deren besondere Fähigkeiten gefragt sind. Wenn das Projekt erledigt ist, löst sich das Netz wieder auf. Gewissermaßen in den Fußspuren des jungen Linus Torvalds betreten wir das Zeitalter der temporären Unternehmen.“ Laubacher/Malone 1999, S. 32).

In der „small business, large network“ Vision wird die Figur des Freelancers stark aufgewertet, sei es, dass er als One-man/woman Mikrounternehmer selbständig Leistungen anbietet, sei es, dass er als Zu- und Kontraktarbeiter einen neuen Arbeitstyp des hochqualifizierten Geistesarbeiters darstellt, der dank Telekooperation potentiell global agiert ohne sein Haus oder sonstige ihm genehme Aufenthaltsorte zu verlassen. Dieses Szenario wird von Daniel Pink, der von einer „Free-Agent Nation“ träumt, auf die Spitze getrieben. Er geht von der Beobachtung aus, dass zwischen 1994 und 98 vier von fünf neuen Jobs in Firmen unter 20 Personen geschaffen wurden, insgesamt 9 Millionen (Pink 2001, S. 29). 69% aller neuen Firmen in den USA entstehen sozusagen im Wohnzimmer bzw. werden vom Haus des Eigentümers aus geführt. Er zitiert Studien, die Zahlen der „home-based business“ zwischen 24 und 27 Millionen angeben. Bis 2002 soll diese Zahl auf 37 Millionen wachsen (ders. S. 40/41). Dies sieht er als Bestätigung einer weitreichenden Abkehr vom „organisational man“, der Eigeninitiative und Kreativität bei der Arbeit stillschweigend gegen Sicherheit und bedingungslose Loyalität zur Firma tauscht. Für die neuen Mikrounternehmer ist nicht Geld, sondern Unabhängigkeit und die Möglichkeit, seine eigenen Prioritäten zu setzen, der Hauptmotivator (ders. S. 77). „For independent workers, freedom matters more than stability, and self-expression has replaced self-denial. Instead of hiding behind an organisation, free agents make themselves directly accountable. And rather than accept a prefabricated notion of success, they are defining success on their own terms. The result is that free agents have replaced the Protestant work ethic into a free agent work credo composed of four key elements: having freedom, being authentic, putting yourself on the line, and defining success on your own terms.“ (Pink 2001, S. 84).

Dass diese Wertschätzung von Selbständigkeit in „Unabhängigkeit und Freiheit“ eine neue Massenbewegung in Richtung Unternehmensgründung, besonders im Hightech Bereich in Deutschland in Gang gesetzt hat, lassen die statistischen Zahlen bezweifeln. Natürlich hat sich die Arbeit verändert, die Teilzeitarbeit hat zugenommen, wenngleich das Normalarbeitsverhältnis, also eine zeitlich unbefristete vollzeitige Stellung, noch immer für die große Mehrheit, nämlich 71% aller Erwerbstätigen, die Regel ist. 1985 waren 11,8% aller Erwerbstätigen selbständig, 1998 waren es 11,2% (Düren/Wedemeyer 2000, S. 172).

Wir haben in den von uns untersuchten Branchen der new economy nur begrenzt einen free agent Trend feststellen können. Noch nicht einmal in den USA, wo wir im vergangenen Jahr auf der Internet World Messe 100 Firmen befragt haben, ist der Einsatz freier Mitarbeiter der Standard. Nach einem Bericht der Unternehmensberatung Coopers/Leybrand aus dem Jahre 1996 waren in Betrieben des New Yorker Silicon Alley bis zu 33% aller Arbeitskräfte als freie Mitarbeiter oder Teilzeitangestellte beschäftigt (Ross 2000, S. 276). In unserer New Yorker Befragung beschäftigten 45% der Befragten gar keine Freelancer bzw. arbeiteten nur mit regulären Angestellten.

Bei unserer telephonischen Umfrage von Anfang diesen Jahres mit über 1000 (1071) Kleinunternehmen kamen ein Drittel ganz ohne den Einsatz von zusätzlichen Freiberuflern aus. Zwar arbeiten die meisten mit freien Mitarbeitern, sie würden aber, wie wir aus Intensivbefragungen der telekooperativ Arbeitenden wissen, sich im Falle einer Personalvergrößerung oft für einen Angestellten entscheiden. Diesem traut man mehr Identifikation zu. Man kann längerfristig mit ihm planen, er ist zur Stelle, sein Wissen wächst mit neuen Aufgaben, es bleibt in der Firma und man baut ein persönliches Verhältnis auf. Man muss sich auch vor Augen halten, dass ein befristeter oder „Freier Mitarbeiter" Job für viele nebenher arbeitende Studenten in Deutschland die beste Möglichkeit bedeutet, berufliche Erfahrungen zu sammeln und einen gleitenden Übergang ins Berufsleben zu organisieren.

Obwohl in der BRD vielfach von einem neuen, positiven Klima gegenüber Gründern und einer gesteigerten Bereitschaft zur Selbständigkeit die Rede ist, haben Autoren wie Lagemann und Welter, die die „neue Kultur" der Selbständigkeit untersuchen, ihr Kapitel zu Annäherungen an den Gründergeist mit der Überschrift „viel Spekulation - wenig exaktes Wissen" (Lagemann/Welter 1999, S. 115) versehen. Der Anteil der Selbständigen in den freien Berufen hat sich immerhin seit 1970 weit mehr als verdoppelt und auch der Anteil der weiblich selbständigen Freiberufler ist von 16,5% im Jahre 1970 auf 32,7% im Jahre 1996 gestiegen (Bögenhold 1999, S. 22). Dies betrifft aber eher Ärzte, die sich noch schnell niedergelassen haben, die Vielzahl der ausgebildeten Juristen ohne Anstellung und andere klassische Professionen. Die „Computerwoche" schreibt 1998: „In der Bundesrepublik sind etwa 10.000 bis 20.000 von etwa 564.000 Freiberuflern in der Informationstechnologie tätig" (Sonderheft 3/98, S. 34). Diese Zahlen sind relativ gering im Vergleich zu den traditionellen freien Berufen (Datenreport 1999, S. 94).

Bei den Problemen der Arbeits- und Berufsstatistik und ihren Klassifikationen könnte es sein, dass im kleinbetrieblich geprägten deutschen Softwaresektor eine beträchtlich größere Zahl von Einmann-Unternehmen und anderen Freiberuflern anzutreffen ist. Das Magazin „Der IT Freiberufler" hat aus dem Stand in zwei Jahren mehr als 15.000 Abonnenten gewonnen und damit einen Großteil der oben genannten 10.000 bis 20.000 IT Freiberufler erfasst. Wir haben aber auf diversen Veranstaltungen für Freiberufler den Eindruck gewonnen, dass von ihnen längst nicht alle freiwillig die Selbständigkeit gewählt hatten. Drohende Arbeitslosigkeit, fehlende Perspektive oder Alter waren für viele starke Motivatoren für die Selbständigkeit. Bei einer Internetbefragung des IAT Gelsenkirchen zu „neuen Selbständigen" gaben immerhin 23% an, dass für sie der Weg in die Selbständigkeit eine Alternative zur (drohenden) Erwerbslosigkeit gewesen ist (Vanselow 2000, S.11). Für diese Gründer ist der Weg in die Selbständigkeit wahrscheinlich die second best choice. Bei unseren Befragten sind es nur sehr wenige, die von Arbeits- und Perspektivlosigkeit als Hauptmotiv für die Selbständigkeit berichten. Es sind etwa 10%, die aufgrund von Enttäuschungen der Angestelltentätigkeit die Selbständigkeit vorgezogen haben. Die glamourösen, vorwärtsstrebenden, energischen und äußerst selbstbewussten Gründergestalten der Illustrierten, deren Firmen hohe Wachstumsraten dank Venture Capital haben, sind die kleine Minderheit unter ihren Unternehmerkollegen.

Die Geschichte des durchsetzungsfähigen Gewinnertyps, der rasche Markterfolge erzielt, ist nur die eine Seite der Medaille. Auf der anderen Seite finden sich die Verlierer, die in die Selbständigkeit entlassenen älteren Programmierer, die weitgehend weiblichen Heere freischaffender Graphiker und Journalisten, die arbeitslosen Akademiker, die sich notgedrungen selbständig machen, die Studienabsolventen, die hoffen, durch die Selbständigkeit zu einem Eintrittsticket für den Arbeitsmarkt zu kommen, die qualifizierten Frauen, die nach der Kinderpause wieder auf den Arbeitsmarkt drängen. Sie sind Teil des Systems der Kooperationsbeziehungen der Internetökonomie. Ohne die Scharen von Freelancern hätten ihre Protagonisten nicht die Möglichkeit, auf jeden Kundenwunsch bezüglich Produktionszeit und Produktspezifika sehr genau einzugehen und allgemein eine Flexibilität zu entfalten, die im Vergleich zu anderen Bereichen wirklich herausragend ist.

Insgesamt ist in der Bundesrepublik sowohl die small business-large network Vision als auch die Vorstellung eines schnell wachsenden Heeres von flexiblen, hochqualifizierten freien Mitarbeitern für die Arbeitsbedingungen und Arbeitsrealität der Beschäftigten des Hightech Sektors allgemein und der virtuell in Telekooperation Arbeitenden im speziellen nicht sehr wahrscheinlich. Sicherlich, für die Verlierer ist Selbständigkeit die zweite Chance. Warum sollte aber die Mehrheit der Arbeitsplatzinhaber in sehr unsicheren Zeiten ihren sicheren Job aufgeben und das unternehmerische Risiko eingehen - auch wenn man praktisch kaum mehr braucht als einen leistungsfähigen Computer? Die amerikanische Antwort im Sinne der Free Agent Bewegung darauf wäre: wegen der Selbstverwirklichung, der Herausforderung und der Unabhängigkeit. Dies sind aber auch die klassischen Werte der postmodernen Angestellten, die zwar um die Grenzen der Abhängigkeit wissen, diese aber - solange sie relativ große Freiheitsspielräume bei der Arbeit haben - nicht als unerträglich empfinden. Die Umfrageforschung zeigt, dass bei der Bevölkerung der BRD der Wert „Sicherheit“ kontinuierlich ganz oben steht, besonders in krisenhaften Zeiten. Davon weichen die Beschäftigten der new economy auch nicht ab. Der Hightech Sektor eignet sich vorzüglich für gesellschaftliche Projektionen, er ist aber bei genauerem Hinsehen als Modell und Laboratorium künftiger Entwicklung nur begrenzt tauglich.

Literatur:

Bammé, Arno; Feuerstein, Günther; Genth, Renate; Holling, Eggert; Kahle, Renate; Kempin, Peter (1983): Maschinen-Menschen Mensch-Maschinen: Grundrisse eine sozialen Beziehung, Reinbek: Rowohlt.

Beyhs, Olive; Hirsch, Bernhard (1999): Vertrauen in Netzwerken - eine Herausforderung für das Rechnungswesen. In: Franz Liebl (Hrsg.): e-conomy: Management und Ökonomie in digitalen Kontexten. Wittener Jahrbuch für ökonomische Literatur, Metropolis Verlag, S. 171-202.

Bhidé, A. V. (2000): The Origin and Evolution of New Businesses. Oxford: Oxford University Press.

Bögenhold, Dieter (1999): Unternehmensgründungen, Unternehmertum und Dezentralität. In: derselbe (Hrsg.): Unternehmensgründung und Dezentralität. Renaissance der beruflichen Selbständigkeit in Europa? Opladen, Wiesbaden: Westdeutscher Verlag, S. 7-27

Bronson, Po (1999): The nudist on the last shift and other true tales of Silicon Valley. New York: Broadway Books.

Datenreport (1999): Statistisches Bundesamt (Hrsg.): Datenreport 1999. Zahlen und Fakten über die Bundesrepublik Deutschland, Bundeszentrale für politische Bildung, Schriftenreihe Bd. 365, Bonn.

Davidow, William H.; Malone, Michael, S. (1993): The virtulal corporation: Structuring and revitalizing the corporation for the 21th century. New York (NY) u.a.: Summit Books. (dt. Übersetzung: Das virtuelle Unternehmen: Der Kunde als Co-Produzent. 2. Aufl. Frankfurt/Main u.a.: Campus, 1997.)

Düren, Harald; Wedemeyer, Michael (2000): Zwischen Burnout und Dropout - Das Leben, ein Kunstwerk? In: Engelmann, Jan (Hrsg.): Kursbuch Arbeit: Ausstieg aus der Jobholder-Gesellschaft – Start in eine neue Tätigkeitskultur? Stuttgart, München: Dt. Verl.-Anstalt, S. 167-179

Glott, Rüdiger; Paul, Gerd (2001): Unlimited jobs for a limited number of young people, Artikel für die Zeitschrift „Young" erscheint 2002.

Krcmar, Helmut; Lewe, Henrik; Schwabe, Gerhard (Hrsg.) (1996): Herausforderung Telekooperation: Einsatzerfahrungen und Lösungsansätze für ökonomische und ökologische, technische und soziale Fragen unserer Gesellschaft, Berlin u.a.: Springer.

Lee, Chong-Moon (2000): Four Styles of Valley Entrepreneurship. In: Lee, Chong-Moon et al. (eds): The Silicon Valley Edge: A Habitat for Innovation and Entrepreneurship. Stanford, California: Stanford University Press, S. 94-123

Littmann, Peter; Jansen, Stephan (2000): Oszillodox: Virtualisierung - die permanente Neuerfindung der Organisation. Stuttgard: Klett-Cotta.

Langemann, B.; Welter, F. (1999): Eine „neue Kultur" der Selbständigkeit? Zu Diskussion einer wirtschaftspolitischen Idee. In: Dieter Bögenhold (Hrsg.): Unternehmensgründung und Dezentralität. Renaissance der beruflichen Selbständigkeit in Europa? Opladen, Wiesbaden: Westdeutscher Verlag, S. 111-126

Malone, Thomas.W.; Laubacher, Robert J. (1998): The Dawn of the E-Lance Economy. Harvard Business Review, September/Oktober, S. 145-152

Malone, Thomas.W.; Laubacher, Robert J. (1999): Vernetzt, klein und flexibel - die Firma des 21 Jahrhunderts. In: Harvard Business Manager 2/99, S. 28-36

Pink, Daniel H. (2001): Free Agent Nation. How America's New Independent Workers Are Transforming the Way We Live. New York: Warner Books.

Reichwald, Ralf.; Möslein, Katrin (1996): Telearbeit und Telekooperation. In: Bullinger, H.-J.; Warnecke, H.-J. (Hrsg.): Neue Organisationsformen im Unternehmen. Ein Handbuch für das moderne Management. Berlin u.a.: Springer.

Reichwald, R.; Möslein, K.; Sachenbacher, H.; Englberger, H.; Oldenburg, S. (1998): Telekooperation. Verteilte Arbeits- und Organisationsformen. Berlin u.a.: Springer.

Rheingold, Howard (1991): Virtual reality. New York (NY) u.a.: Summit Books.

Ross, Andrew (2000): Jobs im Cybespace. In: Engelmann, Jan (Hrsg.): Kursbuch Arbeit: Ausstieg aus der Jobholder-Gesellschaft – Start in eine neue Tätigkeitskultur? Stuttgart, München : Dt. Verl.-Anstalt, S. 270-285

Shell (2000): Deutsche Shell (Hrsg.): Jugend 2000. Gesamtkonzeption und Koordinator: Arthur Fischer. Opladen: Leske + Budrich.

Vanselow, Achim (2000): Die Lebens- und Arbeitssituation von „Neuen Selbständigen". Ausgewählte Ergebnisse einer Online-Befragung, in: Petra Getfert (Hrsg.) Arbeitsbedingungen in IT-Arbeitsfeldern. Dokumentation des Workshops vom 28.9.2000, Sozialforschungsstelle Dortmund, Band 121, S. 11-20

Die Sicht eines kleinen Unternehmens auf die Branchenentwicklung und seine Kooperationserfahrungen

Ein Kommentar zum Beitrag von Gerd Paul[1]

Günter Tröger
Systemhaus Chemnitz GmbH
Annaberger Str. 240
09125 Chemnitz
gtr@shc.tcc-chemnitz.de

Ich bin Geschäftsführer der Systemhaus Chemnitz GmbH. Wir sind ein Kleinunternehmen mit 12 Beschäftigten und befassen uns mit Business-Software. Darunter verstehen wir ERP-Systeme, Internetanwendungen bis hin zur Gestaltung von Portalen. Wir haben auch bereits auf der Grundlage eines e-Shop-Systems einen Business-Shop als Pilotlösung entwickelt.

Ich bin zunächst einmal aufgefordert, meine Pro's und Contra's zum Vortrag des Herrn Paul vorzutragen. Ich möchte insbesondere auf die Fragen eingehen, die im Referat aufgeworfen worden sind. Ich muss sagen, dass ich bei verschiedenen Fragen zu anderen Schlüssen gekommen bin als Herr Paul. Bei den Unternehmensanforderungen würde sicher jeder sofort zustimmen, dass die Innovationsfähigkeit und Flexibilität eines Unternehmens gesichert sein muss, um sich eine wichtige Marktposition zu erarbeiten, und dass es seine Ertragsfähigkeit sichern muss. Das sind die Grundzüge von gut funktionierenden Unternehmen, gleich welcher Größe.

Herr Paul hat dargelegt, dass der Weg dorthin heißt: Aufbrechen straffer Unternehmensgrenzen. Jetzt kommt mein erstes „Aber": Wenn wir uns VW, Daimler Chrysler, die IBM oder andere Handelskonzerne ansehen, passiert das dort auch, aber in anderer Form. Diese großen Unternehmen versuchen ständig zu expandieren und zu wachsen. In Deutschland sind wir in der Situation, dass es nur noch ganz wenige Handelskonzerne gibt. Die Konzentration geschieht in der Art und Weise, dass andere Unternehmen übernommen worden sind.

Nehmen wir das Beispiel VW. Herr Piech ist dafür bekannt, dass er ein Unternehmen nach dem anderen weltweit hinzugewonnen hat. Er hat nicht auf Netzwerke gesetzt, wo die Unternehmensteile selbständige Einheiten sind, sondern Unternehmen dazugekauft. Bei Daimler-Chrysler verrät bereits der Name das gleiche Vorgehen. Meines Wissens wurden auch Mitsubishi-Anteile erworben, um weltweit agieren zu können. Die Reduktion auf wenige große Konzerne hat nicht in der Art und Weise stattgefunden, dass Teile des Unternehmens z.B. von Daimler ausgelagert wurden, sondern es hat das Umgekehrte stattgefunden - zumindest bei den großen Unternehmen. Ergebnis ist ein noch größerer Konzern, der mehr Macht hat und weniger Wettbewerb vorfindet und vielleicht auch träger wird.

[1] Dieser Beitrag basiert auf den Tonbandaufzeichnungen während des Workshops.

Nun soll hier von den kleinen Unternehmen die Rede sein. Für sie stellt sich die Frage: Wie kommen die Kleinunternehmen in dieser Situation zurecht? Natürlich kann ich hier nur aus eigener Erfahrung berichten. Mir ist in meiner Unternehmenspraxis aufgefallen, dass durch das Medium Internet sehr viel mehr Flexibilität organisiert werden kann. Es ist heute möglich, dass Entwickler an unterschiedlichen Standorten an einer Thematik arbeiten und auch zum Erfolg gelangen, weil das wichtige Problem der Kommunikation zwischen den Teammitgliedern dadurch zu einem großen Teil gelöst werden kann. Es wird nicht vollständig gelöst. Wer glaubt, dass mit multimedialen Instrumentarien die Entwickler an völlig verschiedenen Orten eine Aufgabe von Anfang bis zum Ende bearbeiten können, der liegt schief. Das funktioniert nicht. Es gibt immer Situationen im Projekt, bei denen es wichtig ist, persönlich zusammen zu kommen: Wir müssen miteinander reden. Ich muss ihre Gestik sehen. Sie können das auch über Kameras machen, aber es ist ein Unterschied, ob ich sie heute nur in Göttingen gesehen und gehört hätte oder ob sie hier im Raum sind. Ich denke, bei aller multimedialer Euphorie darf man das nicht unterschätzen. Ich kann eine Aufgabe am Ende nur dann zum Erfolg führen, wenn ich das bedenke.

Ich möchte diesen Silicon-Valley-Mythos noch einmal herausgreifen. Er bewegt mich immer wieder. Warum funktioniert das dort hervorragend? Und warum soll das an einem anderen Ort, z.B. in Deutschland nicht möglich sein? Sind hier die rechtlichen Bedingungen anders als in den USA im Silicon Valley? Haben wir andere staatliche Rahmenbedingungen? Gibt es hier andere und vielleicht schlechtere Fördermöglichkeiten? Steht bei uns weniger Risikokapital zur Verfügung? Bezüglich des Risikokapitals hat sich meiner Meinung nach in Deutschland vieles geändert. Der hohe Wissensstandard mag bezogen auf das Silicon Valley auch stimmen. Ich habe jedoch auch schon die Meinung gehört, dass das Niveau der Ausbildung der Leute in den USA keinesfalls besser ist als bei uns in Deutschland. Dabei spreche ich jetzt insbesondere über den Softwarebereich. Es wird sogar das Umgekehrte behauptet. Beispielsweise hatte die Intershop AG bereits organisatorische Maßnahmen ergriffen, um die gesamte Entwicklung in die USA zu verlagern. Das wurde schnell wieder rückgängig gemacht und die Entwicklung wurde wieder nach Jena zurückgeholt. Dafür gab es verschiedene Ursachen, zum einen, dass nicht genügend gut ausgebildete Fachkräfte im Softwarebereich - also insbesondere Informatiker - vorhanden sind. Und zum zweiten ist auch die Bindung an ein Unternehmen in den USA sehr viel weniger ausgeprägt als in Deutschland. Wenn sie also sehr komplexe Vorhaben realisieren wollen - und da würde ich die Aufgaben, welche die Intershop AG in Jena bearbeitet, hinzurechnen - dann lassen sich diese mit Leuten, die im Kopf schon das nächste Unternehmen oder die nächste Aufgabe haben, sehr viel schlechter realisieren. Das Know How, das wir erwerben, zahlt sich eigentlich erst nach zwei, drei Jahren aus. Dann trägt das seine Früchte. Wenn sie mit Leuten zusammenarbeiten, die nur für eine Aufgabe - sagen wir für ein halbes oder ein Jahr- mit ihrem Unternehmen verbunden sind, dann werden sie nie hinreichend Know How in einer Firma konzentrieren können, dass sie schwierige und komplexe Aufgaben lösen können. Das ist zumindest meine Erfahrung.

Ich mache mir immer wieder Gedanken, warum es kein Silicon Valley in Deutschland gibt. Herr Paul, sie stellen fest, dass es so in Deutschland nicht funktionieren würde. Gut. Aber im Grunde ist das für mich zu wenig. Ein ähnliches Phänomen der Nicht-Wahrnehmung bzw. Nicht-Übertragbarkeit zeigt sich für mich an der Situation des Standes der Softwarebranche in den Ostländern. Ich verfolge sehr aufmerksam Messen, Zeitschriften und Veröffentlichungen. Wenn sie Bücher oder Fachzeitschriften durchsehen, sei das die Computerwoche, das Java Spektrum oder Zeitungen, die sich mit dem Softwareentwicklungsprozess auseinandersetzen, dann passiert es ganz selten, dass sie in diesen Foren einen Artikel aus dem Osten, aus den neuen Bundesländern finden. Da frage ich mich auch: Warum ist das so? Warum werden wir nicht anders wahrge-

nommen und warum lässt sich die Softwarebranche hier bei uns in den neuen Bundesländern nicht besser entwickeln? Ich sehe den Ansatz eines Silicon Valley im Münchner Raum. Da findet aus meiner Sicht eine vielleicht vergleichbare Konzentration von Softwareunternehmen statt. Für mich wäre es nun interessant, was da anders ist und warum sich das nicht wiederholen lässt? Denn man muss meiner Meinung nach einfach zur Kenntnis nehmen, dass im Grunde fast alle Innovationen im IT-Bereich aus den USA kommen. Es gibt abgesehen von SAP mit R3 kaum noch eine deutsche Standardsoftware.

Das virtuelle Unternehmen - ein Netzwerk von durch Informationstechnologien verbundenen selbständigen Unternehmen mit spezifischen Kompetenzbereichen - so hatte es Herr Paul vorgestellt. Hier dazu mein „Aber". Aus meiner Sicht stellt sich die Frage, wie ich Kernkompetenzen sichern kann. Stellen wir uns vor, wir entwickeln eine sehr komplexe Software - wir müssen immer unterscheiden, ob wir über eine einzelne Programmieraufgabe reden oder über ein komplexes Softwaresystem - die ein Team von 20 Leuten erfordert. Das Thema wird in 12 oder 18 Monaten realisiert. Wenn ich als kleines Unternehmen nicht in der Lage bin, dieses komplexe Softwaresystem zu produzieren, muss ich also in dieses Netzwerk eintreten und mir Verbündete suchen. Verbündete Unternehmen, um von den Kompetenzen und von den Kapazitäten her diese Aufgaben durchführen zu können. Ich glaube, mir würden schon graue Haar wachsen, bevor ich die Aufgabe angegangen bin - wenn ich an das Danach denke. Eine Aufgabe ist nicht damit beendet, dass ich sie durchgeführt habe. Sie haben Garantiepflichten. Wie ist der gesellschaftlich-rechtliche Status der Unternehmer-VG? VG habe ich als Abkürzung eingeführt. Es ist also keine GmbH und keine AG, sondern eine virtuelle Gesellschaft. Wie ist sie strukturiert? Gibt es dafür eine Rechtsform? Wie sieht der rechtliche Rahmen aus, um solche Netzwerke erfolgreich agieren lassen zu können bzw. zu organisieren? Sie müssen noch weiter fragen. Wer schließt den Werksvertrag für das Produkt ab? (Wir nehmen an, das es ein größeres softwaretechnisches Produkt ist, was von der VG erstellt werden soll.) Wie kommt der Werksvertrag zustande? Ich glaube, der ganze rechtliche Rahmen stimmt da noch nicht. Wie steht es mit den Garantieverpflichtungen aus? Nach 15 oder 18 Monaten löst sich das Netzwerk wieder auf. Wer sichert ab, dass bei späteren Problemen oder bei einer Weiterentwicklung die Mitglieder des Netzwerkes noch zur Verfügung stehen?

Ich möchte noch auf ein weiteres Problem in diesem Zusammenhang zu sprechen kommen. Stellen sie sich vor, wir beteiligen uns an einer Ausschreibung. Es geht wieder um ein relativ komplexes Softwareprodukt, das erstellt werden soll. Der Kunde wünscht ein Produkt, das die Bündelung verschiedener Kompetenzen bei der Realisierung erfordert. Es gibt zwei Bieter auf dem Markt: ein (großes) Unternehmen, das fast alle Funktionalitäten mit den im Unternehmen vorhandenen Kompetenzen realisieren kann. Es erzählt dem Kunden, wie groß es ist. Und das erlebe ich immer wieder, wenn ich bei Veranstaltungen bin und andere Kollegen aus meiner Branche auftreten. Dann erzählen sie mir als erstes, wie groß sie sind. Sie erzählen nicht, was sie können, sondern wie groß sie sind, wie viel Leute dort arbeiten. Warum machen sie das? Um dem Kunden zu suggerieren: Bei uns bist du gut aufgehoben. Wir sind ein solides Unternehmen. Wir sind gewachsen. Bei uns hast du Sicherheiten. Wir als kleines Unternehmen sind auf der anderen Seite. Wir sind das Netzwerk. Wir verbünden uns mit anderen Partnern, um diese Kompetenzen auch bereitstellen zu können. Was glauben sie, für wen sich der Kunde entscheidet? Ich glaube, ich brauche diese Frage nicht zu beantworten. Er gibt natürlich dem großen Unternehmen den Auftrag, das fast alle Kompetenzen im Unternehmen hat. Das erscheint ihm sicherer. Auch an diesem Punkt sehe ich bei den Netzwerken mehr Probleme als Vorzüge.

Trotzdem möchte ich betonen, dass es trotz alledem wichtig ist, eine eigene Vision als kleines Unternehmen zu haben. Sie brauchen ein klares Konzept für die Entwicklung eines Produktes.

Die Umsetzung eines solchen Konzeptes ist schon komplizierter, das weiß ich aus eigener Erfahrung . Wir haben natürlich auch ein Produkt entwickelt - auch die Möglichkeit von Fördermitteln genutzt. Ich hatte ein sehr gutes Team zusammengestellt mit guten jungen Leuten, Absolventen von dieser Hochschule hier, und ich denke, es ist soweit alles ganz gut gegangen. Es hat jedoch die Fachkompetenz gefehlt, das Know How für den Anwendungsbereich. Wir hatten große Schwierigkeiten, das den jungen Leute zu vermitteln. Das ist auch ein Prozess: Sie analysieren ein komplexes Thema, das Team muss wachsen, es muss Erfahrungen sammeln und dazu kommen dann noch technische Probleme der Realisierung.

Abschließend noch etwas zu meinen Erfahrungen aus der Zusammenarbeit mit großen Unternehmen. Wir haben immer aus der wirtschaftlichen Notwendigkeit heraus mit großen Firmen zusammengearbeitet. Allerdings war das für mich nicht immer von Vorteil. Als im vergangenen Jahr drei wichtige Know How-Träger aus meiner Firma weggegangen sind, sind zwei davon Freelancer geworden und der dritte ist zu eben dieser großen Firma gegangen. Zur Zeit sind sie alle drei bei genau der großen Firma tätig, mit der ich damals zusammengearbeitet habe. Ich kann also ungefähr sagen, was es heißt, mit Nicht-Gleichen zusammenzuarbeiten. Zur Zeit versuche ich, eine Kooperationen mit gleichrangigen Unternehmen aufzubauen. Solche Kooperationen ergeben sich manchmal auch durch Zufall. Man ist im Gespräch mit anderen Firmen, bestimmtes Know How ist in dieser Firma nicht vorhanden. Die andere Firma ist willens, uns zu sehr guten Konditionen einzukaufen und auch Vereinbarungen zu treffen, z. B. sich gegenseitig keine Mitarbeiter abwerben. In diesem Fall ging es um ein spezielles Wissen zu Oracle Datenbanken. Es gehört sehr viel Vertrauen dazu, wenn solche Netzwerke funktionieren sollen. Das möchte ich abschließend noch einmal betonen.

Kommunikation und Kooperation in der Softwareentwicklung

Sabine Sonnentag
Technische Universität Braunschweig
Institut für Psychologie
Spielmannstraße 19
D-38092 Braunscheig
s.sonnentag@tu-bs.de

Dieser Beitrag fasst mehrere empirische Studien zusammen, in denen Merkmale und Vorgehensweisen von besonders leistungsstarken Softwareentwicklern untersucht wurden. Schwerpunkt der Darstellung ist dabei der Bereich „Kommunikation und Kooperation".

1 Fragestellung

Die bisherige Forschung über leistungsstarke Personen im allgemeinen und über leistungsstarke Softwareentwickler und Programmierer im besonderen konzentrierte sich auf deren Merkmale und Vorgehensweisen im kognitiven Bereich (Ericsson/Lehmann, 1996; Sonnentag, 2000). Mögliche Unterschiede zwischen leistungsstarken und weniger leistungsstarken Personen in anderen Bereichen wurden weitgehend vernachlässigt, darunter auch der Bereich Kommunikation und Kooperation. Dies ist vor allem deshalb problematisch, weil gerade im Softwarebereich Arbeit oft im Team erledigt wird und somit besondere Anforderungen an die Kommunikations- und Kooperationsfähigkeit der dort Tätigen gestellt werden. In der Vergangenheit äußerten sich Forscher eher skeptisch darüber, ob leistungsstarke Personen gleichzeitig auch über besondere Kompetenzen im kommunikativen und kooperativen Bereich verfügen (Shanteau, 1988; Stein, 1995). Es wurde angenommen, dass leistungsstarke Personen oft Menschen seien, mit denen man recht schlecht zusammenarbeiten könne.

Empirische Untersuchungen zu dieser Frage lagen jedoch kaum vor. Würden sich die Befürchtungen, dass leistungsstarke Personen über eher geringe Kompetenzen im kommunikativen und kooperativen Bereich verfügen, bewahrheiten, stünden Software-Projekte vor einem Dilemma: einerseits sind Software-Projekte auf leistungsstarke Personen angewiesen, die die Arbeit auf einem qualitativ hohen Niveau erledigen; andererseits würden aber genau diese leistungsstarken Personen nur wenig zur Zusammenarbeit im Team beitragen, die meist notwendig ist, um ein gutes Gesamtprodukt zu erstellen. Somit würde man einerseits zwar leistungsstarke Personen für die Arbeit gewinnen wollen, müsste aber andererseits deren relativ schwache kommunikativen und kooperativen Kompetenzen fürchten.

Ziel der empirischen Untersuchungen war somit, herauszufinden, ob und wie sich leistungsstarke Softwareentwickler eher von ihren durchschnittlich leistungsstarken Kollegen im kommunikativen und kooperativen Bereich unterscheiden. Dabei wurden Antworten auf die folgenden drei Forschungsfragen gesucht: (1) Wie werden leistungsstarke Softwareentwickler von ihren Kollegen wahrgenommen? (2) Was sehen leistungsstarke Personen bei der Aufgabenbearbeitung als wichtig an? (3) Was tun leistungsstarke Personen? Diese drei Forschungsfragen wurden in drei, aufeinander aufbauenden empirischen Studien untersucht.

2 Wie werden leistungsstarke Personen von ihren Kollegen wahrgenommen?

In einer ersten Studie gingen wir der Frage nach, wie leistungsstarke Personen von ihren Kollegen wahrgenommen werden (Sonnentag, 1995). Dabei haben wir keine fertigen Kategorien vorgegeben, sondern erfragten freie Beschreibungen der Merkmale leistungsstarker Softwareentwickler.

2.1 Methode der Studie 1

Die Daten wurden im Rahmen des „Interdisziplinären Projekts zur Arbeitssituation in der Softwareentwicklung“ (IPAS; Bittner/Hesse/Schnath, 1995; Brodbeck/Frese, 1994) erhoben. An der in diesem Projekt durchgeführten Hauptuntersuchung nahmen insgesamt 200 Personen aus 29 professionellen Software-Projekten aus 19 Firmen teil. Die Stichprobe beinhaltete sowohl Softwareentwickler, Personen mit Projektleitungsfunktion als auch Benutzervertreter und einige wenige andere. Die Befragten hatten durchschnittliche Berufserfahrung in der Softwareentwicklung von 5.7 Jahren. Die hier beschriebenen Auswertungen beruhen auf Daten von 159 Personen.

Zunächst führten wir in den beteiligten Projekten Peer Nominations durch: In Einzelinterviews fragten wir, wer im Team ein besonders guter Softwareentwickler bzw. eine besonders gute Softwareentwicklerin ist. Nachdem eine konkrete Person benannt wurde, fragten wir weiter, was diese Person auszeichnet, d.h. zu einem sehr guten Entwickler bzw. Entwicklerin macht. Diese Antworten klassifizierten wir anhand eines hierarchischen Kategoriensystems.

2.2 Ergebnisse der Studie 1

Alle Antworten wurden insgesamt den folgenden sechs Oberkategorien zugeordnet: Fachkompetenz, Teamfähigkeit, Arbeitsstil, kognitive Fähigkeiten, Benutzerorientierung und Motivation. Fachkompetenz, die die Unterkategorien ‘Berufserfahrung’, ‘projektspezifische Kompetenzen’ und ‘fachspezifisches Problemlösen’ beinhaltete, wurde am häufigsten als ein Merkmal eines sehr guten Entwicklers bzw. einer sehr guten Entwicklerin genannt (von 68.6 % der Befragten). An zweiter Stelle wurde die Teamfähigkeit genannt (von 54.1 % der Befragten), darauf folgte die Kategorie ‘Arbeitsstil“ (von 49.1 % der Befragten genannt). Alle anderen Merkmale fanden sich deutlich seltener in den Antworten. Die Kategorie ‘Teamfähigkeit’ umfasst als Unterkategorien die ‘Kooperationsfähigkeit’ (beispielsweise Fähigkeit und Bereitschaft, Wissen an andere weiter zu geben, Unterstützung von Kollegen bei fachlichen Problemen), ‘Kommunikationsfähigkeit’ (beispielsweise rege Beteiligung in Diskussionen, Verständlichkeit von Erklärungen und Fragen, gute Darstellung von Ideen und Konzepten), ‘menschliche Qualitäten’ (beispielsweise Ruhe, Freundlichkeit, Gelassenheit) sowie ‘Projektleitungskompetenz’.

Diese Ergebnisse zeigen, dass – zumindest in der Kollegensicht – die kommunikativen und kooperativen Kompetenzen ein wesentliches Merkmal leistungsstarker Softwareentwickler sind. Dies ist ein erster Anhaltspunkt dafür, dass sich inhaltliche Leistungsstärke und Kompetenzen im Bereich Kommunikation und Kooperation nicht unbedingt widersprechen. Einwenden könnte man an dieser Stelle jedoch, dass nicht alle leistungsstarken Softwareentwickler als ‘besonders teamfähig’ beschrieben wurden. Offen ist somit, wie die Personen wahrgenommen wurden, bei denen die Teamfähigkeit nicht als ein wichtiges Merkmal genannt wurde. Allerdings wurde auch nicht bei allen die Fachkompetenz als ein wesentliches Merkmal erwähnt, was darauf hinweist,

dass die durch die Kollegen abgegebenen Beschreibungen nicht notwendigerweise vollständig sind. Ein zweiter möglicher Einwand betrifft die Validität der Charakterisierungen durch die Kollegen. So könnte man argumentieren, dass diese Charakterisierungen nicht so sehr die tatsächlichen Merkmale der beschriebenen Personen wiedergeben, sondern vielmehr die impliziten Theorien der Befragten. Das würde bedeuten, dass die Befragten die Merkmale beschreiben, von denen sie glauben, dass sie ein sehr guter Softwareentwickler hat bzw. haben müsste.

Um diesen alternativen Erklärungen zu begegnen, haben wir weitere Untersuchungen durchgeführt, in denen wir die Merkmale leistungsstarker Softwareentwickler auf direkterem Wege erfasst haben. Zum einen haben wir erhoben, was leistungsstarke Personen selbst als wichtig ansehen, zum anderen wie sie sich tatsächlich in Kommunikationssituationen verhalten.

3 Was sehen leistungsstarke Personen bei der Aufgabenbearbeitung als wichtig an?

Die Frage, was leistungsstarke Personen selbst als wichtig ansehen, bearbeiteten wir im Kontext einer Studie, in denen die Bearbeitung einer Software-Designaufgabe im Vordergrund stand (Sonnentag, 1998). Konkret wollten wir wissen, welche Strategien leistungsstarke und durchschnittliche Softwareentwickler kennen und empfehlen, um Software-Designaufgaben gut zu bearbeiten.

3.1 Methode der Studie 2

An der Untersuchung nahmen 40 Softwareentwickler teil, die im Durchschnitt über eine knapp siebenjährige Berufserfahrung verfügten. Maß für die Leistungsstärke der teilnehmenden Softwareentwickler war deren Leistung in einer Software-Designaufgabe, dem 'Lift Control Problem' (Guindon, 1990). Diese Leistung wurde von unabhängigen Ratern anhand der Kriterien 'Qualität des Lösungsalgorithmus', 'Modularität', 'Verständlichkeit' und 'Detailliertheit' beurteilt. Die Personen mit den 20 besten Designlösungen wurden als 'leistungsstark' angesehen, die anderen als 'durchschnittlich'.

Als Maß für das Wissen über Strategien wurde die Anzahl der Strategien erhoben, die einem vorgestellten unerfahrenen Kollegen als Vorgehensprinzip empfohlen wurden (vgl. dazu Wolff, 1989). Die genannten Vorgehensprinzipien wurden mehreren Kategorien zugeordnet, wobei eine Kategorie 'Kooperation mit Kollegen' umfasste.

3.2 Ergebnisse der Studie 2

Insgesamt empfahlen 30 % der untersuchten Softwareentwickler, dass der vorgestellte unerfahrene Kollege mit anderen zusammenarbeiten sollte. Dabei betonten die leistungsstarken Personen die Nützlichkeit der Zusammenarbeit mit anderen signifikant häufiger (47 % im Vergleich zu 14 % der durchschnittlichen Personen). Hier könnte man argumentieren, dass leistungsstarke Personen möglicherweise die Kooperation nicht tatsächlich als wichtiger ansehen, sondern dass es ihnen generell leichter fällt, Vorgehensprinzipien zu verbalisieren. In der Studie haben wir deshalb ein zusätzliches Maß für die Verbalisierungsfähigkeit erhoben. Unter der Berücksichtigung dieser Verbalisierungsfähigkeit bleiben die Unterschiede zwischen den leistungsstarken und durchschnittlichen Softwareentwicklern jedoch bestehen.

Insgesamt heißt dies, dass leistungsstarke Personen die Kooperation mit anderen als wichtiger ansehen als durchschnittliche Personen. In einem weiteren Set von Analysen sind wir weiter der Frage nachgegangen, was die Softwareentwickler selbst als wichtig ansehen. Konkret haben wir an derselben Stichprobe der Softwareentwickler untersucht, wie sie in schwierigen Kooperationssituationen vorgehen würden (Sonnentag/Lange, in press). Auch da zeigte sich, dass die leistungsstarken Personen mehr Vorgehensmöglichkeiten nennen können, wie man schwierigen Kooperationssituationen begegnen könnte. Streng genommen implizieren diese Ergebnisse jedoch noch nicht, dass sich leistungsstarke Personen tatsächlich anders verhalten als andere. Sie legen jedoch den Schluss nahe, dass leistungsstarke Personen andere kognitive Repräsentationen von Aufgaben und Arbeitssituationen sowie den zu präferierenden Lösungsansätzen haben. Innerhalb dieser Lösungsansätze spielen Aspekte von Kommunikation und Kooperation eine große Rolle.

4 Was tun leistungsstarke Personen?

Die Frage, was leistungsstarke Personen in ihrer Arbeit wirklich tun, wurde in einer weiteren Studie untersucht (Sonnentag, 2001). Ziel dabei war es, objektive Daten über das Verhalten von leistungsstarken und durchschnittlichen Softwareentwicklern in einer realen Kommunikationssituation zu erhalten. Als ein Beispiel für eine reale Kommunikationssituation wurden Teamsitzungen ausgewählt.

4.1 Methode der Studie 3

In dieser Studie wurden zehn Teamsitzungen in zehn unterschiedlichen Softwareprojekten untersucht. Alle Sitzungen waren reguläre Projektmeetings, in denen reale Aufgaben und Probleme der jeweiligen Projekte besprochen wurden. Alle Teamsitzungen wurden auf Video aufgezeichnet und anschließend ausgewertet. Die Sitzungen dauerten zwischen 62 und 165 Minuten, im Durchschnitt 100 Minuten. An Sitzungen nahmen insgesamt 60 Personen aus den zehn Softwareprojekten teil. Bei den einzelnen Sitzungen waren zwischen drei und acht Personen anwesend.

Die Leistungsstärke der einzelnen Sitzungsteilnehmer wurde wiederum in einem Peer Nomination erhoben. Einige Wochen vor den Sitzungen wurden die Projektmitarbeiter schriftlich befragt, wer in ihrem Team ein sehr guter Softwareentwickler bzw. eine sehr gute Softwareentwicklerin sei. Alle Personen, die mindestens zweimal genannt wurden, wurden in den weiteren Analysen als „leistungsstark" angesehen, Personen, die nicht genannt wurden, als durchschnittlich. Personen, die einmal genannt wurden, wurden von den Analysen ausgeschlossen (an den Sitzungen nahmen sie natürlich teil). Keiner der Untersuchungsteilnehmer erhielt eine Rückmeldung darüber, wer von ihnen als leistungsstark klassifiziert wurde.

Das Kommunikationsverhalten in den Teamsitzungen wurde anhand der Videoaufzeichnungen ausgewertet. Dabei wurde für jeden gesprochenen Satz festgehalten, wer ihn gesagt hat. Zusätzlich wurden prozess-steuernde Aktivitäten ausgewertet. Als prozess-steuernde Aktivitäten wurden Äußerungen zum (1) Sitzungsmanagement, (2) Zielformulierung, (3) Problemanalyse und (4) Feedbacksuche kodiert. Ein Teil der Videoaufzeichnungen wurden zur Überprüfung der Beobachterübereinstimmung von zwei unabhängigen Ratern ausgewertet. Die erzielte Beobachterübereinstimmung kann als gut bezeichnet werden.

4.2 Ergebnisse der Studie 3

Eine erste Inspektion der Daten zeigte, dass Projektleiter/innen überdurchschnittlich häufig als „leistungsstark" benannt worden waren. Dies ist an sich plausibel, stellte jedoch ein besonderes Problem für die Auswertung der Daten dar. Würde sich bei den Analysen zeigen, dass Projektleiter/innen, die überdurchschnittlich häufig als besonders leistungsstark wahrgenommen werden, sich besonders stark an den Sitzungen beteiligen, könnte dieses Ergebnis nicht eindeutig interpretiert werden: die starke Beteiligung der leistungsstarken Projektleiter/innen könnte sowohl an ihrem Projektleiterstatus, als auch an ihrer Leistungsstärke liegen. Um die Ergebnisse der Analysen eindeutig interpretieren zu können, wurden in die Auswertungen nur diejenigen leistungsstarken Personen aufgenommen, die nicht gleichzeitig eine Projektleiterfunktion hatten. Somit beschränkten sich diese Auswertungen auf sieben Teamsitzungen – in zwei dieser Sitzungen war der Projektleiter bzw. die Projektleiterin als leistungsstark benannt worden; allerdings war in diesen Sitzungen noch mindestens eine weitere leistungsstarke Person anwesend. Die Analysen bezogen sich somit auf diese weiteren leistungsstarken Personen nicht aber den Projektleiter bzw. die Projektleiterin (in den Sitzungen waren sie jedoch natürlich anwesend).

Die Daten wurden für jede Sitzung getrennt analysiert. Tabelle 1 zeigt die Ergebnisse für die Gesamtbeteiligung an den einzelnen Teamsitzungen. Es wird deutlich, dass sich in vier der sieben Sitzungen die leistungsstarken Personen signifikant stärker am gesamten Sitzungsgeschehen beteiligten als die durchschnittlich leistungsstarken Personen (Sitzung 1, 3, 4, 5). In einer weiteren Sitzung (Sitzung 2) zeigte das Ergebnis in dieselbe Richtung, war allerdings nur marginal signifikant. In zwei Sitzungen jedoch zeigte sich ein entgegengesetztes Ergebnis (Sitzung 6) bzw. kein Unterschied zwischen den leistungsstarken und durchschnittlichen Personen (Sitzung 7). Dies waren genau die beiden Sitzungen, in denen auch ein leistungsstarker Projektleiter bzw. eine leistungsstarke Projektleiterin anwesend waren. Das bedeutet, dass sich in diesen Situationen die anderen leistungsstarken Personen mit ihren Äußerungen eher zurückhielten. Sie mischten sich meist nur dann in das Geschehen ein, wenn sie direkt gefragt wurden.

Tabelle 1: Beteiligung von leistungsstarken und durchschnittlichen Softwareentwicklern an der Gesamtsitzung

Sitzung	Vorstrukturierung	Vergleich	Projektleiter/in
1	Ja	↗ ($p<.05$)	Durchschnittlich
2	Ja	↗ ($p<.10$)	Durchschnittlich
3	Ja	↗ ($p<.05$)	Durchschnittlich
4	Nein	↗ ($p<.05$)	Durchschnittlich
5	Nein	↗ ($p<.05$)	Durchschnittlich
6	Nein	↘ ($p<.05$)	Leistungsstark
7	Nein	n.s.	Leistungsstark

Anmerkung:
↗ Leistungsstarke Personen beteiligen sich stärker (Signifikanzniveau in Klammern).
↘ Leistungsstarke Personen beteiligen sich weniger (Signifikanzniveau in Klammern).

Tabelle 2 zeigt die Ergebnisse für die Beteiligung an prozess-steuernden Aktivitäten. Dabei wurde zwischen stark vorstrukturierten und wenig vorstrukturierten Sitzungen unterschieden. Stark vorstrukturierte Sitzungen (Sitzung 1, 2, 3) zeichneten sich dadurch aus, dass bereits vor der Sitzung ein detaillierter – oft impliziter - Plan vorlag, welche Teilnehmer wann im Sitzungsablauf zu Wort kommen. In den wenig vorstrukturierten Sitzungen (Sitzung 4, 5, 6, 7) war dies nicht der Fall, obwohl auch dort inhaltliche Tagesordnungen vorhanden waren. Bei der Analyse zeigte sich, dass in den stark vorstrukturierten Sitzungen die leistungsstarken Personen sich nicht stärker an prozess-steuernden Aktivitäten beteiligten. In einer der Sitzungen (Sitzung 2) beteiligten sie sich sogar weniger an diesen Aktivitäten. In den wenig vorstrukturierten Sitzungen zeigte sich ein anderes Bild. Hier waren in drei der vier Sitzungen (Sitzung 4, 5, 6) die leistungsstarken Personen signifikant stärker an den prozess-steuernden Aktivitäten beteiligt. In Sitzung 7 war dies nicht der Fall; hier war jedoch die (ebenfalls leistungsstarke) Projektleiterin stark prozess-steuernd tätig.

Tabelle 2: Beteiligung von leistungsstarken und durchschnittlichen Softwareentwicklern an der prozess-steuernden Aktivitäten

Sitzung	Vorstrukturierung	Vergleich	Bemerkungen
1	Ja	n.s.	
2	Ja	↘ ($p<.10$)	
3	Ja	n.s.	
4	Nein	↗ ($p<.05$)	
5	Nein	↗ ($p<.05$)	
6	Nein	↗ ($p<.05$)	
7	Nein	n.s.	Starke Strukturierung durch Projektleiterin

Anmerkung:
↗ Leistungsstarke Personen beteiligen sich stärker (Signifikanzniveau in Klammern).
↘ Leistungsstarke Personen beteiligen sich weniger (Signifikanzniveau in Klammern).

Insgesamt hat diese dritte Untersuchung gezeigt, dass sich leistungsstarke Personen auch in ihrem tatsächlichen Kommunikationsverhalten von durchschnittlich leistungsstarken Personen unterscheiden und sie sich generell stärker an Sitzungen beteiligen. Diese Untersuchung weist jedoch auch darauf hin, dass leistungsstarke Personen nicht generell stark prozess-steuernd aktiv werden. Sie beteiligen sich vor allen in wenig vorstrukturierten Situationen als prozess-steuernd, d.h. dann, wenn eine solche Steuerung aufgrund der geringen Vorstrukturierung notwendig ist. Wenn die Vorstrukturierung jedoch hoch – und damit eine starke Prozesssteuerung auch nicht erforderlich ist – zeigen sie diese Aktivität nicht. Dies bedeutet, dass leistungsstarke Personen ihre Kommunikationsaktivitäten stark an die Erfordernisse der jeweiligen Situation anpassen.

5 Zusammenfassende Diskussion

In den dargestellten Studien wurde die Kommunikation und Kooperation aus unterschiedlichen Perspektiven untersucht: aus der Perspektive der Kollegen, aus der Perspektive der leistungsstarken (und durchschnittlichen) Softwareentwickler selbst sowie aus der Perspektive der unabhängigen Beobachtung in realen Situationen. Die Ergebnisse aus allen drei Studien weisen recht übereinstimmend darauf hin, dass es Unterschiede im Bereich Kommunikation und Kooperation zwischen leistungsstarken und durchschnittlich leistungsstarken Softwareentwicklern gibt: aus der Perspektive der Kollegen werden leistungsstarke Personen als besonders teamfähig beschrieben, aus der Perspektive der leistungsstarken Personen selbst wird der Kooperation eine besondere Bedeutung zugesprochen und aus der Beobachtungsperspektive wird schließlich deutlich, dass sich leistungsstarke Personen besonders stark an Sitzungen beteiligen und vor allem in schlecht strukturierten Situationen stark prozess-steuernd eingreifen.

Dies heißt, dass aufgrund der vorliegenden Untersuchungen die eingangs geäußerte Befürchtung, dass leistungsstarke Personen sich durch eher geringe Kompetenzen im kommunikativen und kooperativen Bereich auszeichnen, keine Bestätigung findet. Für Software-Projekte ist dies eine gute Nachricht. Die Besetzung von Software-Projekten mit inhaltlich leistungsstarken und kommunikativ-fähigen Mitarbeitern beinhaltet keinen Widerspruch. Vielmehr gehen inhaltliche Leistungsstärke und positive Aspekte der Kommunikation und Kooperation miteinander einher.

Trotz des recht konsistenten Gesamtergebnisses sind die dargestellten Untersuchungen mit einigen Einschränkungen verbunden. Zwei sollen hier benannt werden:

1 An den Untersuchungen nahmen primär Softwareentwickler aus Projekten teil, in denen – zumindest im Ansatz – Teamarbeit realisiert wurde. Dies impliziert, dass die Situation in Arbeitskontexten, in denen Softwareentwicklung primär als Einzelarbeit geschieht, sich anders darstellen könnte.

2 Ein zweites Problem stellt die Verwendung von Peer Nominations in einem Teil der berichteten Analysen dar. Es ist nicht auszuschließen, dass die Ergebnisse der Peer Nominations auch die wahrgenommene Kommunikations- und Kooperationsfähigkeit widerspiegeln. Das würde bedeuten, dass bestimmte Personen als besonders leistungsstark benannt wurden, weil sie im Bereich Kommunikation und Kooperation positiv auffallen. Wenn diese Personen dann im folgenden Schritt als besonders kompetent in diesem Bereich beschrieben werden bzw. sich stark an Kommunikationsprozessen beteiligen, ist dies relativ trivial. Allerdings zeigen die Ergebnisse der Studie 2, dass sich die Unterschiede zwischen leistungsstarken und durchschnittlichen Softwareentwicklern auch zeigen, wenn man die Leistungsstärke nicht über die Peer Nominations, sondern die Leistung in einer zu bearbeitenden Aufgabe operationalisiert. Somit ist anzunehmen, dass auch die anderen Ergebnisse bestehen bleiben, wenn man Leistungsstärke nicht nur über Peer Nominations erfassen würde.

Für die Ergebnisse, dass sich leistungsstarke und durchschnittliche Personen im Bereich Kommunikation und Kooperation unterscheiden, bieten sich mehrere Erklärungen an. Erstens ist es möglich, dass Kommunikation und Kooperation ursächlich zur Leistungsstärke beitragen. Durch intensive Kommunikation und Kooperation wird es möglich, wichtige Information von Kollegen oder Vorgesetzten zu erfragen, die dem einzelnen bei der Leistungserbringung zu gute kommt. Ähnliches gilt auch für das Erbeten von Feedback, das in der Folge auch leistungssteigernd wirken kann, da durch das Feedback Suboptimalitäten in Lösungsansätzen schneller deutlich werden können. Sollte die leistungsfördernde Wirkung der Kommunikation und Kooperation tatsächlich zutreffen, liegen Konsequenzen für die Praxis deutlich auf der Hand: In den Software-

Projekten müsste darauf hingearbeitet werden, dass ausreichend und effizient kommuniziert und kooperiert wird. Flankierende Maßnahmen der Personalauswahl und –förderung, die auf Kommunikation und Kooperation fokussieren, könnten diese Prozesse unterstützen.

Zweitens ist aber auch denkbar, dass Leistungsstärke einen Effekt auf Kommunikation und Kooperation hat. Leistungsstarke Personen haben in der Regel ein umfangreicheres und besser strukturiertes Wissen über ihr Gebiet und wissen meist besser, wie man bei der Aufgabenbearbeitung vorgehen sollte. Als Konsequenz werden sie beispielsweise von ihren Kollegen häufiger um Rat gefragt, wie bei Problemen zu verfahren sei oder häufiger zu internen Reviews hinzugezogen. Dies impliziert die Annahme, dass leistungsstarke Personen stark kommunizieren, weil sie leistungsstark sind.

Schließlich ist auch nicht auszuschließen, dass sowohl Kommunikation und Kooperation als auch Leistungsstärke von weiteren Faktoren, sogenannten Drittvariablen, abhängig sind. So könnte es beispielsweise sein, dass Softwareentwickler, die besonders motiviert sind, sowohl viel kommunizieren und kooperieren als auch eine gute Leistung zeigen – ohne dass Kommunikation und Kooperation kausal auf die Leistung wirken.

Um entscheiden zu können, welche dieser Erklärungen tatsächlich zutrifft, sind weitere Untersuchungen notwendig. Denkbar ist dabei auch, dass mehrere Mechanismen wirksam sind: so könnte es sein, dass Leistung einen Effekt auf die Beteiligung an Kommunikations- und Kooperationsprozessen hat und dass in Folge diese intensive Beteiligung an Kommunikation und Kooperation wiederum leistungsförderlich wirkt. Dies hieße, dass sich Leistungsstärke auf der einen und Kommunikation und Kooperation auf der anderen Seite möglicherweise wechselseitig verstärken.

Literatur

Bittner, U., Hesse, W.; Schnath, J. (1995): Praxis der Software-Entwicklung: Methoden, Werkzeuge, Projektmanagement. Eine Bestandsaufnahme. München: Oldenbourg.

Brodbeck, F. C.; Frese, M. (1994): Produktivität und Qualität in Software-Projekten. Psychologische Analyse und Optimierung von Arbeitsprozessen in der Software-Entwicklung. München: Oldenbourg.

Ericsson, K. A.; Lehmann, A. C. (1996): Expert and exceptional performance: Evidence of maximal adaptation to task constraints. Annual Review of Psychology, 47, S. 273-305

Guindon, R. (1990): Designing the design process: Exploiting opportunistic thoughts. Human-Computer Interaction, 5, S. 305-344

Shanteau, J. (1988): Psychological characteristics and strategies of expert decision makers. Acta Psychologica, 68, S. 203-215

Sonnentag, S. (1995): Excellent software professionals: Experience, work activities, and perceptions by peers. Behaviour & Information Technology, 14, 289-299.

Sonnentag, S. (1998): Expertise in professional software design: A process study. Journal of Applied Psychology, 83, S. 703-715

Sonnentag, S. (2000): Expertise at work: Experience and excellent performance. In C. L. Cooper & I. T. Robertson (Eds.), International Review of Industrial and Organizational Psychology. Chichester: Wiley, pp. 223-264

Sonnentag, S. (2001): High performance and meeting participation. An observational study in software design teams. Group Dynamics: Theory, Research and Practice, 5, S. 3-18

Sonnentag, S.; Lange, I. (in press): The relationship between high performance and knowledge about how to master cooperation situations. Applied Cognitive Psychology.

Stein, E. W. (1995): Social and individual characteristics of organizational experts. International Journal of Expert Systems, 8, S. 121-143

Wolff, S. (1989): Knowledge acquisition and possibilities for elicitating expert knowledge. In: Klix, F.; Streitz, N. A.; Waern, Y.; H. Wandtke (eds.): Man computer interaction research MACINTER II. Amsterdam: Elsevier, pp. 413-421

Leistungsstarke Softwareentwickler sind zur Teamfähigkeit erzogene Freaks!

Ein Kommentar zum Beitrag von Sabine Sonnentag[1]

Frank Stockmann
IVS-Solutions AG
Annaberger Str. 240
09125 Chemnitz
stocki@ivs-solutions.de

Ich bin Leiter der Entwicklungsabteilung unserer Firma und selbst aktiver Softwareentwickler. Ich möchte in meinem Kommentar auf die Teamfähigkeit von leistungsfähigen Entwicklern eingehen, auf ihre Kommunikationsfähigkeiten und die Zusammenhänge zwischen Kooperation, Kommunikation und Leistungsstärke. Ich möchte gleich zu Beginn anmerken, dass eben diese Zusammenhänge für mich kausal sind - die wichtigste Erkenntnis, die ich aus dieser Studie mitnehme.

Beim Stichwort „leistungsfähige Softwareentwickler" hatte ich sofort die Assoziation zum Einzelkämpfer. Das war so eine erste spontane Reaktion auf dieses Stichwort. Bei intensiverem Nachdenken ist mir jedoch deutlich geworden, dass diese Assoziation nicht stimmt. Bei den Ausführungen von Frau Sonnentag hat mich insbesondere gewundert, dass leistungsfähigen Softwareentwicklern ein besonders hohes Maß an sozialer Kompetenz zugeordnet wurde. Das bestätigen meine Erfahrung nicht in dem Maße. Ich denke, das ist eher eine Wunschvorstellung. Ein leistungsstarker Entwickler sollte natürlich auch teamfähig und sozial kompetent sein. Die Erklärung dieses Widerspruches zwischen den empirischen Ergebnissen der Untersuchung von Frau Sonnentag und meinen eigenen Erfahrungen ist folgende: Softwareentwicklung ist ein hochgradig kreativer Prozess. Softwareentwickler sind - so meine These - in diesem Prozess wie Künstler. Ich meine dabei hoch motivierte Softwareentwickler, die mit sehr viel Engagement und Tatendrang bei der Sache sind. Sie sind wie Künstler und sie haben auch die Eigenheiten von Künstlern. Ein Beispiel, an dem ich das darstellen möchte, ist die Kritikfähigkeit. Ein hochmotivierter Softwareentwickler lässt seine Vorgehensweise und seine Arbeitsresultate nicht gern diskutieren oder kritisieren. Der Grund dafür ist, dass durch seine große Motivation, durch sein großes Engagement eine sehr enge Bindung zum Produkt und zur Vorgehensweise entsteht. Kritik trifft ihn dann zwangsläufig persönlich. Um sich davor zu schützen, lässt er dieselbe lieber gar nicht erst zu.

Wie entsteht Teamfähigkeit und Kooperation bei leistungsstarken Softwareentwicklern? Gute Softwareentwickler können Probleme schnell erkennen und gut strukturieren. Sie können gleichzeitig Problemlösungsstrategien und Alternativen dazu entwickeln. Gute Softwareentwickler mit

[1] Dieser Beitrag basiert auf den Tonbandaufzeichnungen während des Workshops.

viel Erfahrung sind in der Lage, Problemlösungsstrategien auf die vorhanden Ressourcen abzubilden. Sie strukturieren das Problem, unterteilen es, und bilden es auf die vorhandenen Ressourcen ab. Zu den Ressourcen zähle ich sowohl die menschliche Arbeitskraft als auch Hardware und Software. Das bedeutet, dass ein leistungsstarker Softwareentwickler Teilaufgaben an andere Softwareentwickler abgibt. Er strebt Kooperation an, um die gestellten Aufgaben effektiv zu lösen - hier sehe ich einen kausalen Zusammenhang. Diese Aufgabenteilung erfordert auch ein bestimmtes Maß an Kommunikation (der zweite kausale Zusammenhang, den ich sehe) und wahrscheinlich entsteht dadurch der Eindruck, dass eben diese guten Softwareentwickler scheinbar teamfähig sind.

Ich spreche bewusst von scheinbarer Teamfähigkeit, denn sie kommt nur durch das Bedürfnis bzw. die Notwendigkeit zustande, Ressourcen optimal zu nutzen. Dieses Vorgehen funktioniert gut in einer Praxis, in der es auch weniger leistungsstarke Teammitglieder gibt oder eine Hierarchie im Unternehmen vorhanden ist. Problematisch wird diese Arbeitsweise unter gleichrangigen Softwareentwicklern. Wenn also ein Problem von Entwicklern gelöst werden soll, die sich in ihren Fähigkeiten gleichen und in ihrer Position im Unternehmen gleichrangig sind. Meine Lösung dafür ist erstens das Akzeptieren von speziellen Fähigkeiten. Auch in einem Team von „Superfreaks“ sind nicht alle gleich. Jeder hat spezielle Fähigkeiten, hat sich im Laufe der Arbeit auf ein Gebiet spezialisiert oder kann einige Sachen besser als andere. Der zweite, inhaltlich besonders wichtige Punkt ist eine klare Definition von Schnittstellen. Wenn sich alle, die in einem Team zusammenarbeiten, bewusst auf klare und fest definierte Schnittstellen einigen, dann ist das eine gute Voraussetzung für konfliktfreies Arbeiten. Der dritte Punkt ist die Toleranz anderer Ergebnisse. Alle drei Punkte sind meiner Meinung nach wichtige Faktoren für das Gelingen von Kooperation, und zwar nicht nur bezogen auf die Entwicklung von Software. Sie beziehen sich auf menschliche Eigenschaften, die sich im Begriff der Sozialkompetenz wiederfinden.

Frau Sonnentag erwähnte bei ihrer Präsentation, dass sich leistungsstarke Softwareentwickler mehr an informellen Treffen und Reviews beteiligen. Ich würde dieses Untersuchungsergebnis auf der Grundlage meiner Erfahrung anders interpretieren. Ich würde sagen: Leistungsstarke Softwareentwickler müssen sich mehr an informellen Treffen beteiligen. Sie müssen mehr an Reviews teilnehmen. Der Grund dafür ist ihr Know How und ihre Leistungsstärke. Kurz gesagt: Wer viel weiß, wird ständig gefragt, muss überall hin, und muss alles machen. Das ist der Nachteil von Leistungsstärke. Ich würde nie formulieren: Weil ich gut bin, spreche ich viel mit anderen Entwicklern, oder weil ich gut bin, gehe ich zu anderen Entwicklern, um deren Quelltext zu diskutieren. Die Formulierung von Frau Sonnentag unterstellt eine aktive Rolle des Entwicklers, als ob er von sich aus das Bedürfnis hätte, sein Wissen weiter und anderen Entwicklern konkrete Hinweise für ihre Programmierarbeit zu geben. Sie sind meiner Meinung nach eher ungewollt gezwungen - weil sie gut sind - durch die Nachfragen ihrer Teamkollegen und -mitarbeiter.

Die Ergebnisse zur Beteiligung in den Diskussionen haben mich nicht überrascht. Leistungsstarke Entwickler beteiligen sich weniger in stark strukturierten Meetings als in gering strukturierten. Eine starke Strukturiertheit bedeutet, ich gebe einen engen Rahmen vor. Dieser enge Rahmen beschränkt auch die Entfaltungsmöglichkeiten, die Möglichkeiten zu Eigeninitiative. Beides sind wichtige Motivationsmomente für leistungsstarke Softwareentwicklern. Liegt eine geringe Struktur in einem Meeting oder in einem Projekt vor, dann findet sich dort auch der Freiraum, den leistungsstarke Entwicklern zur Entfaltung benötigen. Strukturen zu schaffen in einem strukturlosen Raum ist ja ein wesentliches Element ihre Art der Problemlösung. So kommen sie von den Aufgaben zum Ziel. Daraus ergibt sich nahezu zwangsläufig eine größere aktiv-verbale Beteiligung.

Ich sehe einen zweiten kausalen Zusammenhang zwischen der Leistungsfähigkeit eines Entwicklers und seinen - erzwungenen - Kommunikationsfähigkeiten. Er muss kommunizieren können oder diese Fähigkeiten entwickeln und ausbauen. Sie sind ein wichtiges Mittel, Leistungsfähigkeit zu beweisen und durchzusetzen. Ich würde jedoch nicht den Schluss ziehen wollen - wie es Frau Sonnentags Ausführungen nahe legen könnten - dass kommunikative Fähigkeiten dabei helfen, leistungsfähiger zu werden. Zwar lernen wir alle in sehr kommunikativen Zusammenhängen - im Studium, in der Arbeit, im Team - trotzdem ist sie nur ein sekundäres Mittel, um wirklich leistungsfähig zu werden. Ich sehe als wesentliche Merkmale für leistungsstarke Softwareentwickler eine klar Motivation, Neugier und einen kreativen Drang. Die Entwickler, die ich kenne, die wirklich gut sind in der Softwareentwicklung, haben eine Art innere Energie. Ihr Bedürfnis ist es, ihre Kreativität genau auf diesem wirtschaftlichen oder wissenschaftlichen Gebiet auszuleben. Sie drücken sich dadurch selbst aus. Sie erhalten aus der Art und den Ergebnissen ihrer Arbeit ihre Motivation. Sie befriedigen darüber ihr Bedürfnisse nach Selbstverwirklichung. Das zieht weitere Erfolgserlebnisse und eine Bestätigung ihres Vorgehens nach sich. Ich glaube Entwickler, die in unserer Branche gut sein wollen, müssen Elemente von einem „Freak“ haben - auch wenn diese Bezeichnung in der Informatik einen negativen Ruf hat. Wer einfach nur sein Handwerk gut lernt, wer alle Befehle, die Syntax, alle Vorgehensweisen solide beherrscht, der wird immer besser als der Durchschnitt sein. Aber um an die Spitze zu kommen - und um solche Entwickler ging es ja hier - braucht es noch einen zusätzlichen Kick, denn dieser Entwickler muss über Grenzen gehen, auch über eigene Grenzen. Er muss vorgegebene Bahnen durchbrechen und ausbrechen können.

Der 40-jährige Softwareentwickler

Altern in der Softwarebranche

Uwe Lünstroth
Lehrstuhl Technikphilosophie
BTU Cottbus
Universitätsplatz 3-4
03044 Cottbus
luen@tu-cottbus.de

1 Einführung

Innovationsstrategien in Softwareunternehmen lassen oft eine Beachtung, Nutzung und Förderung der Erfahrung älter werdender Entwickler vermissen. Dieser Beitrag diskutiert Ursachen und mögliche Maßnahmen zur Verbesserung dieser Situation. Die Diskussion über die „Greencard" für Informationstechnologie-(IT)-Fachkräfte förderte erfreulicherweise auch die Zustimmung zu einer notwendigen, forcierten Weiterbildung von Software-Spezialisten[1]. Viele IT-Verbände, sowohl der Arbeitgeber-, der Unternehmensseite als auch der Seite der betroffenen Informatiker und Programmierer, der IT-Selbständigen sowie der Gewerkschaft betonen die Nützlichkeit der Initiative der Bundesregierung für die spezifische Aufgabe, einen aktuellen, mittelfristigen Stellenmangel zu beheben. Gleichzeitig geht der Tenor der Äußerungen dahin, dass zukünftig verstärkt auf eine zügige und praxisorientierte Ausbildung als auch auf eine verbesserte Unterstützung der Weiterbildung von Computerspezialisten der verschiedenen Qualifikationsniveaus und auch der verschiedenen Altersbereiche von Seiten der Verbände geachtet werden muss.

2 Die Vorurteile

Unter den jüngeren Entwicklern herrscht die Vorstellung vor, dass sich viele der älteren Mitarbeiter auf dem einmal erworbenen Fachwissen ausruhen und innovative Impulse eher blockieren. Dieses Urteil tritt in den unterschiedlichen Unternehmen durchgehend auf - oft unabhängig davon, ob man persönlich ältere Softwareentwickler tatsächlich kennt. Zur Illustration sollen einige der Vorurteile im folgenden aufgelistet werden.[2]

Die Einstellung von älteren Entwicklern zu Risiko- und Innovationsorientierung unter der Berücksichtigung der Randbedingungen bei der Teamarbeit wurden generell so bewertet:

[1] Vgl. beispielsweise: Computerwoche Nr. 11 vom 17.3.2000, Seite 1: „Schröder bewilligt 20 000 Green Cards".

[2] Die im folgenden wiedergegebenen Aussagen geben den Eindruck unserer explorativen Befragungen wieder. Sie haben kein „statistisches Gewicht", sind eher anekdotisch und ersetzen eine Meinungsumfrage nicht. Dennoch kann man sie auf dieser Ebene der Untersuchung für charakteristisch halten.

- „Die älteren Softwareentwickler sind vorsichtiger, sie haben mehr Angst vor Fehlern, da sie woanders keine Arbeit mehr finden können. Sie werden zu Abwägenden und Zögerern."
- „Wenn man in eine Gruppe von jüngeren Entwicklern einen Älteren setzt, dann wird der automatisch zum Bremser."

Andererseits hört man jedoch häufig eine uneingeschränkt positive Bewertung der sozialen Kompetenz und auch der Kundenorientierung bei älteren Softwareentwicklern:

- „Der Ältere kann dem Kunden besser zuhören."
- „Die älteren Entwickler haben einen Erfahrungsvorsprung. Gerade von der Universität kommende Informatiker denken noch nicht in Zusammenhängen des Marktes und des globalen Wettbewerbs."

Verallgemeinernde Vorurteile gegenüber älteren Mitarbeitern sind aber genau so zu finden, z.B. bezüglich der Fähigkeit des Lernens bei schnellem technologischen Wandel:

- „Die älteren Entwickler kommen nicht mehr so mit. Die packen das nicht mehr und bringen bezüglich neuer Methoden und Synthesen nur die Hälfte wie Hochschulabgänger."
- „Die Älteren tun sich schwer mit neuen Konzepten, wie z.B. der objektorientierten Programmierung. Eine solche Umlernproblematik wird es als „Generationenkonflikt" immer geben. Das sieht man heute schon bei rivalisierenden Gruppen jüngerer Entwickler mit verschiedenen Programmiersprachen. In 20 Jahren ist auch hier das Denken verhärtet."
- „Gerade ältere Softwareentwickler wollen sich eher auf ihren Fachkenntnissen ausruhen. Jüngere sind auf die Technologie versessen. Ältere bringen ihre Erfahrung in Märkte ein und das, was sie aus früheren Fehlern gelernt haben."

Fast durchweg - auch unter Informatik-Experten - wird in den Unternehmen das Fazit gezogen, ältere Entwickler seien an sich für die Arbeit mit älteren Programmen und Systemen gut geeignet, aber für die neuesten Anwendungen, von denen die jüngeren Entwickler begeistert sind, nicht zu gebrauchen.

Dieses problematische Bild vom älteren Softwareentwickler hat sich aus unterschiedlichen Gründen in diesem Tätigkeitsbereich festgesetzt. Eine wesentliche Ursache bilden die Erfahrungen mit vielen der älteren Softwareentwickler der 80-er und 90-er Jahre, die in den 70-er Jahren als Umschüler aus kaufmännischen und pädagogischen Berufen in die Softwareentwicklung gewechselt sind. Ihnen fehlte ein fundierter Wissenshintergrund aus einer Hochschulausbildung in Informatik. Deshalb fiel es ihnen schwerer, mit der technologischen Entwicklung Schritt zu halten. Neue Konzepte bei Software und Hardware wurden von ihnen als zu komplex empfunden, die notwendigen Lernanstrengungen als zu groß. Beispielsweise zeigten sich in den durchgeführten Fallstudien, dass auch bei Beibehaltung der Programmiersprache Cobol der Wechsel vom strukturierten zum objektorientierten Programmieren als eine zu große Herausforderung an Entwickler im Alter zwischen 40 und 45 Jahren empfunden wurde.

Die oben kurz skizzierten Vorurteile entstehen weiterhin durch die Ablösung von Softwarekonzepten und Programmiersprachen, so dass schon nach wenigen Jahren die neu Ausgebildeten sich mit einem Wissenskanon beschäftigen, der sich gegenüber demjenigen Wissen, mit dem die schon länger im Tätigkeitsfeld Beschäftigten arbeiten, doch stark unterscheidet. Die älteren Systeme und Programme, auch wenn sie sich ökonomisch betrachtet durchaus noch „rechnen", haben bei den jüngeren Entwicklern das Image des altmodischen und sind für sie uninteressant.

Durch eine solche Einstellung werden Kontakte und der Erfahrungsaustausch zwischen den verschieden alten Softwareentwicklern nicht gerade gefördert.

Als Folge dieser Vorurteile tritt in den Unternehmen bei den Teams tendenziell eine Trennung nach Altersgruppen auf. Oft werden entlang der technologischen Generationen verschiedene Teams gebildet, in denen auch jeweils separierte Generationen von Softwareentwicklern mitarbeiten. Verstärkt und zementiert wird diese Tendenz aber auch durch die Einsatzplanung in vielen größeren Unternehmen, in denen die Priorität bei Programmen liegt, die längerfristig mit großer Sicherheit und in hohem Maße störungsfrei laufen. Angesichts solcher eher konservativen Einstellungen ist die Gefahr der Nischenbildung besonders groß, da zunächst einmal nicht abzusehen ist, wann man auf ein neues Softwarekonzept übergeht. Solange aber ein solcher „Systemwechsel" für den älter werdenden Mitarbeiter nicht abzusehen ist, werden die Weiterbildungsmaßnahmen kaum in die Richtung gehen, eine Aneignung aktueller Technologien anzustreben. Das alles führt dazu, dass man sich tatsächlich in vielen Entwicklerteams auf dem Fachwissen des engen Spezialbereichs beschränkt und ausruht, da es so scheinen mag, dass dieses Wissen noch eine unbegrenzte Verwertungsdauer besitze.

3 Ergebnisse unserer Fallstudien

Der Lehrstuhl für Technikphilosophie der Brandenburgischen Technischen Universität Cottbus untersuchte in den vergangenen Jahren im Rahmen eines Verbundvorhabens des BMBF[3] zu den Folgen des demographischen Wandels[4] die Situation und die möglichen unterstützenden Maßnahmen für älter werdende Softwareentwickler in 16 ausgewählten Fallstudien-Unternehmen verschiedener Branchenausrichtung.

Das hauptsächliche Ergebnis lässt sich in einem Satz zusammengefasst erläutern als die Erkenntnis, dass keineswegs Defizite aufgrund eines erhöhten Lebensalters (der Softwareentwickler über 40 Jahre) als Ursache dafür angesehen werden können, warum ältere Software-Experten oft einem negativen Image unterliegen. In der Regel (so erbrachten qualitative Interviews mit Personalleitern und Mitarbeitern in der Softwareentwicklung) ermangelt es an der Nutzung von Weiterbildung und Erfahrung mit Tätigkeitswechseln. Aufgrund derartiger Defizite erscheinen die älteren Mitarbeiter in den Softwareunternehmen den jüngeren Entwicklern häufig als inflexibel und für das Geschäft im sich technologisch rasch wandelnden IT-Bereich als ungeeignet.

Erst die Vernachlässigung von Standards der Weiterentwicklung der Qualifikation und der Erweiterung der ausgeübten Tätigkeit lässt das Bild des älteren gegenüber dem jüngeren Entwickler, die ihre Innovationsfähigkeit aus in der noch aktuellen Ausbildung und teilweise dem Karrierewillen schöpfen, schlecht aussehen. Das spezifische Innovationspotential der älteren Softwareentwickler, das diese aufgrund der Markt- und Kundenkenntnis sowie durch übergreifende Erfahrung mit Projekten und deren Organisation einbringen könnten, wird oft auch aufgrund einer einseitig auf die Nutzung des aktuellen Hochschulwissen abgestimmten Personalpolitik nicht ausgeschöpft oder erst gar nicht entwickelt.

---

[3] Für einen kurzgefassten Überblick über die Ergebnisse des Gesamtprojektes, d.h. aller fünf Forschungsverbünde, siehe die 60-seitige Broschüre „Zukunftsreport demographischer Wandel" (Pack et al. 1999), die über folgende Internetseite bestellt werde kann: http//www.demotrans.de

[4] Für den ausführlichen Ergebnisbericht unseres Forschungsprojektes (Fördernummer BMBF 01 HH 96098) siehe Berndes et al. 2001

Dabei wird zunächst vergessen oder gar verdrängt, dass die heute jungen Mitarbeiter in spätestens 10 bis 15 Jahren ebenfalls zum „alten Eisen“ gehören werden, wenn ihnen nicht permanente Weiterbildung und eine Weiterentwicklung auf verschiedenen Erfahrungsebenen nahegebracht werden kann. So wesentlich dabei die Eigeninitiative des Entwicklers selbst in den Vordergrund gestellt werden muss, so wenig reicht der Appell an diese aus. Unter den in der Softwareentwicklung arbeitenden Mitarbeitern finden sich immer noch relativ viele Personen, die meinen, auch nach zehnjähriger Berufszugehörigkeit noch in einem engen fachspezifischen Bereich arbeiten zu können. Der Wechsel zu anderen Software-Paradigmen und IT-Konzepten fällt dann schwer, wenn die Entwickler sich nicht frühzeitig darauf einstellen, auch für den Bereich übergreifender Erfahrung (Projektmanagement, Kundenorientierung, Qualitätssicherung, etc.) Weiterbildung zu betreiben.

Im folgenden sollen ein paar der Maßnahmen benannt und erläutert werden, die in den Gesprächen mit den Unternehmensverantwortlichen der softwareentwickelnden Firmen als mögliche Schritte in Richtung einer Förderung älter werdender Softwareentwickler angesehen wurden. Wir möchten damit eine vertiefte inhaltliche Diskussion der Weiterbildungs- und Laufbahnproblematik im Bereich der Softwareentwicklung, sowohl für das dortige Personalmanagement als auch zur „Bewusstseinsbildung“ der Mitarbeiter anregen.

4 Teamzusammensetzung in den Phasen des Softwareentwicklungsprozesses

Die Softwareunternehmen sind daran interessiert, Spezialisten und Mitarbeiter mit aktuellem technologischen Wissen zu beschäftigen. Deshalb ist ein jüngerenzentriertes Personalmanagement in der Softwarebranche durchaus verständlich, ist dies doch ein Gebiet, in dem das aktuelle Wissen über Technologien entscheidend für die Wettbewerbssituation ist.

Auf der anderen Seite führt eine Innovationsstrategie, die sich zu einseitig auf das aktuelle Wissen und die Beschäftigung junger Mitarbeiter konzentriert zu einer Haltung, die eine Entwicklung von auch in der IT-Branche notwendiger, übergreifender Erfahrung zumindest behindert. Berufliche Erfahrung wird in jüngerendominierten Teams eher negativ bewertet und als hinderlich bei einer innovativen Tätigkeit angesehen. Oft wird so beispielsweise der Beitrag von Erfahrung im Umgang mit den Kunden, im Sinne einer Beurteilungsfähigkeit von deren Wünschen vor dem Hintergrund technologischer Möglichkeiten (Kundenorientierung der Innovationstätigkeit) übersehen. Auch die Bedeutung der Erfahrung im Zusammenhang mit der Durchführung und Leitung von Projekten wird unterschätzt.[5]

Im Rahmen einer ausführlichen Delphi-Expertenbefragung unter 29 Experten der Informatik, Arbeitspsychologie und des Personalmanagements wurde die Relevanz bestimmter Leistungspotentiale[6] für die verschiedenen Phasen der Softwareentwicklung[7] ermittelt. Dazu wurden die

[5] Dies mag auch ein verursachendes Element in der Misere des Projektmanagements bei Softwareprojekten sein, dass zu viele der zukünftigen Projektleiter meinen, auf frühere Projekterfahrungen nicht mehr zurückgreifen zu dürfen, da diese mit dem Aufkommen neuer Technologien ebenfalls als veraltet zu betrachten seien.

[6] Ein Leistungspotential stellt eine Voraussetzung für das Erhalten, Erlernen und Anwenden von Fertigkeiten dar und enthält damit den eher formalen Anteil der beruflichen Qualifikation als auch darüber hinausgehende soziale Kompetenzen. Der in der betriebswirtschaftlichen Literatur verwendete Begriff der Qualifikation wird in ähnlicher Weise definiert wie der Begriff Leistungspotential, allerdings ohne Betonung der dynamischen, altersvariablen Veränderlichkeit. Während ein Leistungspotential explizit als Voraussetzung verstanden wird, bezeichnet der Begriff Qualifikation einen Zustand (Vgl. Littek et al. 1983, S. 128).

Experten gebeten anzugeben, in welchen Phasen die einzelnen Leistungspotentiale von besonderer Bedeutung sind.

Insbesondere ist zu der Zuordnung der für die einzelnen Phasen bedeutsamen Leistungspotentiale folgendes anzumerken: Ähnlich wie die Abstraktionsfähigkeit wird auch der Erfahrung zu Beginn des Softwareentwicklungsprozesses eine primäre Bedeutung zugewiesen. Als mittelmäßig bedeutend wird die Erfahrung in allen weiteren Phasen außer der Codierung, dem Modultest und der Wartung angesehen. Nicht überraschend ist, dass die soziale Kompetenz beim Kunden als Leistungsmerkmal primär für den Beginn und das Ende des Softwareentwicklungsprozesses angesehen wird. Für die zwischen Systementwurf und Implementation liegenden Phasen ist die soziale Kompetenz, offensichtlich wegen kaum auftretendem Kundenkontakt, nicht gefragt. Eine mittlere Bedeutung wird dieser Leistungsfähigkeit bei der Implementation beigemessen.

Durch die Experten wurden die physischen und sozialen Merkmale in ihrer Bedeutung für die einzelnen Phasen des Prozesses der Softwareentwicklung bestimmt. Unter Zuhilfenahme der gerontologischen Aussagen (siehe Berndes et al. 2001) ließen sich danach diejenigen Phasen der Softwareentwicklung benennen, in denen entweder eher eine größere Anzahl älterer Mitarbeiter erwünscht sein sollte, oder auf der anderen Seite eine geringere eher die Gewähr für eine bessere Innovationsfähigkeit des Unternehmens bieten würde. Fazit: Insbesondere die Phasen, die speziell Kompetenz im Kundenkontakt erfordern, und diejenigen, in denen wesentlich Erfahrungsmomente zum Tragen kommen, sollten für ältere Mitarbeiter offen stehen.

Als Schlussfolgerungen für die Verwendung älter werdender Softwareentwickler (also Mitarbeitern der Softwareentwicklung ab etwa einem Alter von 35-40 Jahren), denen in der Experten-Befragung eine Zunahme von Erfahrung und sozialer Kompetenz zugesprochen wurde, ergaben sich folgende Aussagen für den Einsatz in bestimmten Phasen des Prozesses der Softwareentwicklung:

- Bei diesen älter werdenden Entwicklern sollte ein Einsatz in den Anfangsphasen der Softwareentwicklung angestrebt werden. Dabei kann die Aufgabenanalyse von Teams durchgeführt werden, in denen ältere Mitarbeiter in der Mehrzahl sind. Für den Systementwurf, für den nach den Expertenaussagen Kreativität und das Einbringen neuer Ideen einen wichtigen Anteil bilden, ist ein stärker altersmäßig gemischtes Team wünschenswert.
- In abgeschwächter Form gilt das zum Systementwurf gesagte auch für den Modulentwurf. Allerdings spielt in diesen Entwicklungsphasen die Abstraktionsfähigkeit eine untergeordnete Rolle, so dass auch ältere Mitarbeiter hier in altersmäßig gemischten Teams eingesetzt werden können, die in diesem Leistungsmerkmal Probleme aufweisen.
- Für die Phasen Integration, Systemtest und Implementation wird wiederum stark Wert auf Erfahrung gelegt, so dass man dabei einen erhöhten Anteil älterer Mitarbeiter in den Teams empfehlen kann. Insbesondere gilt das für den Systemtest, aber auch für die Implementation, da hier das Vorhandensein von Verantwortungsbewusstsein nach Aussagen der Experten besonders wichtig erscheint.
- Ähnlich wie in der Beurteilung der Anfangsphase des Entwicklungsprozesses gilt das Argument größerer Sozialkompetenz beim älter werdenden Entwickler auch in den abschließen-

[7] Für die Phaseneinteilung wurde das einfache Wasserfall-Modell der Software-Entwicklung zugrundegelegt. Es geht von einem linearen Ablauf in den Phasen Aufgabenanalyse, Systementwurf, Modulentwurf, Codierung, Modultest, Integration, Systemtest, Implementation, Nachsorge und Wartung aus.

den Phasen. Dort gibt es den Ausschlag zur Empfehlung von überwiegend älteren Mitarbeitern in den Teams, die sich um die Implementation und die Wartung kümmern. Sozialkompetenz im Umgang mit dem Kunden ist in dieser Entwicklungsphase bedeutsam, da erneut der für die Akzeptanz des Produkts wichtige Kundenkontakt auftritt.

5 Eigenverantwortung und Unternehmensverantwortung in der Mitarbeiterentwicklung („Laufbahnplanung")

Von den Unternehmen wird häufig die Eigenständigkeit und Eigenverantwortlichkeit des Softwareentwicklers in Bezug auf Weiterbildung und Laufbahnplanung in einem sich so schnell wandelnden Bereich wie der Informationstechnologien betont. Auch aus der Perspektive vieler jüngerer und älterer Entwickler ist die Weiterbildung eine im Eigeninteresse des Entwicklers selbst liegende Aufgabe. Dementsprechend findet man in der Regel, dass in einem eng spezialisierten Bereich die Weiterbildung durch Lesen anwendungsbezogener Fachliteratur und durch den Besuch von Kursen für aktuell benötigte Anwendungen vorherrscht. (Ab einem Alter von über 50-Jahren nimmt diese fachbezogene Weiterbildungsneigung aufgrund sinkender Motivation und auch verringerter Unterstützung durch die Unternehmen deutlich ab.)

Es geht darum, die Gewinnung von Erfahrung zu ermöglichen und zu fördern sowie bei älteren Entwicklern verstärkt zu nutzen. Ideal wären geeignete Rahmenbedingungen für die längerfristige Laufbahnentwicklung:

- Tätigkeitswechsel
- Absprache von Weiterbildung, gerade auch im überfachlichen Bereich
- Hinzunahme weiterer Tätigkeitselemente zu der eng spezialisierten Kerntätigkeit
- Überleitung in stärker erfahrungsgeleiteten Tätigkeitsbereichen
- altersgemischte Teams, die nach Erfahrungsstufen aufgebaut sind

Der Umstand, dass eine definitive Laufbahnstrategie in sich stark wandelnden Computerberufen für einen solch langen Zeitraum gar nicht festgelegt werden kann, dient den Mitarbeitern und Personalverantwortlichen in Softwareunternehmen leicht als Ausrede dafür, sich gar keine realistischen Perspektiven zurechtzulegen. Diese werden aber zur generellen Orientierung benötigt und sollten gerade den schnellen technologischen Wandel berücksichtigen.

Somit wird sich der Mitarbeiter in der Softwareentwicklung nicht nur Gedanken über die Arbeit in einem eng spezialisierten Arbeitsbereich machen, sondern längerfristig nach Tätigkeiten streben, in denen übergreifende Erfahrungselemente bedeutsamer sind. Dann werden auch begeisterte Programmierer eine Weiterbildung in überfachlichen Bereichen nicht mehr ablehnen und sich beispielsweise stärker für Tätigkeiten als Projektleiter interessieren, da dort wichtige, überfachliche Erfahrungen gesammelt werden können.

Auch der Kundenkontakt könnte so zu einer größeren Priorität und Akzeptanz beim fachlich interessierten Mitarbeiter gelangen, der erkennt, dass auf Dauer eine gute Position auf dem Arbeitsmarkt selbst als IT-Experte nur durch praxiserprobte Zusatzqualifikationen erhalten werden kann.

Der in diesem Beitrag benutzte Begriff der Laufbahnentwicklung unterscheidet sich vom Begriff „Karrierewege", da er nicht am voraussehbaren Durchlaufen stringenter Hierarchieebenen orientiert ist. Unter Laufbahnentwicklung soll daher die Festlegung auf ein *Spektrum* möglicher weite-

rer Wege verstanden werden. Damit wird das Offenhalten und Fördern dieser bestimmten Optionen für die weitere berufliche Entwicklung betont.[8] Die Vorteile der Laufbahnplanung liegen darin, dass

- eine langfristige Perspektive gegeben wird,
- eine relative Einengung auf eine bestimmte Tätigkeitsperspektive jenseits der hochspezialisierten Tätigkeit vorgenommen wird, die in der Regel das erste Jahrzehnt der Berufstätigkeit des hochqualifizierten Mitarbeiters im Bereich der Softwareentwicklung ausfüllt.

Primär soll ein begrenztes Spektrum von Weiterentwicklungsmöglichkeiten für die berufliche Laufbahn des Softwareentwicklers offen gehalten werden, wozu das Festlegen und Erreichen bestimmter Meilensteine[9] der Weiterbildung nach der Erstausbildung nur als ein Planungsinstrument unter mehreren verstanden werden kann. Als weiterer Aspekt ist die kontinuierliche Verarbeitung gemachter Erfahrungen ebenso bedeutsam. Dasselbe gilt auch für die Frage, wie lange die bisherige Tätigkeit noch herausfordernd genug ist. Diese Frage muss im Bewusstsein der älter werdenden Softwareentwickler verankert werden.

Eine längerfristig orientierte Laufbahnentwicklung trägt dazu bei, von vornherein durch entsprechende Maßnahmen eine fachliche Nischenbildung zu vermeiden.[10] Es handelt sich dabei um Maßnahmen, die eine kontinuierliche Entwicklung des Mitarbeiters vorsehen. Diese muss vom Unternehmen angestoßen und gefördert werden. Besonders wichtig erscheint dabei die Laufbahnplanung auch für diejenigen Mitarbeiter, die zur Zeit in einer starken Wachstumsphase in expandierenden Software-Häusern eingestellt werden. Denn sie werden die älteren Mitarbeiter von morgen sein.

Deshalb besteht die Möglichkeit und Notwendigkeit, gleich von Beginn der beruflichen Laufbahnen ein starkes Bewusstsein für die Ziele und die Probleme der Laufbahnentwicklung zu schaffen. Der Unternehmensführung und dem Mitarbeiter muss bewusst sein, dass es in einem sich so schnell wandelnden Ingenieurbereich auf Dauer nicht ausreicht, eine Spezialisierungsnische zu suchen.

Zusammenfassend lässt sich sagen, dass durch die Maßnahmen der Laufbahnplanung folgende Ziele verfolgt werden:

- Strukturierung der Weiterbildungsmaßnahmen
- Ermöglichung und Verarbeitung von Erfahrungen aus dem Geschäftsfeld, den spezifischen Bedingungen im Unternehmen und aus dem Kundenkontakt
- Ermöglichung der berufsbiographischen Weiterentwicklung der Mitarbeiter in der Softwareentwicklung

Von Unternehmenspraktikern wurden als Gütekriterien für die Zielerreichung und für die Anwendbarkeit der Laufbahnmodelle benannt:

1) Ermöglichung einer mittelfristigen Weiterbildungsplanung,
2) Umsetzbarkeit (z.B. der Schaffung der entsprechenden Stellen).

[8] Vgl. Behrens 1993 und 1996.

[9] Diese Meilensteine geben Zeitpunkte für die Neuerwerbung von Fachwissen an und konkretisieren somit Zwischenziele, zu denen Maßnahmen zur Erlangung einer zukunftsfähigen Qualifikation durchgeführt werden. Diese längerfristig angelegte Planung von bestimmten Weiterbildungsmaßnahmen ist ein wichtiger Aspekt.

[10] Vgl. Pack 1999, S. 19-23.

- Ermöglichung und Durchführung lebenslangen Lernens und des Erwerbs von Spezialisierungen in einem zweiten, zukunftsfähigen Softwarebereich
- Erhalt der Fähigkeit abstrakten Denkens, um Übergänge zu anderen Konzepten der Hard- und Software zu ermöglichen
- Verwerten speziell der Erfahrung im Rahmen des Kundenkontaktes im Zusammenhang mit dem Erlernen von unternehmerischem Denken, ggf. bis hin zur Ermöglichung der gewerblichen Selbständigkeit des Entwicklers
- Verhandlung und Vereinbarung der Laufbahnplanung aufgrund der Möglichkeiten des Unternehmens und der Wünsche des Entwicklers[11]
- Laufbahnplanung kann dazu beitragen, Kompetenzträger stärker an das Unternehmen zu binden[12]

Für das Personalmanagement sollte im Zusammenhang mit der Laufbahnentwicklung des Softwareentwicklers[13] das Ziel der Erhaltung der Arbeitsmarktfähigkeit des Entwicklers zusammen mit der Unternehmensaufgabe einer Steigerung der Innovationsfähigkeit Priorität haben.

Schon aus diesen Zielstellungen ergeben sich Schlussfolgerungen für die primären Ziele der Laufbahnplanung. Die Weiterbildung muss sowohl an den Vorstellungen des Mitarbeiters über seine Laufbahnentwicklung ausgerichtet werden wie an den Möglichkeiten im Unternehmen. Dazu ist hilfreich, wenn auch die Abteilung, in der der Softwareentwickler eingesetzt ist, eine längerfristige strategische Ausrichtung besitzt und entsprechende Ideen und Vorentscheidungen in Form von Leitbildern für einen Zeitraum von 10 Jahren als Orientierung auch für die Laufbahnentwicklung des Mitarbeiters dienen können.

Weiterhin müssen vom Personalmanagement Vorstellungen darüber vorliegen, welche Übergänge von einer zu einer anderen Spezialisierung denkbar oder wo die Übernahme einer erweiterten Verantwortung im Rahmen einer Fachkarriere im Unternehmen möglich sein werden.[14] Aufbau-

---

[11] Die Laufbahnplanung dient der Orientierung über Möglichkeiten des Fortschreitens zu neuen Tätigkeiten und Stellen. Diese sind nicht nach vorgegebenen Betriebszugehörigkeitsdauern zu erreichen, sondern müssen von den Mitarbeitern angestrebt werden. Es gibt keine Aufstiegsautomatik und keine Erfolgsgarantie.

[12] Mitarbeiter im Bereich hochqualifizierter Dienstleistung, zumal im Tätigkeitsbereich der Software-Entwicklung, werden nicht allein durch Aspekte des formalen Status im Unternehmen gehalten, sondern auch wesentlich durch informelle Aspekte. Man will gefragt werden und als fachliche Autorität im Team anerkannt sein.

[13] Leider konnten die Möglichkeiten einer Laufbahnentwicklung für Fachinformatiker (also für Personen mit Qualifikationen unterhalb eines Hochschulabschlusses) nicht thematisiert werden, da in den untersuchten Unternehmen keine Gelegenheit bestand, diese zu interviewen. Deshalb beziehen sich diese Überlegungen allein auf Fallstudienergebnisse von Hochschulabgängern (sei es von Informatikern oder von Physikern, Ingenieuren, etc.). Zudem wäre die weitere Entwicklung in diesem jungen, erst ein paar Jahre bestehenden Ausbildungsgang erst noch abzuwarten bzw. in ähnlicher Weise in eine Laufbahnplanung einzubeziehen, wie sie hier vorgestellt und als Maßnahme vorgeschlagen wird.

[14] Es sei nochmals betont, dass der Entwurf von Laufbahnmodellen nur hinweisenden Charakter haben kann und auch nicht mehr festlegen bzw. leisten *soll*. In diesem Sinne wird ein solches Modell als Hilfsmittel des Personal- und Weiterbildungsmanagements verstanden. In den Unternehmen bestehen teilweise bereits einfache Orientierungen (z.B. nach Entwicklung/Systembetreuung, Beratung, Vertrieb), die von den Unternehmensvertretern z.T. als Lösungsmöglichkeit angesehen wurden. Diese Einteilungen stellen zwar einen ersten Ansatz dar, reichen aber für die Unterstützung des primär fachlich interessierten Softwareentwicklers nicht aus. Ihm liegt an einer Differenzierung seiner Fachkarriere, deren Ausgestaltung weitgehend unternehmensspezifisch zu definieren ist.

end auf diesen Einsichten lassen sich als wesentliche Teilziele formulieren, die man in primär empfehlenswerte Maßnahmen umsetzen kann:

- Abstimmung der Mitarbeiterentwicklung zwischen Entwickler und Vorgesetztem,
- längerfristige Vorbereitung des Wechsels zu anderen Tätigkeitsbereichen,
- ständige Weiterbildung für das Spezialgebiet,
- Vorbereitung auf neue Konzepte und Paradigmen.

Das Personalmanagement muss die Möglichkeit erhalten, sowohl eine Warnfunktion und eine Weiterbildungsfunktion zu erfüllen. Dequalifizierungstendenzen müssen frühzeitig erkannt werden (Warnfunktion) und Weiterbildung ist als Zielstellung des Personalmanagements zu betreiben und in eine längerfristig angelegte Laufbahnentwicklung einzubeziehen. Dazu sind für die Personalverantwortlichen entsprechende Instrumente zu schaffen. Nach den Ergebnissen der Fallstudien muss man aber bereits einen gewissen Erfolg darin sehen, wenn in Zukunft zunächst einmal die Standard-Hilfsmittel ernster genommen werden. Dieses kann und muss geschehen bezüglich folgender Punkte:

- Vielfach bestehen Richtlinien und Formulare für Personalgespräche, die in der Regel einmal im Jahr vom Mitarbeiter und seinem direkten Vorgesetzten durchgeführt werden. Oft wird das Ergebnis dabei auch schriftlich fixiert. Trotzdem ist es vielfach noch so, dass erst beim Auftreten größerer Probleme Zuständige des Personalmanagements hinzugezogen werden. Hier braucht man Möglichkeiten des frühzeitigeren Eingreifens. Generelle Ergebnisse der Gespräche müssen ausgewertet werden und um dieses zu können, könnten z.B. Laufbahnschemata entwickelt und genutzt werden. Diese sollen nicht direkt einzelne Stellen beschreiben, sondern vielmehr Maßnahmen erleichtern, die dem Erhalt der Qualifikation dienen.
- Ein wichtiges Auswertehilfsmittel kann in größeren Unternehmen eine Skill-Datenbank sein, in der Qualifikationen der Mitarbeiter verzeichnet sind. Besonders hierfür sind natürlich Schemata zur Auswertung erforderlich, die das zu sammelnde Material strukturieren und auch in größeren Unternehmen handhabbar halten.
- Eine fortschrittliche Variante der Weiterbildungsförderung ist das Angebot von Beratung über individuell zugeschnittene Weiterbildungsangebote (Weiterbildungsmanager). In Großunternehmen ist möglicherweise sogar die Einrichtung eines Profit-Centers rentabel, das intern entsprechende Dienstleistungen anbietet.

6 Tätigkeitsfelder für ältere Softwareentwickler

Zusammen mit der auf uns zukommenden demografischen Entwicklung (in 15 Jahren wird der Anteil der über 45-jährigen an der Erwerbsbevölkerung von heute 35 % auf 45 % gestiegen sein) lässt sich erkennen, dass danach gestrebt werden sollte, in Zukunft mehr ältere Software-Spezialisten in diesem Bereich zu halten. Sicherlich wird deren Hauptaufgabe nicht in der Programmiertätigkeit liegen, sondern Tätigkeiten betreffen, welche die Softwareentwicklung begleiten bzw. ihr in konzeptioneller wie organisatorischer Hinsicht vorangehen. Diese Bereiche dürften aber immer wichtiger werden. Eine Aufwertung der Kompetenzen bezüglich Kundenorientierung und Innovationsfähigkeit sowie die Nutzung entsprechender Erfahrungen älterer Softwareentwickler gewinnt zunehmend an Bedeutung und sollte stärker gefördert werden.

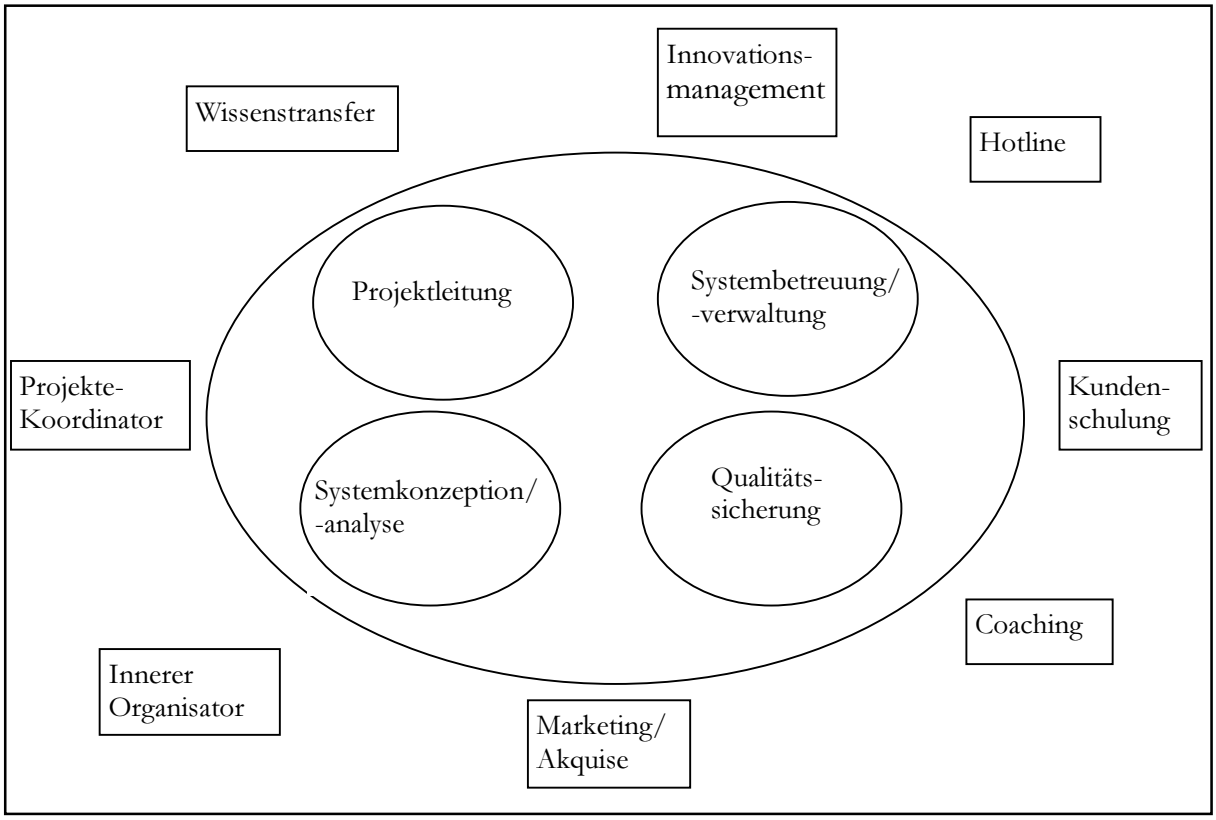

Abbildung: Tätigkeitsbereiche im engeren und weiteren Umfeld der Softwareentwicklung, die besonders gut für erfahrene, ältere Entwickler geeignet erscheinen.

In der Abbildung werden solche möglichen Tätigkeitsbereiche benannt, in denen für die über 40-jährigen Entwickler besondere Entwicklungs- und Entfaltungsmöglichkeiten gesehen werden.[15] Eine Vorbereitung auf solche Stellen, insbesondere mit höheren Managementanteilen wie Projektleiter und Projekt-Koordinator, sollte günstigerweise bereits längerfristig im Rahmen der Mitarbeiterentwicklung durch Weiterbildungsmaßnahmen und Tätigkeitswechsel sowie stetige Ausweitung des Verantwortungsbereiches angestrebt werden. Bei den hier vorgestellten Tätigkeitsbereichen interessiert darüber hinaus aber besonders, ob ein Einsatz trotz einer bestehenden Vernachlässigung der vorausgegangenen Personalentwicklung der betroffenen Mitarbeiter noch möglich erscheint.

Es sollen hauptsächlich diejenigen Rollen ausgewählt werden, die für die Entwickler aufgrund ihrer durch Berufs- und Lebensalter positiv gewachsenen Leistungspotentiale besonders gut geeignet sind. Die vorgeschlagenen und mit Unternehmenspraktikern diskutierten Aufgaben sind, wie auch der Begriff der Rollen bereits andeutet, nicht unbedingt und in jedem Fall als Vollzeitjobs zu verstehen. Je nach Unternehmen und Mitarbeiter muss diesbezüglich individuell entschieden werden.

Für die Auswahl von Stellenbewerbern aus den Reihen der älteren Entwickler ist das Personalmanagement insbesondere an besonders kritischen und bei älteren primär ausbau- und einsetzbaren Fähigkeiten interessiert. In den von uns durchgeführten Tiefenfallstudien in drei

[15] Für eine ausführliche Diskussion der Tätigkeitsbereiche siehe Berndes et al. 2001, Kapitel 5.

Softwareunternehmen unterschiedlicher Größe ergaben sich die psychische Belastbarkeit, die soziale Kompetenz und die überfachliche Erfahrung als wesentliche Kompetenzen.

7 Fazit

Ältere Softwareentwickler benötigen in der Regel keine „Schonarbeitsplätze", sondern die Möglichkeit, in schrittweise ausgeweiteten Teilbereichen erworbene Erfahrung einbringen zu können. Weder sollte eine zu enge Spezialisierung über einen langen Zeitraum beibehalten werden (und lang sind in der Softwareentwicklung schon Zeiten von 5-10 Jahren) noch ein für alle gültiger Anspruch der Weiterentwicklung zu Managementtätigkeiten gefordert werden. Aber eine speziell vorteilhafte Möglichkeit für einen größer werdenden Teil älterer Mitarbeiter sollte dabei die Gewinnung und Entfaltung von Erfahrung mit einer kundenorientierten Nutzung neuer Technologien darstellen. Wie schon eingangs betont, ist es für den innovativen Tätigkeitsbereich der Softwareentwicklung besonders bedeutsam (und in Anbetracht der Tendenzen am Dienstleistungsstandort Deutschland in Zukunft vorrangig), die in schnellem Wandel befindlichen technologischen Möglichkeiten jeweils auf mögliche Vorteile für den Kundenkreis hin auszuloten und entsprechende, nicht allein technologiegetriebene Innovationen zu ermöglichen.

Literatur

Behrens, J. (1993): Laufbahngestaltung für Service-Ingenieure im Außendienst von Computerfirmen. In: Bullinger H-J et al. (Hrsg.): Alter und Erwerbsarbeit der Zukunft. Berlin, Heidelberg, New York: Springer, S. 227-232

Behrens, J. (1996): Vorausschauende Personalpolitik: Laufbahngestaltung als neue Aufgabe des Arbeitsschutzes. In: Frerichs F (Hrsg.): Ältere Arbeitnehmer im Demographischen Wandel: Qualifizierungsmodelle und Eingliederungsstrategien. Dortmunder Beiträge zur Sozial- und Gesellschaftspolitik Bd. 7, Münster: Lit, S. 123-139

Berndes, S.; Lünstroth, U. (1998): INVAS (Innovationen bei veränderten Altersstrukturen) - Fallstudien aus der Software-Entwicklung). Forum der Forschung. BTU Cottbus: Eigenverlag. 4. Jhg., Heft 6, S. 26-33

Berndes, S.; Lünstroth, U. (2000): Technology, Organisation and Qualifications in Software Development. In: Coakes, E.; Lloyd-Jones, R.; Willis, D. (Hrsg.): The New Sociotech, Computer Supported Co-operative Work (CSCW), London: Springer.

Berndes, S., Kornwachs, K., Lünstroth, U. (2001): Software-Entwicklung - Erfahrung und Innovation. Ein Blick auf demographische Veränderungen. Berlin, Heidelberg, New York: Springer. (im Druck).

Littek, W., Rammert, W., Wachtler, G. (Hrsg.) (1983): Einführung in die Arbeits- und Industriesoziologie. 2. Aufl., Frankfurt/Main: Campus.

Pack, J.; Buck, H.; Kistler, E.; Mendius, H. G.; Morschhäuser, M; Wolff, H. (1999): Zukunftsreport demographischer Wandel: Innovationsfähigkeit in einer alternden Gesellschaft. BMBF, Bonn

Ältere Softwareentwickler als begehrte Mitarbeiter

Ein Kommentar zum Beitrag von Uwe Lünstroth[1]

Wolfgang Häcker
IT-Services & Solutions GmbH
Rathausstr. 7
09111 Chemnitz
Wolfgang.Haecker@de.ibm.com

Ich komme von der Firma IT- Services & Solutions GmbH. Wir haben mehr als 1500 Mitarbeiter, 16 Niederlassungen deutschlandweit. Unser Hauptsitz ist in Chemnitz. Wir bieten ausschließlich Dienstleistungen im Softwarebereich an. Das Offering, das ich vertrete, nennt sich Software Application Services und befasst sich mit der Entwicklung von Anwendungen, deren Wartung und Pflege, mit der Migration von Anwendungen, mit der Entwicklung und Integration webbasierter Anwendungen. Auf diesem Gebiet sind in Chemnitz 125 Personen tätig, deutschlandweit sind es mehr als 400. Nun zu den älteren Softwareentwicklern: Ich denke, ich darf das etwas drastischer ansprechen, ich gehöre nicht zu den älteren Softwareentwicklern, sondern zu den alten. Im Beitrag von Herrn Lünstroth tauchten die Umschüler der 70er Jahre auf. Diese Generation Entwickler war ganz einfach der Zeit geschuldet. Zum damaligen Zeitpunkt gab es kein Informatikstudium, sondern nur das der Informationstechnologie und Informationstechnik. Beides waren mehr oder weniger hardwarenahe Studiengänge. Sie waren kaum von der Softwareentwicklung geprägt. Deshalb musste zu solchen Maßnahmen gegriffen werden. Die Umschüler aus den 70er Jahren sind heute alle älter als 45 Jahre. Aber sie können prima Assembler, sie können prima PL/1, sie können prima COBOL programmieren.

62% meiner Mitarbeiter sind älter als 45 Jahre! 25% sind älter als 55! Was tun wir mit ihnen? Herr Lünstroth erwähnte in seinem Vortrag, dass es auch noch ältere Systeme und Programme gibt, deren Betrieb sich ökonomisch betrachtet durchaus noch rechnet, die aber bei den jüngeren Entwicklern das Image des altmodischen haben und deshalb uninteressant sind. Ich möchte betonen, dass die Wartung und Pflege derartiger Anwendungssysteme ein spezielles Geschäftsfeld in Deutschland ist. Und ich darf ihnen sagen, dass man auf diesem Gebiet gutes Geschäft machen kann. Ich bin in der Wirtschaft tätig. Dort ist dieses Argument der erste Beweggrund, sich einem Aufgabengebiet zuzuwenden. Konkret heißt das: Es gibt in Deutschland eine Menge großer Kunden - Banken, Versicherungen und auch andere - die derartige Anwendungssysteme in Betrieb haben. Diese Systeme sind in den 70/80er Jahren entstanden und die Mitarbeiter, die sie entwickelt haben, gehören zu den älteren Softwareentwicklern. Im Rahmen unseres Serviceangebotes „Anwendungsbetreuung“ übernehmen wir verantwortlich die Wartung und Pflege der Anwendungen des Kunden und entwickeln sie bei Erfordernis weiter. Das ist beispielsweise erforderlich, wenn sich gesetzliche, tarifliche oder andere Grundlagen ändern. Warum sollte also

[1] Dieser Beitrag basiert auf den Tonbandaufzeichnungen während des Workshops.

ein Kunde derartig bewährte Anwendungen ablösen? Nur weil sich die Programmiertechniken geändert haben? Das macht kein Kunde. Jeder Kunde fragt sie: Wie viel kann ich mit einem neuen System mehr Geschäft machen. Deshalb finde ich es wichtig, Innovationen nicht nur von der technologischen Seite zu betrachten, sondern wir sollten als erstes fragen, welchen Nutzen bringen die Innovationen dem Kunden.

Um das Geschäftsfeld auch absichern zu können, muss ich (junge) Leute finden, die sich dieser Aufgabe annehmen. Und hier wirkt der von Herrn Lünstroth vorgeschlagene Altersruhestand oder Vorruhestand für mich als Bedrohung, denn dann verliere ich mit einem Mal 25% meiner Mitarbeiter. Leider bietet mir die Universität keine auf Mainframes ausgebildeten Informatiker an. Sollte es doch welche geben, dann wollen sie sich mit Sicherheit nicht in ihrer alltäglichen Arbeit damit beschäftigen. Was tun wir also? Wir bilden analog zu den Umschulungen der 70er Jahre junge Quereinsteiger - meist mit Informatik-Grundausbildung - auf diesen Gebieten aus. Und wir haben tatsächlich in den letzten fünf Jahren 80 Anwendungsentwickler und 50 Systemingenieure auf Mainframes ausbilden lassen und setzen sie erfolgreich in unseren Projekten ein. Das sind alles intelligente Leute, die leider etwas studiert hatten, das in der Wirtschaft nicht so nachgefragt war (ist). „Brotlose Künstler" wie Mathematiker, Physiker, Lehrer. Diese Leute haben eine Ausbildung genossen mit JCL, ISPF, REXX, DB2, IMS, PL/1, COBOL. Wir haben sie natürlich auch für die neuen Architekturen ausgebildet, d.h. auch in Middleware, MQ-Series zum Beispiel, damit sie auch Verbindungen zur neuen „Welt" herstellen können. Nicht ohne Grund gibt es den Begriff „web to host". Damit haben diese Leute gleichzeitig eine Verbindung in die Zukunft, in die sie sich weiter entwickeln können, d.h. die Ausbildung stellt in diesem Sinne keine Sackgasse dar. 12% meiner Mitarbeiter in Chemnitz sind so ausgebildete Leute und wir starten Anfang des nächsten Jahres eine weitere Klasse mit 16 Personen. Voraussetzung ist, dass die Bewerber ein abgeschlossenes Hochschulstudium haben und strukturiert denken können.

Problematisch ist für mich das von Herrn Lünstroth vorgestellte Laufbahnmodell. Es endet im wesentlichen in Management- oder Projektleitungstätigkeiten. Das Verhältnis von Projektleitern zu Mitarbeitern ist bei uns etwa 1:7. Dem stehen die 62% meiner Entwickler gegenüber, die älter als 45 Jahre. Es können also nicht alle Projektleiter werden. Es muss auch noch welche geben, die programmieren.

Ich kann zudem ihre Aussagen nicht bestätigen, dass es schwer ist, in Teams eine gute Mischung zwischen jüngeren und älteren Softwareentwicklern hinzubekommen. Im Bereich der webbasierten Anwendungsentwicklung sind 40% der Mitarbeiter jünger als 30 Jahre und 40% sind älter als 35 Jahre. Älter als 40 Jahre sind 16% und älter als 50 Jahre sind 4%. Damit haben wir gute Erfahrungen gemacht.

In einem Service-Unternehmen sind die Mitarbeiter das Kapital des Unternehmens. Da kann Weiterbildung keine Privatangelegenheit sein. Die Firma IT Services & Solutions GmbH gibt jährlich ca. 3% des Umsatzes für die Ausbildung ihrer Mitarbeiter aus. Genauso legen wir wert auf ein Bezahlungsmodell, welches die Ausbildung und den Bildungsstand mit einbezieht. Entscheidend ist nicht nur die konkret ausgeübte Tätigkeit, sondern auch, wie breit ich qualifiziert bin und wie breit ich demzufolge im Unternehmen eingesetzt werden kann. In dieser Hinsicht gibt es vermutlich einen Unterschied zwischen Unternehmen, die Software entwickeln oder Dienstleistungen für Software anbieten und Unternehmen, die einen Bereich Softwareentwicklung haben, deren Kerngeschäft aber auf anderen Gebieten abgewickelt wird.

Abschließend möchte ich einen Spruch anbringen, dessen Urheber ich leider nicht kenne, der meiner Meinung nach jedoch viel Wahrheit beinhaltet: „Man wird mit zunehmenden Alter nicht weiser, sondern vorsichtiger." Er bestätigt also die Aussagen von Herrn Lünstroth über ältere Softwareentwickler, dass sie vorsichtiger sind und bremsen. Manchmal ist das jedoch auch gut so.

Anwendungsorientierung und empirische Forschung in der Softwaretechnik

Schritte zu einer kooperativen Methodenentwicklung

Yvonne Dittrich
Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Softcenter
S37225 Ronneby, Schweden
yvonne.dittrich@bth.se

1 Einleitung

Empirische Forschung und Kooperation mit der Industrie werden für die Softwaretechnik immer wichtiger. Wurde Softwaretechnik 1969 noch gegründet, um der Praxis aus der Softwarekrise zu helfen (Bauer 1993), hat sich das Bild heute gewandelt. Heute existiert eine diversifizierte und oftmals gut funktionierende Praxis. Methoden und Werkzeuge aus der Forschung müssen sich in dieser Praxis bewähren. Sinnvoll ist daher, sie zusammen mit denjenigen zu entwickeln, die sie später vielleicht anwenden werden. Auf der anderen Seite haben sich durch die Erfahrung aus der Praxis auch die Problemstellungen geändert. Will man relevante Probleme erforschen, ist der Kontakt zur Praxis unumgänglich.

Anwendbarkeit von Software ist immer noch eine der problematischsten Eigenschaften. Mehr als 70% der Softwareentwicklungskosten entfallen auf Wartung, (Nosek, Palvia 1990), ein Grossteil davon auf ‚perfektive' Wartung, die Anpassung von Software an sich ändernde oder sich erst in der Nutzung herauskristallisierende Anforderungen. Evolutionäre Softwareentwicklung (Floyd et al. 1989a) und Methoden, die die Teilnahme von künftigen Anwendern bei der Gestaltung von Software zulassen, sind mittlerweile fast 15 Jahre alt (Floyd et al. 1989b). Trotzdem werden sie in industriellen Kontexten kaum, zumindest nicht bewusst, eingesetzt. Mittels empirischer Forschung, die eine kooperative Methodenentwicklung unterstützt, kann es gelingen, anwendbarere Methoden zu entwickeln.

Mit diesem Artikel möchte ich unseren Ansatz zu empirischer Forschung in der Softwaretechnik zur Diskussion stellen. In den letzten Jahren haben wir in verschiedenen Projekten Erfahrungen mit qualitativer empirischer Forschung und darauf aufbauenden Interventionen gesammelt. Schwerpunkt war dabei, wie Methoden der anwendungsorientierten Softwareentwicklung an die industrielle Praxis angepasst werden können und wie ihre Anwendung unterstützt werden kann. Wir wenden dabei Grundsätze der benutzungsorientierten Softwareentwicklung auf die Entwicklung von Methoden für Softwareentwickler an. Methoden werden dabei als Werkzeuge

betrachtet, die genauso an ihren Nutzungskontext angepasst werden müssen wie Software an den ihren (Floyd/Züllighoven 1997).

Im nächsten Abschnitt diskutiere ich verschiedene Ansätze empirischer Forschung um daraus unseren Ansatz zu motivieren. Anschließend stelle ich ein Kooperationsprojekt mit einem schwedischen Telekomunternehmen vor, und veranschauliche daran unser Vorgehen. Abschließend stelle ich erste Ergebnisse vor. Der Beitrag stellt eine Weiterentwicklung der Ansätze aus (Dittrich 2002) dar.

2 Anwendungsorientierung...

Methoden, die anwendungsorientierte Gestaltung in den Mittelpunkt von Softwareentwicklung stellen, werden seit Mitte der 70er hauptsächlich in Skandinavien entwickelt. Mit der 2jährlich stattfindenden Participatory Design Konferenz wurden die Ergebnisse und Ansätze dann in den USA und damit über Skandinavien und Finnland hinaus bekannt. Forschung findet normalerweise als Aktionsforschung statt. Forscher übernehmen die Rolle von Softwareentwicklern oder von „Participatory Designern" (Kensing 2000) gegenüber den künftigen Benutzern. Sie entwickeln und erproben Methoden und Vorgehensweisen, analysieren und reflektieren ihre Erfahrungen. Die Erfahrungsberichte und die darauf aufbauenden theoretischen Arbeiten zeigen, dass die Beteiligung von künftigen Anwendern bei der Softwareentwicklung möglich ist, und dass dadurch die Anwendbarkeit von Software verbessert werden kann. Ich kann die mittlerweile recht umfangreiche Literatur an dieser Stelle nicht ausreichend würdigen.[1] Die folgenden Punkte fassen für mich die wichtigsten Grundpfeiler der anwendungsorientierten Softwareentwicklung zusammen:

- Kooperation zwischen künftigen Benutzern und Entwicklern ist der Kern bei der Gestaltung der Software und deren Einbettung in den Nutzungskontext. Die künftigen Anwender fungieren dabei als Anwendungsexperten, die einen wichtigen Beitrag zu der Antizipation künftiger Arbeitsformen und davon ausgehend der Einbettung von Computeranwendungen leisten (Ehn 1988). Maßnahmen und Organisationsformen, um eine gleichberechtigte Kooperation zu ermöglichen, sind zum Beispiel gemeinsame Workshops, Anwenderqualifikation, aber auch eine Projektorganisation, die Anwendern erlaubt, sich eine unabhängig Meinung zu Designvorschlägen zu bilden. Erfahrungen über Kooperation in heterogenen und räumlich verteilten Anwendergruppen wurden dokumentiert und haben zur Entwicklung von entsprechenden Methoden beigetragen (Krabbel/Wetzel 1998).
- Normale Sprache und konkrete Repräsentationen, wie zum Beispiel ‚rich pictures', ‚mockups', und Prototypen als Designmedien sowie ‚boundary objects' sind methodische Werkzeugen, die sowohl von Benutzern als auch Softwareentwicklern in bezug auf den jeweiligen Fachzusammenhang interpretiert werden können (Ehn 1988). Ethnographische und andere sozialwissenschaftliche Arbeitsplatzstudien können im partizipativen Prozess eingesetzt werden, um ein gegenseitiges Verständnis zu unterstützen (Karasti 2000).
- Iterative und evolutionäre Prozessmodelle ermöglichen es mit Hilfe von frühzeitigem Feedback die mit der Softwareeinführung einhergehenden Veränderungen der Arbeitsorganisation

[1] Siehe (Ehn 1988), (Floyd et al. 1989b), und das Scandinavian Journal for Information Systems 1998 für verschiedene Übersichtsdarstellungen.

und -praxis in die weitere Entwicklung mit einzubeziehen (Floyd et al. 1989a, Bürkle et al. 1995, Christensen et al. 1998).

Neben einem reichhaltigen Methodenrepertoire hat die Forschung zur anwendungsorientierten Softwareentwicklung auch zu einem reflektierteren Verständnis der Wechselwirkungen zwischen Nutzungskontext und Computerunterstützung beigetragen. Besonders Bødker hat in ihren Texten herausgearbeitet, wie sich Gestaltungsprozesse in den Anwendungsalltag hinein fortsetzen und wie die Anwendung zum Beispiel von frühen Prototypen in der Gestaltung von Software wirksam werden kann (Bødker 1999). Dabei wird deutlich, dass Nutzungsqualitäten von Software wie Aufgabenangemessenheit und Anwendbarkeit im Zusammenhang von Software und Anwendungskontext begründet sind. Sie sind keine unabhängigen Attribute von Programmen an sich.

Die reichhaltigen Erfahrungen mit unterschiedlichen Methoden, Prozessmodellen und Repräsentationsformen weisen darauf hin, dass es keinen idealen Prozess und keine kanonische Form der anwendungsorientierten Softwareentwicklung gibt. Vielmehr müssen die Werkzeuge der Softwareentwickler - eben jene Methoden, Vorgehensweisen und Repräsentationsformen - dem jeweiligen Anwendungs- und Entwicklungskontext angepasst werden. Hierin mag einer der Gründe für die zögerliche Verbreitung der Methoden und Vorgehensmodelle in der kommerziellen Softwareentwicklung liegen: Die Methoden und Modelle wurden in Forschungsprojekten entwickelt und für nicht-kommerzielle Anwender optimiert. Ist man an einer weiteren Verbreitung von anwendungsorientierter Softwareentwicklung interessiert, muss die bestehende Forschung ergänzt werden durch Forschung, die die konkreten Bedingungen zum Gegenstand hat, unter denen Software in kommerziellen Kontexten entwickelt wird, und die die Anpassung von Methoden, Prozessmodellen und Repräsentationsformen an diese Entwicklungskontexte erforscht.

3 ... und empirische Forschung in der Softwaretechnik

Forschung, die die Anpassung von Methoden an kommerzielle Praxis zum Gegenstand hat, diese evaluieren und darauf aufbauend Methoden weiterentwickeln will, verlangt nach einem empirischen Ansatz. Die ersten systematischen Ansätze zu empirischer Forschung haben sich Anfang der 80er herauskristallisiert. Dabei bildeten sich verschiedene Schulen heraus, die sich erst in letzter Zeit aufeinander beziehen. Die verschiedenen Ansätze unterscheiden sich in der Wahl empirischer Methoden, darin, wie Hypothesen generiert oder Erkenntnisse entwickelt werden, und darin, wie Verbesserungsvorschläge erarbeitet und begründet werden. Mit diesen erkenntnistheoretischen Unterschieden geht auch ein unterschiedliches Verhältnis zu den Projekten einher, die Gegenstand der empirischen Forschung sind. Die Beziehung zu den jeweiligen Mitarbeitern wird unterschiedlich gehandhabt.

Im folgenden beschreibe ich drei verschiedene Ansätze der empirischen Forschung. Für meine Frage im Zusammenhang mit dem vorliegenden Artikel können nicht alle einschlägigen Forschergruppen gewürdigt werden. Deshalb beschränke ich mich auf prototypische Referenzen und Personen. Ziel ist es, die Verschiedenheit empirischer Ansätze in der Softwaretechnik aufzuzeigen, nicht eine vollständige Landkarte empirischer Forschung zu präsentieren.

3.1 Das Software Engineering Laboratory

Basili und das von ihm gegründete Software Engineering Laboratory verfolgen einen Forschungsansatz, der an naturwissenschaftliche Sichten und Methoden angelehnt ist (Basili 1996).

Ziel ist, mit Hilfe realitätsnaher Versuchsanordnungen Ursache-Wirkungszusammenhänge zu modellieren und damit Ansatzpunkte zur Methodenentwicklung und -verbesserung zu entwikkeln, die diese Kausalzusammenhänge ausnutzen. Die Verbesserungsvorschläge werden dann wiederum experimentell verifiziert. Für solche Experimente wird eine Versuchsumgebung, ein Labor benötigt, in dem die Rahmenbedingungen, unter denen Softwareentwicklung stattfindet, kontrolliert werden können. Industrie und Forschung können zusammen Labore aufbauen, in denen realistische Rahmenbedingungen und Problemstellungen hergestellt werden können.

Basili und Green 1994 beschreiben ein Beispiel solcher experimenteller Forschung. Die Ausgangshypothese, dass durch Lesen von Programmen und Peer Review mehr Fehler gefunden und korrigiert werden als durch Testen, wird zuerst anhand von Programmstücken getestet, in die absichtlich Fehler eingebaut wurden. Studentische Hilfskräfte und erfahrene Softwareentwickler korrigieren die Programme, teils nur mit Hilfe von Bleistift und Papier, teils am Computer mit der Möglichkeit, das Programmstück auszuführen. In einem zweiten Experiment werden in einer Lehrveranstaltung für verschiedene Studentenprojekte unterschiedliche Regeln eingeführt. Einige Projekte entwickeln ihre Software auf traditionelle Art, in einigen wird Testen durch mit Peer Review ersetzt. Als auch dieses Experiment Lesen als die bessere Methode zum Fehlersuchen bestätigt, werden in Kooperation mit Anwendungspartnern zwei Fallstudien konstruiert. Beide Methoden werden in vergleichbaren, realen Projekten angewendet und die Ergebnisse werden verglichen.

Über die konkrete Gestaltung der Zusammenarbeit mit der Praxis wird wenig berichtet. Sie scheint hauptsächlich als Zusammenarbeit mit dem Management aufgefasst zu werden. Die Projekte und ihre Mitarbeiter werden beobachtet, ihre Leistung wird gemessen, analysiert und ausgewertet. Sie scheinen weder an dem Design der Experimente und Fallstudien, noch an der Entwicklung von Verbesserungsvorschlägen oder Auswertungskriterien beteiligt zu sein.

Die in den letzten Jahren entwickelten Ansätze zum Software Process Improvement (SPI) teilen mit dem Software Engineering Laboratory Ansatz die Betonung von quantitativen Messungen als Evaluationskriterien sowie das Ziel den Softwareentwicklungsprozess in Ursache-Wirkungs-Zusammenhängen zu verstehen und damit kontrollieren zu können. Zudem nehmen sie die Existenz eines Ideals - wie zum Beispiel im Capability-Maturity-Model (Paulk et al. 1993) und eines darin vordefinierten Weges dahin an (Aaen et al. 2001). Wie oben argumentiert, hat die Forschung zur Anwendungsorientierung gezeigt, dass Anwendbarkeit nicht unbedingt quantitative erfassbar ist. Die Möglichkeit, ein Ideal des anwendungsorientierten Softwareentwicklungsprozesses zu konstruieren, ist zu bezweifeln: Die Kontexte, in denen Software angewendet und entwickelt wird, sind zu unterschiedlich, um einen gemeinsamen Maßstab daran anlegen zu können.

Sieht man von der normativen Orientierung und von den quantitativen Evaluationskriterien ab, so liefert das Verfahren, das für die Softwareverbesserung vorgeschlagen wird – Iterationen über Bestandsaufnahme, Entwicklung von Verbesserungsvorschlägen, und deren Implementation – ein geeignetes Modell für bewusste, reflektierte und zielgerichtete Intervention im Rahmen von Kooperationsprojekten zwischen Forschern und kommerziellen Softwareentwicklern.

3.2 Reflexive Softwareentwicklung

Reflexive Softwareentwicklung ist der Titel, den Lars Matthiassen seinem Ansatz gegeben hat. „The problems, challenges and opportunities involved in systems development practice are considered the starting point for systems development research. Research activities yield experience-based knowledge that leads to new and hopefully improved practices. The knowledge that is

developed is both interpretive, helping us to understand and make sense of practice, and normative, providing support for performing systems development or improving present practices." (Mathiassen 1998, 80f) Er bezieht sich dabei auf Donald Schöns reflexiver Praxis (Schön 1982). Die Projektmitglieder sind aktiv beteiligt an der Konstruktion der Forschungsergebnisse, indem sie ihre Praxis und Rationalität für die Forscher sichtbar machen und die Forschungsergebnisse in ihrer Praxis wiederum umsetzen. Mathiassen zitiert Schön: „[T]he reflective researcher cannot maintain distance from much less superiority to, the experience of practice … he must somehow gain an inside view of the experience of practice." (Mathiassen 1998, 81) Mathiassen beschreibt seinen Forschungsansatz als Aktionsforschung. Der Forscher nimmt aktiv Teil und unterstützt die Projektmitglieder zum Beispiel bei der Verbesserung ihrer Softwareentwicklungspraxis. Die Projektmitglieder unterstützen den Forscher, indem sie verschiedene Dokumentationsformen verwenden, die sowohl ihnen selbst helfen ihre Praxis zu reflektieren, als auch diese für die Forscher dokumentieren (Mathiassen 1997, Kapitel 2, 3, 4). Methoden und Prozesse, sowie deren Strukturierung und Dokumentation, werden als Werkzeuge für Softwareentwickler verstanden. Die teilnehmende Beobachtung oder auch die beobachtende Teilnahme kann mit Fallstudien und Experimenten ergänzt werden, wo es notwendig oder hilfreich erscheint (Mathiassen 1998, 81).

In (Iversen et al. 1998) wird ein Projekt zur Softwareprozessverbesserung beschrieben, in dem ein solches kooperatives Vorgehen angewendet wurde: Anstatt eine übergeordnete Norm eines in Bezug auf Kosten und Entwicklungszeit kontrollier- und optimierbaren Prozesses anzulegen, werden Probleme von Projektmanagern als Ausgangspunkt genommen. Verbesserungsvorschläge werden zusammen mit denjenigen entwickelt, die sie hinterher umsetzen sollen. In einem anderen Artikel zum selben Projekt diskutieren Nørbjerg und Nielsen, inwieweit die normativen Vorgaben des CMM auf die Bedingungen anwendbar sind, unter denen Softwareentwicklung in diesem Fall stattfindet (Nielsen/Nørbjerg 2001).

Das kooperative Vorgehen bei der Entwicklung von Verbesserungsvorschlägen lässt sich mit dem Anliegen, anwendungsorientierte Softwareentwicklung praxisfähig zu machen, besser vereinbaren, als das normorientierte SPI. Es nimmt die Erfahrung der beteiligten Softwareentwickler als Beitrag zur Prozessverbesserung ernst. Die Entwickler selbst werden Subjekt der Methodenentwicklung und -anpassung. Ihr Feedback, was hilfreich ist und was nicht, und die Erfahrungen, die mit der Implementation von Methoden gemacht wird, stellen wichtige empirische Daten für einen qualitativen Forschungsansatz dar.

3.3 Softwareentwicklung als kooperative Arbeit

Softwareentwicklung findet in der Regel im Team statt. Softwareentwicklung ist von daher auch Forschungsgegenstand, wenn es um das Verstehen von kooperativer Arbeit und deren Unterstützung - Computer Supported Cooperative Work (CSCW) - geht (Schmidt/Sharrock 1996). Empirische Forschung, die in der CSCW-Diskussion dazu vorgestellt wird, verwendet oft Methoden aus der qualitativen Sozialforschung, zum Beispiel Ethnographie oder Ethnomethodologie.

Erforscht werden zum Beispiel die Anwendung gegenständlicher Repräsentationen (Suchman/Trigg 1993), organisatorische Rahmenbedingungen und ihr Einfluss auf die Arbeitspraxis von Softwareentwicklungsteams (Button/Sharrock 1994, 1996, 1998), oder wie sich die Projektteams in Reaktion auf ihre organisatorische Umgebung entwickeln (Potts/Catledge 1996, Robertson 1996, Tellioglu, Wagner 1996). Ein weiteres Forschungsgebiet sind die Wechselwirkungen zwischen der Arbeitspraxis von Softwareentwicklern und Computerunterstützung (Grinter 1996, 1998). Softwareentwicklung als Beispiel für räumlich verteilte Arbeit ist eine

der Forschungsfragen, die eine Schnittstelle zur Softwaretechnikforschung darstellen (Hersleb et al 2000).

Das Hauptinteresse dieser Forschung ist es zu verstehen, wie Mitglieder von Softwareentwicklungsprojekten ihren Arbeitsgegenstand gemeinsam konstruieren und ihre Arbeit darauf hin koordinieren. Die Forscher, oft Sozialwissenschaftler, versuchen die Softwareentwicklung als Arbeitspraxis aus der Sicht der Softwareentwickler zu verstehen. Die Artikel wirken deshalb aus Informatiksicht, als ob sie die nach Softwaretechnikmaßstäben schlechte Praxis bewunderten, ohne die interessante Frage zu stellen, weshalb das Beibehalten einer nachteiligen Praxis leichter erscheint als die nachteiligen Gewohnheiten zu verändern.

Ethnographische und andere qualitative sozialwissenschaftliche Forschungsansätze erlauben es dem Forscher, sich ein Bild von der Praxis der Softwareentwicklung aus einer Teilnehmerperspektive zu machen. Sie helfen die kooperative Seite der Softwareentwicklung besser zu verstehen. Es ist jedoch problematisch, sozialwissenschaftliche Methoden naiv in der Softwaretechnik anzuwenden: Softwaretechnikforscher wollen - und sollen - letztendlich die beforschte Praxis verbessern. Sie sind also keine Unbeteiligten, wenn sie an einem Projekt beobachtend teilnehmen. Allein schon durch ihre Präsenz greifen sie daher in die beobachtete Praxis ein. Wessen Perspektive sie in Hinsicht auf Veränderungen übernehmen beeinflusst, was für sie wie sichtbar wird. Die Veränderungsorientierung softwaretechnischer Forschung erfordert daher ein reflektierteres Vorgehen seitens der Forscher.

Mit dem Konzept der Aktionsforschung, als der reflektierend beeinflussenden Teilnahme, kann eine mehr aktive Rolle bei der empirischen Forschung methodisch gefasst und wissenschaftlich begründet werden (Stringer 1999).

3.4 Schritte zu einer kooperativen Methodenentwicklung

Die Diskussion der existierenden Ansätze zu empirischer Forschung hat gezeigt, dass keiner der existierenden Ansätze unverändert angewendet werden kann, wenn es darum geht, Softwareentwicklung in Richtung auf Anwendungsorientierung hin zu verbessern bzw. die Methoden der anwendungsorientierten Softwareentwicklung praxistauglich zu machen. Wir haben deshalb begonnen, einen eigenen Ansatz zu entwickeln, der methodische Momente aller drei Richtungen kombiniert. Im folgenden Abschnitt berichte ich von einem Kooperationsprojekt, in dem wir erste Erfahrungen mit diesem Ansatz gesammelt haben. Hier wird unser Vorgehen stichpunktartig beschrieben.

Veränderungsvorschläge sollten auf ein gründliches Verständnis der Arbeitspraxis, die verbessert werden soll, aufbauen. Wir wenden Methoden der qualitativen Sozialforschung, insbesondere Interviews, teilnehmende Beobachtung, Interaktionsanalyse und Dokumentanalyse an, um die Arbeitspraxis der Softwareentwickler, mit denen wir zusammenarbeiten, zu verstehen. Wichtig ist es gerade in Arbeitszusammenhängen, die Beobachtungen und deren Auswertung mit den Beobachteten rückzukoppeln. Sie erhalten dadurch die Möglichkeit, die Darstellung ihrer Praxis in ihrem Arbeitsumfeld zu beeinflussen. Darüber hinaus können wir Missverständnisse bereinigen und eine gemeinsame Grundlage für das weitere Vorgehen schaffen. In Dittrich/Rönkkö 1999 und Rönkkö 2002 haben wir erste Erfahrungen damit zusammengefasst.

Im Hinblick auf Veränderungsvorschläge verfolgen wir ein kooperatives Vorgehen ähnlich dem Ansatz zur ‚reflektiven Softwareentwicklung'. Allerdings spielen wir eine mehr aktive Rolle, d.h. wir bringen unser Wissen zu Methoden der anwendungsorientierten Softwareentwicklung ein. In Workshops und Projekttreffen diskutieren wir mit dem Projektteam, was die Projektteilnehmer

als problematisch und verbesserungswürdig auffassen. Wir entwickeln konkrete Maßnahmen, die dann im Projekt umgesetzt und von den Forschern begleitet werden.

Dieses zyklische Vorgehen ermöglicht es, methodische Innovationen und Veränderungen in realen Softwareprojekten auszuprobieren und auszuwerten, ohne den Projekterfolg zu gefährden. Anders als die Ansätze des SPI haben wir jedoch nicht das Ziel, die lokale Praxis einer unabhängigen Norm anzupassen. Sondern wir verfolgen ein offenes Vorgehen, in dem wir selbst anhand dessen, was sich als anwendbar erweißt und wie es an die konkrete Projektsituation angepasst werden muss, mehr über anwendungsorientierte Softwareentwicklung in der Praxis lernen.

4 Softwaregestaltung, die Veränderung unterstützt

Mobile Telekommunikation ist eine Technologie unter ständiger Entwicklung. Der Wettbewerb zwischen verschiedenen Anbietern trägt zusätzlich zum Veränderungsdruck bei. Es gibt von daher wenig Standardsoftware, die die Geschäftsprozesse unterstützen. Flexible und leicht anpassbare Software ist von daher ein strategischer Geschäftsvorteil und ermöglicht es den Anwendern, ihre Arbeitspraxis der Logik der Aufgaben anzupassen.

Das Programm, dessen Entwicklung wir begleitet haben, verwaltet Verträge und berechnet auf diesen Verträgen beruhende Zahlungen, die von Ereignissen angestoßen werden.[2] Mit der bisherigen Software konnten nur Verträge verwaltet und Zahlungen berechnet werden, die von einer bestimmten Sorte Ereignissen angestoßen werden. Verschiedenen Vertragstypen und die Priorisierung zwischen Zahlungen entsprechend verschiedener Verträge, die von demselben Ereignis getriggert wurden, waren im Quellcode implementiert. Verträge und Zahlungen, die nicht in dieses Schema passten, mussten von Hand verwaltet und berechnet werden. Das führte dazu, dass wünschenswerte Änderungen der Geschäftspraxis für die Anwender zusätzliche und lästige Arbeit darstellten oder überhaupt nicht möglich waren.

Neben der Universität und dem Telekommunikationsanbieter ist ein kleines Softwarehaus an dem Projekt beteiligt, dass ein flexibles Datenbanksystem entwickelt hatte und mit Hilfe der Projektergebnisse zur Produktreife weiterentwickeln will.[3] Flexible Software ist an sich schon ein Schritt in Richtung Anwendungsorientierung. Allerdings beziehen sich die wissenschaftlichen Beiträge meist auf General-Purpose-Software, d.h. auf Software, die nicht für einen spezifischen Anwendungskontext gestaltet wurde. Inwieweit lassen sich diese Ergebnisse auf Spezialanwendungen übertragen? Und wie verändert sich das Verhältnis zwischen Entwicklung, Benutzung und Wartung? Das waren die Forschungsfragen, mit denen wir in das Projekt gegangen sind.

4.1 Teilnehmende Beobachtung

Ein Doktorand hat regelmäßig an Projektsitzungen und Arbeitstreffen des Projektes teilgenommen und diese auch aufgezeichnet. Während der Vorstudie konnte die Entwicklungspraxis so im Grossen und Ganzen nachvollzogen werden. Zusätzlich haben wir mit Hilfe von Interviews

[2] Um die Geschäftsinteressen unseres Kooperationspartners zu schützen, können wir keine näheren Angaben zum Charakter der Verträge und Ereignisse machen.

[3] Das Forschungsprojekt wird von der schwedischen 'Knowledge foundation' unterstützt. Olle Lindeberg schreibt im Rahmen des Projektes seine Dissertation. In diesem Artikel wird das Projekt nur zur Illustration des methodischen Ansatzes verwendet. Ausführlichere Darstellungen sind in (Dittrich et al. 2001) und (Dittrich/Lindeberg 2002) zu finden. Für weitere Information zum Datenbanksystem siehe (Diestelkamp/Lundberg 1999).

sowie einer Analyse des Quellcodes und der Entwicklungsdokumente die Entstehung und Veränderung des existierenden Programms nachvollzogen (Dittrich/Lindeberg 2001).

Nach der Vorstudie hat sich der Schwerpunkt der Arbeit von Workshops zu verteilter Programmierarbeit verschoben. Um auch weiterhin einen guten Einblick in die Projektpraxis zu erhalten, hat einer der Forscher eine kleinere Programmieraufgabe übernommen. Zusätzlich haben wir - wo es sinnvoll erschien - Interviews durchgeführt, um die Sichtweise von Schlüsselpersonen zu einem bestimmten Zeitpunkt einzufangen.

Nach Abschluss der Entwicklungsarbeiten haben wir in einem Workshop zusammen mit den Projektmitgliedern das Projekt rekapituliert und diskutiert. Die Workshops und Treffen, in denen wir Methoden und Implementationstechniken vorgestellt haben, haben wir ebenfalls aufgezeichnet.

Die Auswertung der Aufzeichnungen wurde angelehnt an (Lamnek 1995) durchgeführt. Zum Teil haben wir sie benutzt, um die Workshops zu dokumentieren und die Ergebnisse nachzuvollziehen.

4.2 Wie kann Software flexibel gestaltet werden?

Von Anfang an wurden in Projekttreffen und Workshops verschiedene Implementationsformen anpassbarer Software diskutiert. Schon die Existenz des flexiblen Datenbanksystems und die Erwägung, es eventuell einzusetzen, hat das Projekt beeinflusst. Wie man jedoch systematisch die stabilen Teile der Software und die Teile, die flexible gestaltet werden sollen, unterscheidet, und wie eine passende Software aussehen könnte, war für alle Beteiligten Neuland.

Im Rahmen einer kleinen, speziellen Untersuchung haben wir uns ein Bild über die konkreten Anforderungen verschafft. In Workshops haben wir unter Verwendung von ‚mock-ups' (Papierrepräsentation der Benutzungsschnittstelle) zusammen mit den Anwendern die Gestaltung einer Benutzungsschnittstelle diskutiert, die es den Anwendern selbst ermöglicht, die Software für neue Vertragstypen anzupassen.

In Workshops mit den Entwicklern haben wir verschiedene Möglichkeiten der Metamodellierung und der flexiblen Implementation vorgestellt und diskutiert. Prototypen, die die Machbarkeit und den Aufwand für eine weitgehend flexible Implementation mit Hilfe des Datenbanksystems demonstrierten, wurden zum Teil von den beiden industriellen Partnern des Projektes realisiert, zum Teil mit Hilfe von Studentenprojekten.

Workshops und Prototyp-Implementation haben es den Projektmitgliedern ermöglicht, verschiedene Realisierungsmöglichkeiten abzuschätzen und sich für ein bestimmtes Design zu entscheiden. Wir haben zusammen mit den Projektmitgliedern verschiedene Vorgehensweisen und Gestaltungsmöglichkeiten ausprobiert. Durch unsere Beteiligung ist das Projektteam reflektierter vorgegangen. Wir haben Gestaltungsmöglichkeiten und methodische Elemente ausprobieren können.

4.3 Flexibility Light

Das Design, für das sich im Projekt letztendlich entschieden wurde, implementiert nur einen Teil der möglichen Flexibilität und benutzt das flexible Datenbanksystem nicht. Ausschlaggebend dafür war, dass eine vollständig flexible Lösung mit generischem Interface nicht benutzerfreundlich genug gewesen wäre, und eine solche Implementation die Wartbarkeit des Programms verschlechtert hätte. Darüber hinaus erlaubt die in diesem Unternehmen praktizierte enge

Zusammenarbeit zwischen IT-Abteilung und Anwenderabteilung in gewissem Rahmen die kurzfristige Implementation von Änderungswünschen. Die Softwarearchitektur ist so gestaltet, dass voraussehbare Änderungen, die über die Tailoringmöglichkeiten hinausgehen, mit geringem Aufwand implementiert werden können.[4]

Die implementiert ‚Flexibility light'- Lösung orientiert sich insofern sowohl an dem Benutzungskontext als auch am spezifischen Entwicklungskontext und den Möglichkeiten der Zusammenarbeit zwischen Entwicklern und Benutzern.

5 Ergebnisse und Ausblick

Anpassbarkeit an sich verändernde Arbeitspraktiken ist eine der Grundforderungen für anwendungsorientierte Gestaltung die sich in der CSCW-Forschung herauskristallisiert haben (Henderson/Kyng 1991, Trigg/Bødker 1994) Die Erfahrungen mit dem beschriebenen Projekt haben gezeigt, dass die Implementation flexibler Lösungen in der industrieller Praxis möglich und sinnvoll ist. Darüber hinaus hat das Projekt deutlich gemacht, dass eine konkrete Implementation neben dem Anwendungskontext auch den Entwicklungskontext und die Form der Zusammenarbeit zwischen Entwicklern und Anwendern berücksichtigen sollte. Die Erfahrungen, die wir mit der Gestaltung flexibler Systeme gesammelt haben, haben wir in einem methodologischen Toolkit zusammengefasst. Wir haben vor, dieses in einem weiteren Projekt zu evaluieren und zu verbessern.

Neben den praktischen positiven Ergebnissen konnten wir in diesem Projekt unseren methodologischen Ansatz zu empirischer Forschung für die anwendungsorientierte Softwareentwicklung nutzbringend einsetzen und Erfahrungen mit Interventionen in Softwareentwicklungsprojekten sammeln. Sowohl die Projektmitglieder als auch die Verantwortlichen in der IT-Abteilung haben das Projekt als positiv erlebt. Die IT-Abteilung des Telekommunikationsanbieters entwickelt Software, wie sich zu unserer großen Überraschung herausgestellt hat, in enger Kooperation mit den künftigen Anwendern, ohne sich explizit auf anwendungsorientierte Softwareentwicklung zu beziehen. Hier sind interessante Möglichkeiten zu einer weiteren Zusammenarbeit gegeben.

Dank

meinen Kollegen Olle Lindeberg und Kari Rönkkö und allen anderen, die mit ihren Diskussionen und mit ihrer Forschung zum Inhalt dieses Artikels beigetragen haben.

Literatur

Aaen, I.; Arendt, J.; Mathiassen, L.; Ngwenyama, O. (2001): A Conceptual MAP of Software Process Improvement. In: Scandinavian Journal of Information Systems, Vol.13, Special Issue on Trends in the Research on Software Process Improvement in Scandinavia: pp. 123-146

Basili, V. (1996): The role of experimentation in software engineering: past, current, and future. Proceedings of the ICSE '96, New York: ACM Press, pp. 442-449

Basili, V.; Greeen, S. (1994): Software Process Evolution at the SEL. IEEE Software, Jg. 11 (July) pp. 58-66

[4] Zu einem detaillierteren Vergleich der Prototypen und des implementierten Designs siehe (Lindeberg/Diestelkamp 2001).

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

Bødker, S. (1999): Computer Applications as mediators of design and use - a developmental perspective. DAIMI PB-542, Computer Science Department, Århus University.

Button, G.; Sharrock, W. (1994): Occasioned practice in the work of software engineers. In: Jirotka, M.; Goguen, J.: Requirements Engineering. Social and Technical Issues. London: Academic Press.

Button, G.; Sharrock, W. (1996): Project Work: The Organization of Collaborative Design and Development. In: Software Engineering Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design. Vol. 5, No.4, pp. 368-386

Button, G.; Sharrock, W. (1998): The Organizational Accountability of Technological Work. Social Studies of Science 28/1: pp. 73-102

Bürkle, U.; Gryczan, G.; Züllighoven, H. (1995): Object-Oriented System Development in a Banking Project: Methodology, Experiences, and Conclusions. Human Computer Interaction 10 (2&3), pp. 293-336

Christensen, M.; Crabtree, A.; Damm, C. H.; Hansen, K. M.; Lehrman Madsen, O.; Marqvardsen, P.; Morgensen, P.; Sandvad, E. ; Sloth, L. ; Thomsen, M. (1998): The MAD Experience: Multiperspective Application Development in evolutionary prototyping, Proceeding of the ECOOP 1998, pp. 1-16.

Diestelkamp, W.; Lundberg, L. (1999): Promis, a Generic Product Information Database System. In: R. Y. Lee, (Ed.): Proceedings of the ISCA14th International Conference, Cancun Mexiko, April 7-9, 1999.

Dittrich, Y. (2002): Doing empirical research in Software Engineering: Finding a path between understanding, intervention and method development. In: Dittrich, Y.; Floyd, C.; Klischewski, R.: Social Thinking - Software Practice. Cambridge: The MIT Press (to be published)

Dittrich, Y.; Lindeberg, O. (2001): Can Software Development be too Use Oriented? Going Native as an issue in Participatory Design. In: Bjornestad, S.; Moe, R. E.; Mørch, A. I.; Opdahl, A. L. (eds.): Proceedings of the 24th Information Systems Research Seminar in Scandinavia, Ulvik in Hardanger, Norway, 11th-14th August 2001. University of Bergen, Department of Information Science.

Dittrich, Y.; Lindeberg, O.; Ludvigsson, I.; Lundberg, L.; Wessman, B.; Diestelkamp, W.; Tillman, M. (2001): Design for Change. Research Report 2001. Blekinge Institute of Technology, Sweden, ISSN 1103-1581

Dittrich, Y.; Lindeberg, O. (2002): Designing for changing work and business practices. (in Vorbereitung)

Dittrich, Y., Rönkkö, K. (1999): Talking Design. Proceeding of the IRIS '99, Department of Computer Science, Aalborg University , pp. 267-280.

Ehn, P. (1988): Work oriented design of computer artefacts. Almqvist &Wiksell International, Stockholm (Sweden).

Floyd, C.; Reisin, F.-M.; Schmidt, G. (1989): STEPS to Software Development with Users. In: Ghezzi, G.; McDermid, J. A. (eds.): Proceedings of the ESEC '89, Berlin.

Floyd, C.; Mehl, W.-M.; Reisin, F.-M.; Schmidt, G.; Wolf, G. (1989): Out of Scandinavia: Alternative Software Design and Development in Scandinavia. In: Journal for Human-Computer-Interaction 4, pp. 253-380.

Floyd, C., H. Zuellighoven (1997) Softwaretechnik, In: Rechenberger, P.; Pomberger, G.: Informatik-Handbuch. München, Wien: Hanser, S. 641-667.

Grinter, R. E. (1996): Supporting Articulation Work using Software Configuration Management Systems. Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design. Vol. 5, No.4, pp. 447-465

Grinter, R. E. (1998): Recomposition: Putting It All Back Together Again. Proceedings of the Computer Supported Cooperative Work CSCW, (Seattle, November 1998), ACM Press, pp. 393-402.

Henderson, A.; Kyng, M. (1991): There is no place like Home: Continuing Design in Use. In: Greenbaum, J.; Kyng, M.: Design at Work, Lawrence Erlbaum Associates, pp. 219-240

Herbsleb, J. D.; Mockus, A.; Finholt, T. A.; Grinter, R. E. (2000): Distance, dependencies, and delay in global collaboration. Proceeding on the ACM 2000 Conference on Computer supported cooperative work, pp. 319-328

Iversen, J. K.; Nielsen, P. A.; Norbjerg, J. (1998): Problem Diagnosis Software Process Improvement. In: Larsen, T. J.; Levine, L.; DeGross, J. I. (eds.): Information systems: Current Issues and Future Changes IFIP. Laxemburg, Austria, pp. 111-130.

Karasti, H. (2000): Co-constructing shared understanding of work practices for system design: The interplay of views. Occational paters from the Work Practice Laboratory, Vol.1, No. 1 University of Karlskrona/Ronneby, Department of human Work Science, Ronneby, Sweden.

Kensing, F. (2000): Participatory Design in a Commercial Context: A Conceptual Framework. In: Proceedings of the PDC 2000. New York: ACM Press, pp. 116-126.

Krabbel, A.; Wetzel, I. (1998): The Customization Process for Organizational Package Information Systems: A Challenge for Participatory Design. In: Henderson Chatfield, R.; Kuhn, S.; Muller, M. (eds.): PDC'98 Proceedings of the Participatory Design Conference, Seattle, Washington, USA, 12-14 November 1998, pp. 45-54

Lamnek, S. (1995): Qualitative Sozialforschung. 2 Bd. 3. korr. Auflage, Weinheim 1995.

Lindeberg, O.; Diestelkamp, W. (2001): How Much Adaptability do You need? Evaluating Meta-modeling Techniques for Adaptable Special Purpose Systems. Proceedings of the Fifth IASTED International Conference on Software Engineering and Applications, SEA 2001.

Mathiassen, L. (1998): Reflective Systems Development. Scandinavian Journal of Information Systems 10 (1&2), pp. 67-118

Mathiassen, L. (1997): Reflective Systems Development. Dr. Tech. Thesis, Aalborg University. http://www.cs.auc.dk/~larsm

Nielsen, P.A.; Nørbjerg, J. (2001): Assessing Software Processes: Low Maturity of Sensible Practice. Scandinavian Journal of Inforamtion Systems, Vol.13, Special Issue on Trends in the Research on Software Process Improvement in Scandinavia, pp. 51-68

Nosek, J.; Palvia, P. (1990): Software Maintenance management: changes in the last decade. Journal of Software Maintenance: Research and Practice, 2, pp. 157-174

Paulk, M.C.; Curtis, B.; Chrissis, M. B.; Weber, C. V. (1993): Capability Maturity Model, Version 1.1. Communications of the ACM, Vol. 35, No. 4., pp. 18-27

Potts, C.; Catledge, L. (1996): Collaborative Conceptual Design: A Large Software Project Case Study. Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design. Vol. 5, No.4, pp. 415-445

Robertson, T. (1996): Embodied Action in Time and Place: The Cooperative Design of a Multimedia, Educational Computer Game. Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design. Vol. 5, No.4, pp. 341-367

Rönkkö, K. (2002): '- Yes, what does that mean?' Understanding distributed requirements handling. In: Dittrich, Y.; Floyd, C.; Klischewski, R.: Social Thinking - Software Practice. Cambridge: The MIT Press (to be published)

Scandinavian Journal of Information Systems (1998), Vol. 10 (1&2)

Schmidt, K.; Sharrock, W. (ed.) (1996): Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design. Vol. 5, No.4.

Schön, D. A. (1983): The Reflective Practitioner. How Professionals Think in Action. New York: Basic Books.

Stringer, E.T. (1999): Action Research. 2nd Edition. Thousand Oaks, Califormia: Sage publication.

Suchman, L.; Trigg, R. (1993): Artificial Intelligence as craftwork. In: Chaiklin, S.; Lave, J.: Understanding Practices - Perspectives on Activity and Context. New York: Cambridge University Pres, pp. 144-178

Trigg, R.; Bødker, S. (1994): From Implementation to Design: Tailoring and the Emergence of Systematization in CSCW. Proceedings of the CSCW '94, ACM-Press, New York: pp. 45-55

Tellioglu, H.; Wagner, I. (1997): Negotiating Boundaries. Configuration Management in Software Development Teams. Computer Supported Cooperative Work, Vol. 6, pp. 251-274

Methoden im Arbeitsalltag einer Entwicklungsfirma

Ein Kommentar zum Beitrag von Yvonne Dittrich[1]

Friedrich Strauss
software design & management AG
Thomas-Dehler-Str. 27
81737 München
strauss@sdm.de

Bevor ich den Beitrag von Frau Dittrich kommentiere, möchte ich meinen Arbeitskontext und Background kurz erläutern: Ich bin Diplom-Informatiker, habe im Bereich Software-Ergonomie promoviert und bin daher vorbelastet, wenn es um das Thema Anwendungsorientierung und Nutzbarkeit geht - übrigens eines der spannendsten Themen in diesem Bereich. Seit 1995 bin ich bei sd&m, als Senior-Berater sind meine Schwerpunkte die Beratung zu grafischen Oberflächen (Ergonomie und Programmierung), derzeit bin ich technischer Chefdesigner und Projektleiter in einem GUI-lastigen Java Client-Server Projekt. Prinzipiell bevorzuge ich alle anfallenden Aufgaben, deren Managementanteil nicht zu groß ist. Was entwickeln wir bei sd&m? Wir erstellen typische Informationssysteme für große Kunden wie Fahrzeughersteller, Banken, Kommunikationsunternehmen. In der Regel ist das eine Software zur Verwaltung von Aufträgen, Verträgen, Buchungen u.ä., die große komplexe Datenbanken und meist nur einfache (lineare) Algorithmen enthalten. sd&m konzentriert sich darauf, individuelle Informationssysteme zu erstellen. Dort liegen unsere Kernkompetenzen, nicht jedoch in der Erstellung oder Einführung von Produkt-Software, oder der Betrieb einer Anwendung inkl. ihrer Infrastruktur. Unser Leitspruch lautet: „Über jedem Standard“ Das, was wir machen, machen wir möglichst gut. Zumindest bestätigen uns führende Beratungsunternehmen, dass wir individuelle Anwendungsentwicklung über dem branchenüblichen Standard betreiben.[2]

Frau Dittrich vertritt in ihrem Papier die These, dass in der traditionellen Softwareentwicklung Anwenderbeteiligung keine oder nur eine marginale Rolle spielt. Meiner Meinung nach werden Anwender nach Möglichkeit schon bewusst einbezogen, aber die dabei verwendeten Methoden sind stark an den kommerziellen Kontext adaptiert. Das könnte eine Ursache dafür sein, dass die Anwenderbeteiligung in der akademischen Sichtweise als unvollständig und suboptimal wahrgenommen wird. Manchmal ist sie auch nicht mehr wiedererkennbar, denn wir müssen in einem Projekt alle - auch in sich widersprüchliche - Anforderungen berücksichtigen. Um meinen Begriff der sich widersprechenden Anforderungen verständlich zu machen, möchte ich darauf etwas ausführlicher eingehen: widersprüchliche Anforderungen finden sich z.B. bei Prototyping als

[1] Dieser Beitrag basiert auf den Tonbandaufzeichnungen während des Workshops.

[2] sd&m wurde in die McKinsey-Studie „Secrets of Software Success” als eines von 100 Unternehmen weltweit aufgenommen. Siehe auch Brössler, Peter; Siedersleben, Johannes (Hrsg) (2000): Softwaretechnik: Praxiswissen für Software-Ingenieure. München, Wien : Hanser.

wichtige Technik zur Anforderungsermittlung im Gegensatz zu Zwang zum Festpreis-Angebot, enge Zeitvorgaben, und effiziente Spezifikationen für ein großes Realisierungsteam. Wenn wir konkret Prototyping vs. Spezifikation betrachten, dann gibt es folgendes Problem: In unserem Kontext, arbeiten häufig 7-10 (oder auch bis zu 60) Mitarbeiter zur Realisierungszeit in einem Projekt. Je nach Vorgehen beim Prototyping haben sich 3-4 (oder bei Großprojekten vielleicht auch 6-8) Mitarbeiter mit dem Prototypen beschäftigt und kennen die damit assoziierte Funktionalität - wie gebe ich nun die gewonnen Informationen an die anderen Mitarbeiter weiter? Sicher nicht ausschließlich über den Prototypen, man benötigt trotzdem eine detaillierte Spezifikation. Überspitzt formuliert: Ich kann nicht durch Ausprobieren des Prototypen herausfinden, was ich programmieren muss bzw. worauf ich mich an meinen Schnittstellen verlassen kann. Mein Eindruck ist, dass bei Software Engineering Methoden wie iteratives Vorgehen, Prototyping, evolutionären Softwareentwicklungsmethoden der für größere langdauernde Projekte in allen Phasen wichtige Dokumentationsaspekt unterbetont ist.

Nimmt man als Basis seines Vorgehens nun kein Prototyping, keinen iterativen oder evolutionären Ansatz, sondern eine Spezifikationstechnik, gibt es entsprechende Probleme. Spezifikationen dienen ja auch der „partizipativen", anwenderorientierten Entwicklung: sie sollen die Kommunikation und Abstimmung mit dem Fachbereich (Auftraggeber, Anwender) unterstützten. Es gibt eine große Bandbreite von vorgeschlagenen Spezifikationsnotationen, als Beispiel betrachten wir die verschiedenen UML-Diagrammarten sowie die Anwendungsfälle. Beide Ansätze sind für unseren Alltag bei sd&m nicht direkt brauchbar: Mit der UML werden detaillierte Beschreibungen propagiert - mit ausufernd vielen Diagrammarten und Darstellungselementen. Diese vielfältigen Diagramme sind für den Anwender häufig nicht in dem Sinne zu verstehen, dass er tatsächlich prüfen kann, ob das, was wir beschreiben, auch das widerspiegelt, was er in seinem Arbeitsalltag braucht. Die Diagramme sind eher zur IT-internen Kommunikation geeignet, erzeugen aber noch das zusätzliche Problem der Überspezifikation: Sequenz-Diagramme zum Beispiel werden leider zu häufig missverstanden und zur visuellen Programmierung in der Spezifikationsphase genutzt. Für die frühen Phasen dagegen werden Anwendungsfälle hervorgehoben (oder „essentielle Anwendungsfälle", Szenarios, und weitere ähnliche Ansätze), wo mit freiem Text genau eine Anwendungssituation beschrieben wird. Würden wir Anwendungsfälle aber wörtlich nehmen, dann hätten wir Spezifikationen mit hunderten von Anwendungsfällen, für die es dann auch noch Sonderfälle und Nebenbedingungen gäbe. Wie man nun UML und Anwendungsfälle nutzt, nun eine gute leserliche und trotzdem geeignet detaillierte Spezifikation zu erhalten, haben wir uns bei sd&m selbst mühsam aneignen müssen. Das Ergebnis ist eine dokumentenorientierte Spezifikationstechnik, in der auch UML-artige Diagramme auftauchen (vor allem Klassendiagramme, ggf. auch Zustands- und Interaktionsdiagramme, selten Sequenz-Diagramme)[3] sowie Anwendungsfälle die zu sehr strukturierten Ablaufbeschreibungen mutiert sind (mit Vorbedingungen, Nachbedingungen, Sonderfälle, Zusammenfassung usw.).

Ich hoffe, dass ich mit den Ausführungen zum Prototyping vs. Spezifikation und zu Spezifikationsnotationen meine Aussage ausreichend skizziert habe, „dass man alle Anforderungen berücksichtigen muss, wenn man eine neue Methode entwickelt und sagt, dass sie dieses oder jenes besser macht".

Die zweite These von Frau Dittrich besagt, kooperative Methodenentwicklung ist nötig, weil die Methodenverbesserungen anhand statistischer Untersuchungen nicht sehr weit tragen. Sie unter-

[3] Besonders die Spezifikation der Benutzerschnittstelle scheint in vielen (frühen) UML-Büchern keine Rolle zu spielen: Entweder werden dort immer Batch-Systeme betrachtet oder die Benutzerschnittstelle wird als trivial angesehen.

suchen Ursache-Wirkungs-Zusammenhänge nach dem naturwissenschaftlicher Ansatz. Dieser Ansatz bringt in bestimmten überschaubaren Bereichen interessante Erkenntnisse zutage. Ich stimme Frau Dittrich aber zu, dass der Ansatz mir jedoch nicht weiter hilft, wenn ich Methoden wirklich in die Praxis umsetzen will. Es ist schnell nachvollziehbar, dass Inspektionen besser sind, als nur zu Testen. Hier lautet die spannende Frage, wie ich mein Team dazu bringe, Inspektionen zu machen und sie richtig zu machen; womit wir dann bei Frau Dittrichs Thema „Kooperative Methodenentwicklung" angekommen sind. Es ist aufschlussreich wissenschaftlich fundiert zu sehen, ob eine bestimmte wissenschaftlich als besser erachtete Technik in der Praxis auch wirklich besser ist. Uns helfen neue Methoden aber nur, wenn wir sie in unserem praktischen Kontext einsetzen können und sie auch zur Verbesserung des Entwicklungsprozesses beitragen. Wenn sich eine wissenschaftlich als schlechter erachtete Methode besser in der Praxis bewährt als eine wissenschaftlich als gut erachtete Methode, dann nehmen wir die wissenschaftlich schlechtere Methode. Wir haben im Bereich der Aufwandsschätzungen - das ist ein wichtiges Problemfeld der Praxis - auch probiert, mit Metriken zu arbeiten. Wir haben dabei mit dem Fraunhofer Institut in Kaiserslautern kooperiert. Für unseren Anwendungskontext haben wir festgestellt, dass wir mit den gemeinsam ausprobierten naturwissenschaftlichen Methoden nicht wirklich besser werden. Die Bemühungen in diese Richtung haben wir daher wieder eingestellt.

Eine weitere These von Frau Dittrich, die ich hier herausgreifen möchte, besagt, dass wir es bei Softwareentwicklung mit kooperativer Arbeit zu tun haben. Das kann ich nur unterstützen. Auch die Ausführungen zur reflexiven Methodenentwicklung finde ich gut. Ich teile ihre Auffassung, dass Bottom up alleine nicht unbedingt Verbesserungen erbringt. Es ist praktikabler, externe Ideengeber zu haben. Jemand, der in neuen Methoden ausgebildet wurde, sie kennt und sagt: „Ich habe hier einen Vorschlag. Lasst uns doch versuchen, diesen in für den Arbeitsalltag verträglichen Schritten nutzbar zu machen." bringt für uns mehr Gewinn, als wenn wir alles allein selbst erfinden müssen. Es ist in unserem Kontext zudem nicht praktikabel, radikale Umstellungen vorzunehmen in der Art: Zur Produktivitätssteigerung führen wir mit dem nächsten Projekt objektorientierte Programmierung oder Extreme Programming oder Iterative Entwicklung mit wöchentlichen Zyklen ein. Wir benötigen überschaubare Schritte, was vor allem bei Paradigmen-Wechseln und bei Änderungen des Vorgehens schwierig ist. Was ist in solchen Fällen ein überschaubarer Zwischenschritt? An dieser Stelle ist eine Unterstützung in der Form sinnvoll, dass Wissenschaftler mit dem Team im Projekt zusammenzuarbeiten.

Ich denke, diese von Frau Dittrich propagierte Form der Unterstützung hat wirklich eine Chance, denn dann werden auch den Wissenschaftlern die Anforderungen, die ich eingangs genannt habe, deutlich.

Anwenderorientierte Softwareentwicklung ist in unserer Firma schon sehr lange - zum Teil unbewusst, zum Teil aus anderen Antrieben - wichtig. Einer unserer wichtigsten Aspekte ist es zu sagen: Wenn wir eine Spezifikation machen, dann wollen wir sie am liebsten in einem gemischten Team aus Kunde und sd&m-Kollegen erstellen. Dieses Prinzip stammt vermutlich nicht aus dem Wissenschaftsbereich, der partizipative, evolutionäre und anwenderorientierte Methoden propagiert, sondern es entstand aus der Erfahrung in der Praxis. Wir müssen eng mit unseren Kunden zusammenarbeiten, wir brauchen den Austausch zwischen ihrem und unserem Wissen. Aber die Ziele, die hinter diesem Prinzip stehen, sind ähnlich. Lediglich die Herkunft ist verschieden und wir haben keinen derartigen Fachbegriff dafür und vermutlich auch einen weniger strukturierten bzw. fundierten Zugang.

Wichtig ist auch weiterhin, dass ein Informatiker sein Handwerkszeug beherrscht. Er muss eine gute Spezifikationen aufschreiben können, die einerseits vom Anwender verstanden wird, die andererseits aber auch auf Spezialfälle eingeht und widerspruchsarm ist, so dass sie eine aussage-

kräftige Basis für die Programmierung darstellt. Ich muss als Informatiker drauf achten, wo Lücken sind, wo Informationen fehlen, worüber noch diskutiert werden muss. Das hat sehr wenig mit der Frage zu tun, ob die Spezifikation formal oder weniger formal im mathematischen Sinne ist. Auf das ebenfalls wichtige Thema der Softwarearchitektur (das schon mit einem guten und an den wichtigen Stellen flexiblem Datenmodell anfängt) will ich hier nicht eingehen.

Es gibt noch eine Reihe weiterer Aspekte, wie beispielsweise die Menschen, Auftragsplanung, Kosten, Zeit, Ressourcen. Bei einem Softwarehaus, das als Auftragnehmer arbeitet, sind diese Aspekte immer wieder sehr wichtig. Sie reiben sich natürlich manchmal mit der Anwenderorientierung, mit den Anforderungen der Anwender auf Änderungen. Es gibt immer wieder Situationen, in denen man diese abwehrt, weil erst einmal ein Abschnitt des Projektes erfolgreich (!) abgeschlossen werden soll. Das bedeutet dann konkret, das Anwender ein halbes Jahr mit der Software leben müssen wie sie ist, bevor wir wieder Änderungen vornehmen. Das ist ein Konflikt, mit dem man leben muss.

Frau Dittrich, ihre Projektidee hört sich gut an. Die Idee mit der flexiblen Datenbank erscheint mir hilfreich. Allerdings hätte ich gern noch mehr konkrete Dinge dazu erfahren. Ich warte also mit Spannung auf weitere Veröffentlichungen.

Software Engineering und Arbeitsalltag in der Softwareentwicklung

Andrea Sieber, Annette Henninger
TU Chemnitz
Fakultät für Informatik
Professur Künstliche Intelligenz
09107 Chemnitz
[Andrea.Sieber|Annette Henninger]@informatik.tu-chemnitz.de

Die folgenden Ausführungen stellen eine Zusammenfassung des abschließenden Gesprächs mit auf dem Workshop „Software Engineering meets Practice“ anwesenden Praktikern dar. Sie dienen dazu aufzuzeigen, was Praxisvertreter aus verschiedenen Unternehmen am Software Engineering bzw. an der Informatik-Ausbildung bemängeln. Wir ergänzen sie zudem mit den Ergebnissen unserer empirischen Untersuchungen im Rahmen des DFG-Projekts „Softwareentwicklung in der Praxis im Kulturvergleich“ (Sieber/Henninger 2001, Henninger/Sieber 2001).

1 Software Engineering und Arbeitsalltag

Die Softwareentwickler und Geschäftsführer der kleinen Unternehmen, die im Rahmen unserer empirischen Untersuchungen im Projekt „Software entwickeln in der Praxis im Kulturvergleich“[1] mit uns sprachen, äußerten sich überwiegend kritisch zu den Methoden des Software Engineering, die ihnen in ihrer Ausbildung vermittelt wurden. Einige berichteten auch, dass Software Engineering überhaupt nicht Teil ihrer Informatik-Ausbildung gewesen sei. Unsere Interviewpartner bezogen sich vor allem auf die frühen Prozessmodelle, insbesondere das Wasserfallmodell und das Spiralmodell, an die sie sich noch vage aus ihrer Ausbildung erinnerten. Im Gegensatz zur schematischen Darstellung in den Modellen empfanden sie ihren Arbeitsalltag als wesentlich komplexer.

In den von uns untersuchten Unternehmen konnten wir insbesondere ein zyklisches und ein lineares Vorgehen identifizieren (Henninger/Sieber 2001 und Sieber/Henninger 2001). Bei einem zyklischen Designprozess erfolgte die Koordination der Entwicklungsarbeit durch die Festlegung von Verantwortungsbereichen. Alle Entwickler arbeiteten jeweils zeitlich parallel an einem Projekt und waren dadurch auch über die Arbeit ihrer Kollegen informiert. Entwürfe wurden im Entwicklungsteams gemeinsam diskutiert und mehrfach überarbeitet, bis eine für alle befriedigende Lösung gefunden wurde. Dabei war es für die Entwickler wenig relevant, in welcher Arbeitsphase des Entwicklungsprozesses sie sich befanden. Ihr Entwicklungsprozess bestand primär aus zwei Phasen: Diskutieren und Umsetzen der Diskussionsergebnisse. Auch bei der eher linearen Vorgehensweise in einem Unternehmen verlief der Entwicklungsprozess in einem längeren Zeithorizont betrachtet zyklisch. Hier entstand die Zyklizität durch Folgeprojekte, d.h. Programmteile wurden in anderen Projekten wiederverwendet oder die bestehende Software

[1] Dieses Projekt wurde im Rahmen der Chemnitzer Forschergruppe „Neue Medien im Alltag: Von individueller Nutzung zu soziokulturellem Wandel” (Bo 929/13-1) gefördert.

wurde für neue Kunden weiterentwickelt. In diesen Firmen gab es eine hierarchische Trennung zwischen Planungsaufgaben, die von den Projektleitern oder den Geschäftsführern übernommen wurden, und ausführenden Entwicklungsaufgaben, die an einzelne Entwickler delegiert wurden. Eine unmittelbare Kooperation zwischen den Entwicklern erfolgte nur zur gemeinsamen Lösung schwieriger Probleme. Eine Firma kombinierte eine hierarchische Trennung von planenden und ausführenden Aufgaben mit der Kooperation in flexiblen Entwicklungsteams, die entsprechend der Anforderungen des Projekts zusammengesetzt wurden.

Im Gegensatz zu der im Wasserfallmodell geforderten formalen Spezifikation zu Beginn eines Entwicklungsprojektes waren die Entwickler in der Praxis häufig mit wenig präzisen Vorstellungen der Kunden über das zu erstellende Softwareprodukt konfrontiert. Lagen schriftliche Dokumente des Kunden vor, so handelte es sich häufig um Skizzen, Diagramme, Tabellen und Texte, die darzustellen versuchten, wie der Kunde arbeitete und welche Funktion die Software dabei übernehmen sollte. In den meisten Fällen stellte sich jedoch heraus, dass diese Darstellungen die Arbeitsprozesse beim Kunden nur sehr grob und verallgemeinernd darstellten. Der Entwicklungsprozess gestaltete sich daher in der Regel als gemeinsamer Lernprozess, in dem sowohl die Entwickler als auch die Mitarbeiter des Auftraggebers nach und nach zu immer mehr Klarheit über das zu erstellende Produkt gelangten.

Trotz dieser abweichenden Vorgehensweise in der Praxis und bestehender Kritik an den im Software Engineering entwickelten Prozessmodellen bezogen sich viele unserer Interviewpartner auf das Wasserfallmodell als Abbild des idealtypischen Verlaufs von Entwicklungsprojekten. Sie assoziierten damit eine Abfolge von Arbeitsschritten, die gleichzeitig Meilensteine für das Projekt darstellten und eine sichere Terminierung im zur Verfügung gestellten Budgetrahmen ermöglichten.

Auch Weltz und Ortmann (1992) stellten bei ihrer Untersuchung des Projektmanagements in 46 Softwareentwicklungsprojekten eine Diskrepanz fest zwischen dem Vorgehen in der Praxis und den normativen Konzepten, welche die Wissenschaft anbietet. Die Organisation des Entwicklungsprozesses in den von Weltz/Ortmann untersuchten Projekten orientierte sich überwiegend am Phasenschema des Wasserfallmodells, das allerdings in der Praxis häufig situativ an die jeweiligen Bedingungen angepasst wurde (Weltz/Ortmann 1992, 81f). Die meisten Projekte, so die Autoren, bewegen sich auf einer „Gratwanderung zwischen den sich widersprechenden Anforderungen einer flexiblen [...] Projektabwicklung und dem Versuch, das Projekt berechenbar zu halten, ohne ein klares Konzept." (ebd., 87). Weltz/Ortmann empfehlen zur Optimierung des Projektablaufs ein inkrementelles Vorgehen, das einen stetigen Kontakt zur (Anwendungs-) Realität beinhaltet und während des Projektverlaufs zunehmend genauere Voraussagen erlaubt (ebd., 144f).

Neben dem Wasserfallmodell gibt es eine Vielfalt weiterer Prozessmodelle. Einen Überblick gibt Bremer (1998). Neuere Entwicklungen sind in Elting/Huber (2001a,b) dargestellt. Dennoch beziehen sich Entwickler und Management überwiegend auf das erste Phasen- bzw. Wasserfallmodell. Eine Ursache für diese Diskrepanz zwischen Forschung und Praxis könnte ein time-lag zwischen Ausbildung und Praxis sein: die Studienzeit der von uns befragten Entwickler lag schon einige Jahre zurück, und auch die damals gelehrten Modelle entsprachen womöglich nicht dem aktuellen Stand der Forschung.

Doch wird aus unseren Interviews auch deutlich, dass die Begrenzung der Modelle auf die reine Softwareentwicklung ohne Berücksichtigung der Vielzahl anderer Faktoren, die ein Projekt und seinen Ablauf beeinflussen, sie für die Praxis untauglich machen. Dahinter vermuten wir einen prinzipiellen Unterschied, der in der Soziologie als das Struktur-Handlungs-Problem behandelt wird. Diese Sichtweise legt nahe, dass es immer zu Brüchen kommen wird zwischen den Regeln

und Normen für einen bestimmten Bereich des Alltags und der Art und Weise, wie konkrete Personen in konkreten Situation handeln. Regeln und Normen sind notwendigerweise abstrakt und blenden spezifische Lösungen aus, um für einen größeren Problembereich anwendbar zu sein. Das konkrete Handeln der Personen ergibt sich jedoch aus den spezifischen Anforderungen und ganz konkreten Problemlagen vor Ort. (vgl. Voß 1988) Aus diesen Gründen sind die Unternehmen in der Praxis zu einer erfahrungsgeleiteten Arbeitsweise übergegangen, wie wir sie weiter oben beschrieben haben.

Neben diesen Differenzen zwischen der Alltagspraxis in der Softwareentwicklung und den im Software Engineering entwickelten Vorstellungen über den Ablauf von Entwicklungsprojekten gibt es jedoch auch Methoden und Werkzeuge, die auf bestimmten Konzepten des Software Engineering beruhen, die durchaus Eingang in die Arbeitspraxis der Unternehmen gefunden haben. Hierzu zählen zum Beispiel die ganze Palette der Entwicklungswerkzeuge, Werkzeuge zur Verwaltung von Quelltext, zur Verständigung zwischen den Entwicklern oder zur Abrechnung von Projekten. Die meisten Entwickler und Geschäftsführer nutzten solche Werkzeuge und empfanden sie auch als hilfreich.

Ebenso verhielt es sich mit der in neueren Konzepten des Software Engineering geforderten Anwendungsorientierung (vgl. dazu Dittrich und Strauss in diesem Band). Bei Softwareprodukten, die aus Auftragsprojekten entstanden, spielte die Orientierung an den Bedürfnissen der Anwender von Anfang an eine große Rolle. Bei Produkten, die ohne Bezug zu einem konkreten Anwendungsbereich in Förderprojekten entwickelt wurden, war die Anpassung an den Nutzungskontext zunächst gering. Konnten die Unternehmen durch ihre Verkaufsversuche Kontakt zu konkreten Anwendern aufbauen, versuchten sie, ihre Produkte durch Umstrukturierungen und Ergänzungen besser an den Anwendungskontext anzupassen.

Das Erkennen der großen Bedeutung von Anwendungsorientierung bedeutete nicht, dass die Anwender tatsächlich am Entwicklungsprozess beteiligt waren. In der Regel hatten die von uns befragten Softwareentwickler lediglich mit dem Management des Auftraggebers oder mit technischen Experten zu tun und traten selten mit den realen Anwendern der Software in Kontakt. Einige Entwickler behalfen sich damit, die Software für fiktive Anwender zu konzipieren: Sie versuchten, sich in deren Lage zu versetzen, und richteten die Funktionalität und die Gestaltung der Oberfläche danach aus. Diejenigen, die tatsächlich mit einzelnen Anwendern Kontakt hatten, schnitten die Software auf die Bedürfnisse dieser realen Personen zu. Sie bezogen dann ihr Wissen aus konkreten Beobachtungen vor Ort. Trotzdem musste die Software dann mit jedem neuen Anwender nochmals geändert werden, wenn das Produkt an andere Kunden verkauft wurde. Oftmals unterschied sich die Priorität bei der Benutzung einer bestimmten Funktion, was eine veränderte Oberflächengestaltung oder eine Programmierung zusätzlicher Funktionen notwendig machte. Mit der Zeit und bei entsprechender konzeptioneller Struktur der Software hatte die Firma jedoch ausreichend Funktionsbausteine vorliegen, die sie jeweils neu zusammensetzen konnte. In diesen Fällen machte sich eine objektorientierte Arbeitsweise bezahlt. Die Anpassung an den Nutzungskontext wurde dadurch gewährleistet, dass die von uns untersuchten kleinen Firmen auf die Zufriedenheit ihrer Kunden angewiesen waren, um weitere Aufträge zu erhalten. Häufig wurden daher Änderungswünsche der Kunden an ersten Versionen der Software berücksichtigt, oder es wurden Folgeprojekte vereinbart, um eine erweiterte Funktionalität zu realisieren und die Software besser an den Nutzungskontext anzupassen.

2 Wünsche aus der Praxis an die Informatik-Ausbildung

Bei der Diskussion auf dem Workshop war die Diskrepanz zwischen den im Software Engineering bereitgestellten Modellen und Methoden und der erfahrungsgeleiteten Vorgehensweise in der Praxis ebenfalls ein wichtiges Thema. Befragt nach ihren Wünschen an die Universität stellten die Praktiker jedoch zuallererst fest, dass sie über weite Strecken mit der Informatik-Ausbildung zufrieden sind.

Die anwesenden Firmenvertreter halten es für sinnvoll, dass Studierende der Informatik in Kernkompetenzen der Disziplin eine solide Ausbildung erhalten und dass sie mit den neuesten Techniken und Trends vertraut sind. Das notwendige Detailwissen im Hinblick auf eine bestimmte Branche, so die übereinstimmende Meinung der Praktiker, erwerben die Absolventen besser in konkreten Projekten, zumal die Anwendungsbranche oft von Projekt zu Projekt variiert. Hier kann es also nur um die notwendige Offenheit der Studierenden für dieses Zusatzwissen gehen und um geeignete Arbeitspraktiken, um sich die notwendigen branchenspezifischen Detailkenntnisse schnell und zielstrebig anzueignen.

Obwohl es aus Sicht der Praktiker sinnvoll ist, dass die Studierenden eine Programmiersprache beherrschen, so scheint es doch unwesentlich zu sein, welche es ist. Die Programmiersprachen, die gerade up to date sind, ändern sich regelmäßig. Da auch mit älteren Programmiersprachen entwickelte Systeme heute noch im Einsatz sind und gewartet und gepflegt werden müssen, spielen selbst Cobol und Assembler in der Praxis eine nicht zu unterschätzende Rolle. Aus diesen Gründen erscheint die Beherrschung grundlegender programmiertechnischer Konzepte wesentlich vorteilhafter zu sein als die frühe Spezialisierung auf eine bestimmte Programmiersprache.

Informatik-Absolventen sollten zudem darauf eingestellt sein, dass sie sich in ihrem Arbeitsalltag mit Kunden abstimmen müssen, die häufig spezielle Vorstellungen davon haben, wie die Software entwickelt werden sollte. Darüber hinaus gibt es in der Regel beim Kunden bereits ein bestimmtes Set an Technologien, zu dem das zu entwickelnde System passen muss. Das bedeutet, dass Softwareentwicklung in der Praxis selten aus der innovativen technologischen Idee heraus bei Null beginnt. In der Mehrzahl der Fälle müssen technologische Innovationen im bestehenden Techniksetting des Kunden umgesetzt werden, wobei die Aufgabenstellung in der Regel unscharf ist und das eigentliche Entwicklungsproblem daraus erst noch abgeleitet werden muss. Daraus ergeben sich hohe Anforderungen an die Kommunikations- und Problemlösungsfähigkeit von Softwareentwicklern.

Um die Informatik-Ausbildung zu verbessern, scheint eine engere Kooperation der Universitäten mit den Unternehmen angebracht. Hier könnten die Studierenden einerseits notwendige Praxiserfahrungen sammeln, andererseits bestünde in der Ausbildung der notwendige Raum, diese Erfahrungen zu reflektieren und auf den Methoden des Software Engineering basierende Handlungsmöglichkeiten aufzuzeigen. Durch die Vermittlung von Praxiskontakten während des Studium ließe sich auch ein weiteres Problem zumindest ansatzweise angehen. Kleine Unternehmen aus der Region Chemnitz beklagen einen Mangel an Bewerbern für offene Stellen. Das mag einerseits an den ökonomischen Rahmenbedingungen in diesen Unternehmen und dem vergleichsweise niedrigen Lohnniveau liegen. Andererseits scheint es für interessierte Absolventen auch schwierig zu sein, diese kleinen Unternehmen kennen zu lernen, da sie Stellen selten öffentlich ausschreiben. Ein Kontakt mit der Universität über die Zusammenarbeit in konkreten Projekten könnte das gegenseitige Kennenlernen ermöglichen und den Absolventen der TU Chemnitz damit auch eine Einstiegsmöglichkeit in den Arbeitsmarkt bieten.

Speziell für das Software Engineering und die Forschungen in diesem Bereich wünschten sich die Praktiker, dass die Studierenden Methoden und Werkzeuge beherrschen, die in den Unternehmen und beim Kunden Akzeptanz finden. Neue Methoden oder Werkzeuge, die in der Universität entwickelt werden, müssen unbedingt zuvor für die Tauglichkeit in der Praxis geprüft werden. Denn häufig scheitert der Einsatz einer Methode daran, dass sie für komplexe Problemstellungen, wie sie in der Praxis die Regel sind, nicht brauchbar ist. Das zumindest scheint das Problem formaler Spezifikationen und CASE-Tools zu sein. Andere Methoden sind für den Alltag zu aufwändig oder Verlangen zu viel Spezialwissen von den Beteiligten. Ein Diskussionsteilnehmer brachte diese Problematik wie folgt auf den Punkt: „Welcher Kunde versteht UML?“

3 Fazit

Es bedarf einiger organisatorischer Anstrengungen, Wissenschaftler, Praktiker und Studierende zu einem gemeinsamen Austausch zu bewegen. Forschung, Entwicklungspraxis und Studium folgen jeweils einer eigenen Logik und sind mit bestimmten Zwängen verbunden. In der Praxis haben drängende Projekttermine verständlicherweise Vorrang vor der Pflege von Kontakten zur Wissenschaft. Auch in der Universität sind die Möglichkeiten begrenzt, solche Kontakte zu pflegen und auszubauen. Der durch die Befristung von Stellen bedingte häufige Wechsel der Mitarbeiter erschwert eine kontinuierliche Zusammenarbeit. Hinzu kommt, dass jede Seite ihre spezifischen Urteile und Vorurteile hat, die es zu thematisieren gilt. Es bedarf also einiger Anstrengung, schon allein, um eine gemeinsame Sprache zu finden. Trotzdem gehen wir davon aus, dass sich solche Zusammenkünfte lohnen und längerfristig gesehen alle Beteiligten davon profitieren können. Doch ein solcher regelmäßiger Austausch will erst einmal etabliert sein. In diesem Sinne hoffen wir, dass unser Workshop den Anfang einer längeren Zusammenarbeit bildet.

Literatur

Bremer, Georg (1998): Genealogie von Entwicklungsschemata. In: Kneuper, R.; Müller-Luschnat, G.; Oberweis, A. (Hrsg.): Vorgehensmodelle für die betriebliche Anwendungsentwicklung. Stuttgart; Leipzig: Teubner, S. 32-59

Elting, Andreas; Huber, Walter (2001a): Immer im Plan?: Programmieren zwischen Chaos und Planwirtschaft. In: c't , H. 2, S. 184-191

Elting, Andreas; Huber, Walter (2001b): Schnellverfahren: Mit Extreme Programming immer im Plan?. In: c't, H. 3, S. 186-191

Henninger, Annette; Sieber, Andrea (2001): Softwareentwicklung in kleinen Unternehmen in Ost- und Westdeutschland. In: Matuschek, I.; Henninger, A.; Kleemann, F. (Hrsg.): Neue Medien im Arbeitsalltag. Empirische Befunde – Gestaltungskonzepte – Theoretische Perspektiven. Wiesbaden: Westdeutscher Verlag, S. 37-54

Sieber, Andrea; Henninger, Annette (2001): Vorgehensmodelle, Projekttypen und Arbeitspraktiken als Bausteine zur realitätsnahen Beschreibung von Softwareentwicklung. In: Dilger, W.; Keitel, E. (Hrsg.): Kultur und Stil in der Informatik? Chemnitzer Informatik-Berichte, CSR-01-01, S. 51-63

Voß, Günter G. (1988): "Schalten und Walten", nichts für sture Bürokraten?: Eine Untersuchung der Bedeutung autonomen und innovativen Handelns von Schalterbeschäftigten in öffentlichen Verwaltungen. In: Bolte, K. M. (Hrsg): Mensch, Arbeit und Betrieb: Beiträge zur Berufs- und Arbeitskräfteforschung. Weinheim: VCH Acta humaniora, S. 55-93

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

Das Chemnitzer Informatik-Studium aus Sicht eines ehemaligen Studenten

Eine Analyse und fünf Gestaltungsvorschläge

René Stöckel
Albatross Industrie-Software
Erich-Weinert-Weg 14
09423 Gelenau
rene.stoeckel@web.de

Die Informatik-Ausbildung in Chemnitz hat einen guten Ruf. Die Mitarbeiter in der Ausbildung sind engagiert. Die Chemnitzer Studenten sind in den Unternehmen gern gesehen. Diesen Ruf gilt es zu halten und auszubauen. Er sollte gehalten werden, indem im Interesse der Studierenden ein gewisses Maß an Kontinuität gewahrt bleibt. Er sollte ausgebaut werden, indem mit Mut Konzepte zur Weiterentwicklung und Verbesserung der Informatik Ausbildung entwickelt werden.

Im folgenden Beitrag analysiere ich das Informatik-Studium auf der Grundlage meiner eigenen Studienerfahrungen, denn ich habe mit Unterbrechungen für meine Praktika von 1992 bis 1999 an der TU Chemnitz Informatik mit Spezialisierung in Rechnernetzen/Verteilten Systemen studiert. Daraus entwickle ich konkrete Vorschläge zur weiteren Verbesserung dieses Studienganges.

1 Das Chemnitzer Informatik-Studium

In der hardwarenahen Ausbildung im Studiengang Informatik kann sukzessiv, modular und aufeinander aufbauend Wissen auf drei hierarchisch miteinander verknüpften Ebenen aufgebaut werden:

Digitaltechnik → Rechnerarchitektur → Rechnersysteme.

In der softwarebezogenen Ausbildung findet sich keine derartige Logik. Statt dessen klafft meiner Meinung nach eine große Lücke zwischen Algorithmen, Datenstrukturen, Programmiersprachen und komplexen Software-Systemen wie Datenbanken und Betriebssystemen. Lediglich wer sich als Informatiker in seiner Ausbildung auf Softwaretechnologie spezialisierte, erlangte fundierte Kenntnisse, wie man von der Programmiersprache sicher und mit Methode zu jenen umfangreichen Systemen gelangt, über die man sich Spezialwissen aneignet. Derartige Kenntnisse betragen z. B. Methoden der Softwarearchitektur und des Softwareentwurf sowie Prozessmodelle der Softwareentwicklung.

Diese Lücke gilt es zu schließen, denn ich nehme an, dass die Mehrzahl der Informatik-Absolventen in ihrem Berufsleben mit Softwareentwicklung konfrontiert wird. Zunehmend trifft das auch auf die auf Hardwareentwurf spezialisierten Informatiker zu, denn auch sie kommen verstärkt mit Softwareentwicklung in Berührung, z. B. bei der DSP-Programmierung oder der

Herstellung von Chip-Karten. In der Praxis liegt demzufolge der Schwerpunkt auf Seiten der Softwareentwicklung. Dem muss man sich längerfristig stellen, denn die auch in der Forschung nach wie vor prognostizierten Schwierigkeiten in der Softwareentwicklung sind sicherlich auch ein Effekt von Bildungsmangel. Zudem habe ich in der Praxis beobachtet, dass es häufig zur Selbstüberschätzung der Informatiker oder aber zu fehlendem Selbstvertrauen bei der Mitarbeit an größeren Software-Systemen kommt.

2 Gestaltungsvorschläge

In diesem Teil meines Beitrages unterbreite ich fünf Vorschläge, wie sich diese Lücke zwischen den einführenden Grundlagen der Softwareentwicklung und der professionellen Entwicklung komplexer Systeme beseitigen lässt.

1. Praxisnahe Grundkenntnisse im Software Engineering als „Allgemeinbildung“

Die notwendigen Grundkenntnisse des Software Engineering stehen bisher nur im Wahlpflichtfach Softwaretechnologie in ausreichendem Maß zur Verfügung. Bei der Vermittlung derselben sollte dabei nicht nur Bezug auf theoretische Modelle, sondern auch auf praktisch bewährte Verfahren genommen werden. Für praktische Übungen ist das gezielte Ausprobieren verschiedener einzelner Methoden wünschenswert. Die dabei gemachten Erfahrungen sollten reflektiert und die jeweiligen Vor- und Nachteile der Methode herausgearbeitet werden.

Die Vermittlung der Grundkenntnisse durch das Fach Softwaretechnologie könnte durch Veranstaltungen der Lehrstühle, die für die Spezialisierungsrichtung im Hauptstudium verantwortlich sind, ergänzt werden. Sie könnten das Problem der Entwicklung komplexer Systeme aus ihrer Perspektive und ihren Praxiserfahrungen aufgreifen. Gleichzeitig muss die Vermittlung von Grundkenntnissen des Software Engineering, des Softwaredesigns und des Vorgehens von Entwicklern in der Praxis im Hauptstudium weiter an Bedeutung gewinnen und zu einem zentralen Baustein werden, auf den die anderen Informatik-Lehrstühle aufbauen können.

2. Verlegung des Softwarepraktikums ins Hauptstudium

Das Softwarepraktikum als Generalprobe der Softwareentwicklungs-Methoden wäre in der „Muße“ des Hauptstudiums besser platziert, nachdem praxisnah und intensiv in das Software Engineering eingeführt wurde.

Das Grundstudium ist relativ stark von Hektik geprägt. Es gilt eine große Anzahl von Vorlesungen und Seminaren zu besuchen, um die Voraussetzungen für das Vordiplom zu erfüllen. Ich habe in dieser Zeit die komplexe Aufgabe eines Softwarepraktikum als unangenehme Belastung empfunden. Ich hatte wenig Erfahrungen, wie ein Ingenieur professionell an Softwareentwicklung herangeht und war aufgrund der knappen Zeit für methodische Fragen eher desensibilisiert. Im Prüfungsstress des Vordiploms habe ich deshalb Punktabzug im Softwarepraktikum in Kauf genommen, um die Sache erst einmal „irgendwie“ zu erledigen.

Beim Anfertigen der Studien- bzw. Diplomarbeit war ich offen für die Anwendung der im Studium kennen gelernten Softwareentwicklungs-Methoden. Allerdings werden diese Arbeiten nach wie vor in der Mehrzahl von Einzelpersonen angefertigt. Die Chance, niveauvolles, koordiniertes Arbeiten mit den behandelten Softwareentwicklungs-Kenntnissen im Team zu erfahren, ist zu diesem Zeitpunkt dann schon vergeben.

Vielleicht ohne es zu wollen, fördert diese Organisation des Chemnitzer Informatik-Studiums folgende mentale Einstellung:

- Eine unprofessionelle, chaotische Entwicklungsweise ist unter Zeitdruck im Team die einzige Lösung.
- Ich kann nur im Alleingang methodisch reflektiert arbeiten.

Sie erweist sich in der Praxis aber meist als sehr hinderlich. Deshalb plädiere ich für eine Verlegung des Softwarepraktikums in das Hauptstudium. Der praktische Teil im Grundstudium sollte darauf ausgerichtete sein, gezielt und reflektiert verschiedenen Methoden im Einzelnen auszuprobieren.

3. Erfahren von Komplexität

Bei der Diskussion der Wünsche der Praktiker an das Software Engineering sprach ein Teilnehmer ein zentrales Problem von Forschungsarbeiten in diesem Bereich an: Die entwickelten Methoden scheitern an der Komplexität realer Aufgaben. Das bedeutet für die Ausbildung, dass die eingesetzten Methoden auch unter diesem Aspekt geprüft werden müssen. Das Softwarepraktikum (im Hauptstudium) scheint ein geeigneter Rahmen zu sein, das Bearbeiten komplexer Aufgaben im Team zu üben.

4. Wiederverwendung anwenden

In anderen Ingenieurdisziplinen steht die Fähigkeit, Bauteile zu verbessern, zu ersetzen oder neue Baugruppen zu bilden weit mehr im Vordergrund als in der Informatik. Auf den ersten Blick scheint das durch die reale Repräsentation der Bauteile wie bspw. im Maschinenbau auch einfacher zu sein. Demgegenüber habe ich in meiner Ausbildung Aufgaben häufig als ein Anfangen von „Null“, als neu bauen, neu entwickeln erfahren. War bereits Software vorhanden, dann wurde sie oft als schlecht und unzureichend eingeschätzt, auch weil die Entwickler nicht verstanden, wie die vorhandene Software intern funktioniert, welche Ideen im Entwurf des Quelltextes stecken. In der Regel war kaum einer in der Lage, die Teile, die gut und sinnvoll funktionierten, für seine Lösung zu übernehmen.

Durch eine gezielte Sensibilisierung der Studenten für die Arbeit anderer und durch Training der Analysefähigkeiten in der Ausbildung ließe sich meiner Meinung nach dieses Problem des Verstehens fremden Quelltextes wesentlich verbessern. Das beinhaltet gleichzeitig die Chance, aus der Analyse von Teilen frei verfügbarer Software viel über die Struktur und Konstruktion von Quellcode - mit den entsprechenden Vor- und Nachteilen - zu lernen. Denn die Quellcodes der frei verfügbaren Software stellen meist gute und elegante Programmierlösungen anschaulich dar.

Am Beispiel der frei verfügbaren Software kann auch eine systematische Vorgehensweise beim Analysieren und bei der Inspektion gelernt und gefestigt werden. Reengineering hatte in Chemnitz eine lange Tradition. In den Zeiten des real existierenden Sozialismus war es eine gängige Methode, sich unbekanntes technologisches Wissen anzueignen. Es empfiehlt sich, diese Fähigkeiten bei den Studenten wieder mehr zu entwickeln und zu pflegen, denn daraus ergibt sich auch heute noch ein technologischer Vorsprung. Mittlerweile gibt es auch in der Softwareentwicklung eine Reihe von abstrakten Begriffen, Metaphern und Entwurfsmustern für verschiedenen Funktionskomplexe. Wenn es gelingt, diese zu verstehen und in der richtigen Situation zu nutzen bzw. anzupassen, kann ich mir als Entwickler viel Aufwand und Mühe sparen.

Daher wäre es sehr nützlich, wenn sich in der pädagogischen Praxis die Softwareentwicklungs-Übungen auch auf ausgesuchte, verfügbare Software in für die Studenten beherrschbarem Umfang beziehen würden. Hier denke ich z. B. an Aufgaben wie:

Besorgen Sie sich den Quellcode der bash (csh, ash oder ksh, oder ...)

a) Zeichnen Sie grob eine Systemübersicht.

b) Wie wäre vorzugehen, wenn man den Algorithmus zum ... durch ... ersetzen will?

c) Führen Sie einen neuen Befehl „count <arg>„ ein, mit dem sich die Anzahl der ausgeführten Befehle der bash abfragen lässt. Was ist zu dabei zu beachten?

d) Wie würden Sie die Funktionalität in einem objektorientierten Modell strukturieren?

5. Software entwickeln im Unternehmen

Das Land Sachsen hat ein sehr breites Angebot an Fördermitteln insbesondere für produktionsnahe Dienstleister im IT-Bereich. Sie bieten Motivation und Hilfe für Studierende, sich in der Region zu engagieren. Bislang ist das Angebot an Vorlesungen, die über die unternehmerische Einbettung von Softwareentwicklungsarbeit konkret und praxisnah informieren jedoch sehr beschränkt. Es fehlen konkrete Informationen zu theoretischen und praktischen Abläufen von

- Ausschreibungsverfahren
- Konzeption und Angebot
- Auftrag und Pflichtenheft
- Lieferung und Abnahme

Eine derartige Vorlesungsreihe mit anschaulichen Beispielen aus der Branche wäre ein sehr nützlicher Beitrag für den Berufsstart bzw. die Existenzgründung der Informatiker und damit auch für die Region.

Ich denke, ich habe eine Reihe von Vorschlägen gemacht, die aus meiner Perspektive mit wenig Aufwand aber viel Effekt umgesetzt werden könnten. Ich wäre froh, wenn das die Mitarbeiter der TU Chemnitz, die ein Interesse an der Verbesserung der Lehre und des Studiums und damit an der Zukunft der Universität haben, auch so sehen würden. Die Nachfrage nach Studierenden aus Chemnitz sowie deren Aktivitäten zur wirtschaftlichen Entwicklung der Region könnten dabei auch im positiven Sinne die Diskussion zur Sicherung der Zukunft des Universitätsstandortes Chemnitz befördern.

Warum gibt es eine Kluft zwischen Theorie und Praxis im Software Engineering?

Petr Kroha
TU Chemnitz
Fakultät für Informatik
Professur Informationssysteme und Softwaretechnik
09107 Chemnitz
petr.kroha@informatik.tu-chemnitz.de

1 Präzisieren der Begriffe

Die meisten Probleme, und nicht nur im Software Engineering, entstehen, weil verschiedene Leute die selben Begriffe auf verschiedene Weise verstehen und interpretieren. Manchmal geschieht dies sogar mit Absicht. Damit wir uns zu der Frage im Titel des Beitrages äußern können, müssen wir also in erster Linie die Begriffe präzisieren. Was versteht man unter dem Begriff Theorie?

2 Was ist eine Theorie?

In der Mathematik ist „Theorie" ein Fachbegriff. Es ist ein System von Axiomen, Sätzen und Beweisen, das uns ermöglicht, formale Systeme zu definieren. Mathematik kennt keine anderen Systeme, und solange die Grundregeln des Baus von formalen Systemen korrekt benutzt werden, lassen sich beliebige formale Systeme definieren. In diesem Sinn hat Mathematik keine Praxis. Manchmal zeigt sich aber, dass die technischen oder naturwissenschaftlichen Systeme sich durch Abstraktion vereinfachen lassen und ein Modell bilden, das im Rahmen eines mathematischen formalen Systems beschreibbar ist. Oft hat es sich gezeigt, dass das entsprechende formale System schon vor mehreren hundert Jahren von Mathematikern definiert wurde.

In den Naturwissenschaften repräsentiert die Theorie eine mehr oder weniger widerspruchsfreie Menge von Hypothesen, die etwas darüber aussagen, wie die Natur konstruiert ist und wie sie sich entwickelt, und es wird versucht, diese Hypothesen durch Experimente entweder zu beweisen oder zu widerlegen. Wenn Widersprüche auftauchen, soll die Theorie durch eine bessere Theorie ersetzt werden, in der sich diese Widersprüche nicht mehr befinden, bzw. in der weniger Widersprüche enthalten sind. Oft sehen wir auf einem Fachgebiet mehrere konkurrierende Theorien, wobei jede mit eigenen Widersprüchen kämpft.

Es gibt andere Fachgebiete, in denen die Experimente entweder zu kompliziert und teuer (Wirtschaftswissenschaft) oder gar unmöglich sind (Philosophie). Hier lassen sich die Widersprüche nicht einfach finden und wir sehen oft Hypothesen, die mehr Glauben und Wünsche als Wissen darstellen, was ihre Autoren aber selbst nicht begreifen oder zumindest nicht zugeben wollen.

Im Alltag wird der Begriff „Theorie" oft spöttisch benutzt und bedeutet etwas Unnützliches, womit bestimmte Leute spielen, damit sie nicht arbeiten müssen. Eine andere Alternative ist, dass

mit dem Begriff „Theorie“ Schulkenntnisse gemeint sind. Das wird oft als ein Synonym für etwas benutzt, was praxisentfernt, praxisirrelevant ist, was nur auf Papier funktioniert und was in der Schule nur deswegen unterrichtet wird, weil die praxisentfremdeten Lehrer nichts Vernünftiges unterrichten können.

3 Wo sind die Ziele des Software Engineering?

Software Engineering ist eine Disziplin, die jemandem, der in einem Softwarehaus die Softwareentwicklung betreibt, Konzepte und Methoden anbieten soll, mit denen die Firma durch diese Tätigkeit Geld (auf dem Markt) verdienen kann. Der Gewinn kann direkt (Geld) oder auch indirekt (Image bei Kunden, zufriedene Mitarbeiter) sein.

Da Software Engineering nicht nur physikalische Systeme (Computer) und mathematische Systeme (Programmiersprachen) einschließt, sondern auch biologische Systeme (Kunden, Programmierer, Benutzer), soziologische Systeme (Team), ökonomische Systeme (Markt) und politische Systeme (Steuersystem und andere Vorschriften), kann man kaum von einer Theorie sprechen, die uns sagt, wie eine Firma auf dem Markt durch Softwareentwicklung Geld verdient.

Es geht also nicht darum, ob die Methoden des Software Engineering der Logik eines formalen Systems entsprechen wie in der Mathematik. Eine viel engere Ähnlichkeit gibt es zu naturwissenschaftlichen Systemen, aber man arbeitet nicht mit natürlichen sondern mit von Menschen künstlich geschaffenen Systemen.

In dem Prozess des Geldverdienens durch Softwareentwicklung sind so viele Parameter zu respektieren und so viele unerwartete Ereignisse möglich, dass man kein sinnvolles Modell konstruieren kann und deswegen auch von keiner Theorie sprechen kann.

Dies ist nicht so überraschend. In anderen Disziplinen des Ingenieurwesens, z.B. im Bauwesen oder Maschinenbau kann man auch kaum von der Theorie des Bauwesens oder der Theorie des Maschinenbaus sprechen. Die Methoden der Maschinenbautechnologie (z.B. Fließband) haben letztendlich auch das Ziel, den Gewinn der Firma zu erhöhen.

Wir sollten in diesem Zusammenhang die eigentliche Wissenschaft (Informatik) und ihre technologische Komponente (Software Engineering) unterscheiden. Als Beispiel benutzen wir den Unterschied zwischen Medizin und der medizinischen Technologie. Während die Heilmethoden (Medizin) unabhängig vom Gewinn der Ärzte entstehen, ermöglichen die technologischen Methoden (z.B. sieben Räume, in denen sich die Patienten in verschiedenen Stadien des Aus- und Anziehens befinden) dem Arzt, seine Praxis besser zu organisieren und mehr zu verdienen.

4 Beitrag der theoretischen Methoden im Ingenieurwesen

Es gibt theoretische Fachgebiete, die ihre Methoden für bestimmte kleine Unterprobleme des Ingenieurwesens anbieten. Für Software Engineering bietet sich Mathematik an, für das Bauwesen ist es Statik, für den Maschinenbau ist es Mechanik.

Wenn eine Methode angeboten wird, bedeutet es bei weitem noch nicht, dass die Methode verglichen mit dem Ziel des Software Engineering relevant ist, d.h. dass sie etwas nützt, wenn sie von einer Firma benutzt wird. Es gibt eine Grauzone von Methoden, deren Autoren zwar behaupten, dass diese Methoden ins Software Engineering gehören, aber allgemein sind diese umstritten, weil sie keiner Firma Gewinn bringen.

Bei den Autoren handelt sich manchmal um Trittbrettfahrer, die die Popularität des Software Engineering für ihre Karriere nutzen wollen, manchmal aber auch um begabte Leute, die weiter in die Zukunft sehen, als ihre Zeitgenossen. Im Anfangstadium lässt sich das nicht immer unterscheiden.

Manchmal werden diese Methoden schnell vergessen, ab und zu geschieht es, dass sich das Umfeld ändert und sie doch Gewinn bringen und dadurch auch ins Software Engineering integriert werden.

Es ist notwendig zu betonen, dass die durch Theoretiker untersuchten Unterprobleme zwar wichtig sind und ihre neue Lösung einen technologischen Vorsprung bedeuten kann, dass aber das Schicksal einer Firma von verschiedenen anderen Einflüssen viel stärker abhängen kann.

Manchmal zeigt sich aber, dass nicht einmal die von den Theoretikern angebotene Lösung von kleinen Unterproblemen schnell genug zustande kommt. Als Beispiel nehmen wir die Verifikation oder die formale Spezifikation. Aus den Erfahrungen der letzten 40 Jahre sehen wir leider, dass diese Methoden, mindestens bis jetzt, viel mehr dazu geeignet sind, ihren Trägern wissenschaftliche Titel zu bringen, als Gewinn einer Softwarefirma.

Das bedeutet nicht, dass diese Methoden keine Bedeutung haben oder dass die Forschung diese Wege verlassen soll. Es macht nur die Unterschiede zwischen Theorien, hier durch Mathematik repräsentiert, und dem Software Engineering klar. In der Mathematik werden Kenntnisse eindeutig mit Hilfe eines formalen Systems formuliert. Dazu werden deduktive oder induktive Methoden benutzt. Es gibt jedoch keine experimentelle Mathematik, obwohl sich jetzt mit sehr leistungsfähigen Computern solche Möglichkeiten in Ansätzen auch anbieten.

Verglichen damit, ist es im Software Engineering umgekehrt. Es ist eine Sammlung von Hinweisen, die einer Firma helfen können, die dazu aber leider meistens nur eine zeitlich begrenzte Gültigkeit haben. Jeder Hinweis basiert auf Erfahrung, beinhaltet Ausnahmen und hat in jedem Kontext der Anwendung verschiedene Vor- und Nachteile, von denen manche erst im Nachhinein klar werden, die aber für die Firma, die in eine bestimmte Technologie investiert, verheerende Folgen haben können.

In diesem Zusammenhang werden Erfahrungen im allgemeinen hoch geschätzt, sie beinhalten aber auch eine Gefahr. Keine Situation aus der Vergangenheit wiederholt sich im Software Engineering komplett und identisch in der Zukunft. Wenn wir Erfahrungen benutzen, benutzen wir eigentlich auch eine bestimmte Abstraktion. Dabei müssen wir auf bestimmte Einflüsse verzichten. Unsere Abschätzung kann aber falsch sein und wir verzichten auf die wesentlichen Eigenschaften. Praktisch ist bekannt, dass Erfahrungen aus der Vergangenheit nicht immer einen Vorteil für die Zukunft bedeuten. Manchmal sind sie eher eine Bremse. Solange wir aber nichts besseres haben, bleibt uns nicht anderes, als Erfahrungen trotz des bekannten Risikos zu benutzen.

Als Hauptunterschied zwischen theoretischen Fachgebieten und dem Software Engineering kann man betonen, dass Theorie nicht unter dem Druck des Marktes steht.

Wie der Druck des Marktes funktioniert, zeigen wir auf anhand der Geschichte der objektorientierten Programmierung, die schon mit SIMULA-67 beginnt. Diese Sprache entstand an der University of Oslo. Dort wurde auch der entsprechende Compiler geschrieben (der Autor dieses Beitrages hatte zufälligerweise die Ehre, ihn auf dem Computer CDC 6600 zu benutzen). Damals hatte aber IBM ihre Sprache PL/1 auf dem Markt durchgesetzt und massiv unterstützt. Die Ideen der Objekte mussten somit noch weitere 20 Jahre warten. Dadurch kann man illustrieren, dass die besten theoretischen Ergebnisse (in diesem Fall auf dem Gebiet der Programmierspra-

chen) für das Software Engineering nur eine begrenzte Bedeutung haben, weil die Einflüsse auf den Gewinn einer Softwarefirma sehr vielfältig sind und die Auswirkung von ihren einzelnen Komponenten nicht richtig vorhersehbar sind.

Wir benutzen noch ein Beispiel. Angenommen, dass das Problem der formalen Spezifikation inklusive der Verifikation und der Umwandlung der Spezifikation in ein lauffähiges Programm theoretisch gelöst würde. Welchen Einfluss könnte dies auf eine Softwarefirma haben?

Das erste Problem jeder Firma ist, genügend Aufträge von zahlungsfähigen Kunden zu erhalten. Dabei helfen die formalen Methoden nicht. Das nächste Problem ist, geeignete Mitarbeiter zu finden und anzustellen. Es hat sich schon gezeigt, dass dies ein Problem ist. Wenn schon ein Mitarbeiter angestellt ist, gibt es die Frage, wie schnell er die neuen formalen Methoden lernt, wie viel die Schulung und seine Fehler kosten, bis er das wirklich beherrscht. Wenn er dann die komplizierten formalen Methoden nutzen kann, stellt sich die nächste Frage, ob es nicht zu riskant ist, diese Technologie zu benutzen, weil, wenn dieser Mitarbeiter in der Hälfte eines Projektes kündigt, die Firma kaum schnell genug jemanden findet, der diese Technologie so gut beherrscht, dass er die Arbeit sofort fortsetzen kann. Es kann auch passieren, dass die Anwendung von formalen Methoden zwar die bessere Qualität des Produktes garantiert, aber gleichzeitig die Entwicklung verlängert und dass sich der Kunde inzwischen für ein Produkt der Konkurrenzfirma entschieden hat, das zwar keine so gute Qualität besitzt, jedoch inzwischen schon längst auf dem Markt ist.

Also, auf die Antwort, warum zwischen der Theorie und Praxis im Software Engineering eine Kluft besteht, gibt es eine einfache Antwort, und zwar: „Weil es keine Theorie des Software Engineering gibt!“

Was in der Praxis als Theorie im Software Engineering betrachtet wird, sind die Methoden aus der Grauzone, die zwar von Theoretikern aus anderen Fachgebieten angeboten worden sind, die aber noch keiner Firma Gewinn gebracht haben. Die Methoden, die schon Gewinn bringen, werden nicht als Theorie bezeichnet.

Welche Erkenntnisse bringen empirische Arbeiten dem Software Engineering?

Eine Antwort auf den Beitrag von Petr Kroha[1]

Christiane Floyd
Universität Hamburg
Fachbereich Informatik
AB Softwaretechnik
Vogt-Kölln-Straße 30
D - 22527 Hamburg
floyd@informatik.uni-hamburg.de

Ich habe meinem Vorredner, Herrn Kroha gerne zugehört und manches von dem, was er gesagt hat, will ich unterstreichen. Dennoch möchte ich mich - und zwar ziemlich deutlich - distanzieren von der geäußerten Auffassung zum Software Engineering. Dies erscheint mir so wichtig, dass ich eine Vorbemerkung dazu machen will, bevor ich zu meinem eigentlichen Thema komme. Ich möchte einen Vergleich bemühen. Wir wollen alle Geld verdienen. Wir sind alle dem Markt unterworfen, egal was wir tun. Trotzdem würde ich nicht sagen wollen, dass etwa die Medizin dazu da ist, damit die Ärzte Geld verdienen. Das glaube ich nicht. Ich glaube nicht, dass der gesellschaftliche Auftrag der Medizin allein darin besteht dafür zu sorgen, dass die im medizinischen Bereich tätigen Professionellen Geld verdienen, obwohl sie sicherlich zum Teil viel Geld verdienen, die Strukturen dies zweifellos begünstigen, obwohl das einerseits seinen Sinn hat und man es andererseits auch kritisieren kann.

Es ist selbstverständlich, dass bei der Softwareentwicklung in der Praxis der Markt eine wichtige Rolle spielt und dass sich Unternehmen wirtschaftlich verhalten müssen. Aber den gesellschaftlichen Auftrag des Software Engineering darauf zu reduzieren, dass es den Softwarefirmen bitte zu mehr Geld verhelfen möge, halte ich für unzulässig. Ich verlange von Software Engineering Methoden, dass zuverlässige Software entsteht, dass benutzungsgerechte Software entsteht. Zugleich muss diese auch wirtschaftlich herstellbar sein. Die von Herrn Kroha geäußerte Auffassung zur Rolle des Software Engineering greift meiner Meinung nach zu kurz. Was er angemahnt hat, würde ich eher auf der wirtschaftlichen Ebene ansiedeln. Es ist vorwiegend die Frage des Gespürs für den Markt, die der Geschäftsführer einer Firma haben sollte, und die durch eine betriebswirtschaftliche Ausbildung gefördert werden kann.

Dagegen verlange ich vom Software Engineering - wie von jeder Ingenieurswissenschaft - Kriterien und Techniken zur konstruktiven Entwicklung, die wissenschaftlich begründet sind. Ich verlange Theorien, um die in dieser Wissenschaft relevanten Produkte mit hoher Qualität und nach geordneten Verfahren herstellen zu können. Dazu dient eine Ingenieurswissenschaft.

[1] Dieser Beitrag basiert auf den Tonbandaufzeichnungen während des Workshops.

Ich stimme Herrn Kroha zu, dass es im Software Engineering keine genügend reichhaltige Theorie gibt. Ich erkenne alle Faktoren an, die er genannt hat, auch wenn ich sie nicht alle als „Systeme“ bezeichnen würde - die Bezeichnung macht einen Unterschied, weil bei mir nicht der Anspruch damit verknüpft ist, alle genannten Aspekte modellieren zu können, wie der Begriff System in den Raum stellt. Ich möchte daher lieber von Ebenen sprechen. Aber ich stimme zu, alle diese Ebenen spielen tatsächlich eine Rolle. Selbstverständlich gebe ich Herrn Kroha auch darin Recht, dass das Software Engineering seine Theorie in der Mathematik bzw. in der Logik gesucht hat. Das scheint mir durchaus gerechtfertigt, insofern es sich auf die Produkte als formale Texte bezieht. Die Grundlagen dafür kann man sich nur dort holen.

Was aber sicherlich im Software Engineering nur sehr mangelhaft herausgebildet wurde, ist eine Theorie, die sich mit dem Prozess der Herstellung dieser Produkte beschäftigt. Mit „Prozess“ meine ich hier nicht eine definierte Vorgehensweise, die der Praxis auferlegt wird, sondern das, was in der Praxis tatsächlich stattfindet. Das muss natürlich eine grundlegend andere Theorie sein. Es muss eine Theorie sein, die in Rechnung stellt, wie Menschen miteinander arbeiten, wie sie sich verständigen zwischen heterogenen Gruppen und bei Situationen von Stress und Druck. Wie sie sich verständigen auf der Basis von Ergebnissen, die umfangreiche Dokumente sind, deren Aussagekraft im Kontext oft nur schwer beurteilt werden kann. Dazu bräuchte man Verständnismodelle über menschliche Erkenntnis, Lernen und Kooperation aus anderen Disziplinen, und solche gibt es auch. Diese müssten in die Informatik importiert werden, und sie werden auch teilweise in die Informatik importiert. In diesen Verständnismodellen geht es um Kooperationsprozesse im Team und es geht um die Kommunikation zwischen Entwicklern und Benutzern. Diese Prozesstheorie scheint mir noch nicht ausreichend entwickelt, und vor allem wird sie nur an wenigen Standorten in die Lehre eingebracht.

Ein Problem, welches ich dabei sehe ist, dass das Software Engineering nicht verstanden hat, welches sein eigentlicher Forschungsgegenstand ist: Es hat sich auf das Verständnis der formalen und technischen Eigenschaften der Produkte konzentriert, was sicher wichtig ist, aber den Prozess zwischen den beteiligten Menschen weitgehend ausgeblendet. Deshalb ist meines Erachtens auch diese Theorie-Praxis-Kluft sehr groß und ich glaube, dass hier empirische Arbeiten – und das ist ja mein eigentliches Thema heute – sehr nützlich sind. Worüber im Software Engineering immer wieder geklagt wird ist, dass man etwas lernt und es in der Praxis nicht anwenden kann, weil die Praxis ganz anders ist. Sie ist chaotisch, da wird gebastelt, es gibt harte Termine und zermürbende Besprechungen, die alles Geplante wieder umwerfen. Wo bleiben dann die schönen Spezifikationen? Man sollte doch – machen wir doch! Dann kommt aber ein Termin, dazu machen wir zwar schnell ein Dokument, doch dann verliert es bald seine Relevanz, und im Extremfall werfen wir es wieder weg.

Diese Diskrepanz scheint mir darauf zu gründen, dass das Prozessverständnis des Software Engineering aus einer Zeit stammt, wo man sehr stark auf ein rein rationales Top-Down-Vorgehen gesetzt hat. Das hat man aus der ruhmreichen Geschichte des klassischen Engineering übernommen, wo es um solche Dinge wie den Bau einer Festung ging. Dort wurde dieses Prinzip im 17. Jhd. erfunden. Monsieur Vauban, ein Franzose, hat für Ludwig XIV. Festungen gebaut und dafür Pläne entworfen. Damals war das eine Sensation. Aus dieser Zeit stammt auch der Begriff Projekt. Er bedeutet Projektion nach vorn, nach einem festen Plan. Es war eine große Errungenschaft, diese systematische Vorgehensweise im Bauwesen einzuführen. Das Ingenieurwesen hat dieses Prinzip weiterverfolgt, sehr lange und bewundernswert erfolgreich, gerade in Deutschland.

Die Frage ist jedoch, warum scheitert dieses Prinzip in der Softwareentwicklung? Was ist denn in diesem Bereich das Neue? Wo liegt das Problem? Warum scheitert diese Prinzip auch zunehmend in heutigen Ingenieursentwicklungen in anderen technischen Bereichen?

Meiner Meinung nach gab es mehrere Veränderungen, die eine neue Qualität mit sich bringen. Immer wieder wird von Komplexität gesprochen, die eine sichere Vorausplanung erschwert. Dabei geht es nicht nur um die Komplexität des zu entwickelnden Produktes selbst, sondern auch um die Einbettung in eine Produktlandschaft mit zunehmend komplizierteren Wechselwirkungen und Kompatibilitätsbedingungen, die sich auch noch durch Weiterentwicklungen im Laufe der Zeit ändern. Dazu kommt, dass Software - anders als materielle Produkte - aus einem einheitlichen abstrakten Baustoff besteht, so dass es von vornherein keine Vorgaben für Zwischenergebnisse und Arbeitsteilung gibt. Deshalb hat Softwarearchitektur eine so fundamentale Bedeutung.

Die wichtigste Veränderung was Software anbetrifft erscheint mir aber, dass wir während der Softwareentwicklung lernen. Wir lernen in einer viel tiefgreifenderen Weise als damals beim Festungsbau. Natürlich hat man auch beim Festungsbau gemerkt, dass der Graben tiefer oder dass das Erdreich schwerer war, als man geglaubt hat. Deswegen gab es immer Abweichungen vom Plan. Aber es liegt im Wesen der Softwareentwicklung, Wissen aufzubauen. Wissen über die benötigte Funktionalität, die technisch vertretbare Art der Realisierung und den wünschenswerten Einsatz von Software, das man am Anfang nicht hat.

Der Grund, warum ich an Dokumente mit vollständigen Anforderungen einfach nicht mehr glaube, ist nicht primär, dass die Entwickler nicht sorgfältig recherchieren oder dass die Anwender sich nicht artikulieren können, sondern dass beide gemeinsam über etwas Abstraktes in die Zukunft projizieren müssen, was nicht verlässlich leistbar ist. Eine Festung kann man sich noch vorstellen, man kann sie zeichnen, man kennt die Materialien, man kann deren Festigkeit berechnen, man kann deren Herstellungskosten abschätzen. Aber bei der Art Produkt, die wir bei Software haben, sind die Verhältnisse anders. Es gibt viel mehr Freiheitsgrade. Computerfunktionen sind unmittelbar mit dem verzahnt, was Menschen tun. Deshalb gibt es eine Wechselwirkung zwischen dem menschlichen Arbeiten und Problemlösen und der Wirkungsweise der Software. Diese ist im einzelnen so im voraus nicht vorstellbar. Deswegen scheitern die linearen sogenannten Wasserfallmodelle, wo man Top-Down als zeitliche Richtlinie nimmt.

Es gibt jedoch Situationen, wo sie trotzdem anwendbar sind, und zwar wenn man vergleichbare Produkte öfter herstellt. Das ist durchaus der Fall in vielen Firmen. Ich komme beispielsweise ursprünglich aus dem Compilerbau. Die ersten Compiler waren chaotisch, aber recht bald hat man den Mehrphasen-Compiler erfunden und damit hatte man eine feste, stabile Architektur. Und diese war dann die Grundlage für solche Projekte. Wenn man also das erste Compiler-Projekt gemacht hat, war man im Chaos. Wenn man das zweite geplant hat, hatte man schon eine Vorstellung, wie Compiler gebaut sein sollen. Beim dritten war man sich schon ziemlich sicher bezüglich der Architektur. Und wenn man noch einmal ein viertes Compilerprojekt für die selbe Sprache gemacht hat, dann konnte man wasserfallartig vorgehen. Dann wusste man, was man spezifizieren muss, was man entwerfen muss, usw.

Ich glaube die Softwaretechnik begreift nur wenig von diesem Zusammenhang zwischen Lernen und Erfahrung aus Produktentwicklung, die übrigens im capability maturity model ausdrücklich angesprochen wird. Die Methoden, die wir haben, beschreiben immer die Produktentwicklung so, als ob man die Erfahrung schon hätte. Sie beziehen sich auf bereits erworbene Erfahrung. Deswegen bin ich der Überzeugung, dass die Disziplin sehr viel lernen kann von empirischen Untersuchungen - soweit eine Disziplin überhaupt lernfähig ist, was eine gute Frage ist. Empiri-

sche Untersuchungen richten - zumindest prinzipiell - den Blick darauf, was tatsächlich in der Praxis passiert.

Was das Software Engineering anbetrifft, so scheint mir die Unterscheidung zwischen interpretierenden Ansätzen, die beschreiben, und solchen, die normativ vorgehen, die also etwas vorschreiben sowie die Wechselwirkung zwischen beiden wichtig. Diese finden wir sowohl bei der Entwicklung und Einführung von Software in der Praxis als auch bei der Entwicklung und Einführung von Methoden. Jeder Technikentwickler schreibt letztlich etwas vor, weil er Technik entwickelt, von der er sich klar machen muss, wenn er ehrlich ist, dass sie in der Realität verändernd wirken wird, und dass sich die Benutzer darauf einstellen müssen. Und der Manager von Projekten schreibt natürlich erst recht etwas vor. Er möchte ja gern haben, dass die Termine eingehalten werden.

Wir müssen im Software Engineering dieses Verhältnis von Beschreiben, Verstehen und Interpretieren zu Vorschreiben verstehen. Mit unseren Methoden schreiben wir nämlich auch Dinge vor: die Aktivitäten, die durchgeführt werden sollen, ihr Verhältnis zueinander, ihre Ergebnisse.

Ich stelle jetzt noch zwei Überlegungen zum Top-Down-Vorgehen an. Erstens ist es nämlich sehr interessant, wenn man sich einerseits die Literatur anschaut und andererseits die Menschen fragt, was sie eigentlich unter Top-Down verstehen. Alle werden sagen: Top-Down hat sich bewährt. Fragt man jedoch näher, was sich bewährt hat, dann kommt als Antwort: Ja die Produkte müssen klar strukturiert sein. Niemand bezweifelt das. Es ist aber ein grundlegender Unterschied, ob man über das Produkt oder den Prozess spricht. Zweitens wissen alle, dass man immer nur eine Sache zugleich gut machen kann. Daraus folgt eine logische Trennung zwischen Ebenen wie Analyse, Entwurf und so weiter. Wie hängt das jedoch mit dieser Vorstellung zusammen, dass man mit dem Abstrakten anfängt, erst entwirft und dann implementiert, möglichst ohne Rückgriffe? Das ist ein idealisiertes Vorgehen, wie Menschen tatsächlich nicht denken. Und es findet ja auch in der Praxis nicht statt.

Dazu gab es große Kontroversen gerade im Software Engineering, und zwar sowohl in der Wissenschaft als auch beim Versuch, Methoden für die Praxis zu entwickeln. Ich möchte an dieser Stelle auf Herrn Dijkstra als prominenten Vertreter dieses Vorgehens verweisen. Er hat diese Methode vehement propagiert und sie durch Fallstudien untermauert, die beispielhaft zeigen sollten, wie er selbst Programme entwickelt. In seinen Papieren hatte es dabei den Anschein, als ob ihm immer durch höhere Eingebung der nächste Schritt zufallen würde.

Sein Gegenspieler war Peter Naur, der die empirische Forschung in der Softwaretechnik begründet hat. Er hat mit einer Gruppe von Studierenden ein Experiment gemacht. Er hat allen die selbe Problemstellung gegeben und sie protokollieren lassen, was sie machen. Sie mussten mit der Stoppuhr aufschreiben, was sie in welcher Reihenfolge gemacht haben. Dabei hat sich nicht nur herausgestellt, dass alle die unterschiedlichen Aktivitäten wie entwerfen, programmieren oder testen in unterschiedlicher Reihenfolge miteinander verzahnt haben, sondern auch, dass die Gewichtungen der Aktivitäten ganz unterschiedlich waren. Daraufhin hat Naur an Dijkstra den Vorwurf gerichtet, dass er lügt.

Für die Praxis der Softwareentwicklung ist das Arbeiten Einzelner an kleinen Problemstellungen und die wissenschaftliche Kontroverse darüber nur bedingt interessant. Vielmehr geht es um die in Firmen vorgeschriebenen Prozessmodelle, deren ursprüngliche Form von Barry Boehm unter dem Namen Wasserfallmodell bekannt wurde. Sie war zunächst rein linear in Phasen beschrieben, obwohl Rückgriffe (die durch die Wasserfallmetapher nicht gerade nahegelegt werden!) von vielen Autoren erlaubt wurden. Die weitergehende Kritik an diesen Modellen wurde unter Stichworten wie evolutionäre Systementwicklung und Prototyping geführt.

David Parnas vermittelt zwischen diesen beiden Positionen. Er sagt in seinem Papier: Wir brauchen einen geordneten Prozess im Engineering. Wir müssen uns in einem Setting wie in einer Firma darauf verständigen, welches Idealbild wir zusammen zu implementieren versuchen, um eine gewisses geordnetes Vorgehen anzustreben. Aber dann ist die Frage: Streben wir ein Idealbild an, das im Widerspruch steht zu dem, wie Menschen arbeiten, oder eines, welches sich besser an die menschliche Arbeitsweise anschmiegt. Etwas wissenschaftlicher könnte man sagen: Schreiben wir eine rein deduktiv-rationale Vorgehensweise vor bei der Softwareentwicklung oder ermutigen wir ein empirisch-experimentelles Vorgehen? Das methodische Umdenken, das damit verbunden ist, hat dazu geführt, das heute von vielen Autoren iterative Vorgehensmodelle (wie zum Beispiel im Unified Process) empfohlen werden.

In welchen Fällen und wie ein iteratives Vorgehen nötig ist, darüber kann und muss man im Einzelfall diskutieren. Aber wenn das so ist, dann braucht man dieses Feedback aus der Praxis, um überhaupt ein Vorgehensmodell darauf abzustimmen. Damit darf man aber eben nicht den Anspruch fallen lassen, ein geordnetes Verfahren zu haben. Dieser Anspruch ist wichtig. Es geht sicher nicht darum zu sagen, weil wir jetzt in drei Firmen beobachtet haben, dass gebastelt wird, finden wir uns damit ab. Das ist damit nicht gemeint. Die deskriptiven Berichte aus der Praxis sollten die Grundlage für ein vertieftes Nachdenken über das sein, was man dann wieder normativ einbringt.

Um mit Rollenmodellen zu argumentieren - wie schon erwähnt, befassen sich Ingenieure allgemein mit Veränderungen. Sie verändern die Wirklichkeit, indem sie technische Produkte herstellen. Diese technischen Produkte bringen enorme Veränderungen mit sich in der Wirklichkeit. Kristen Nygaard hat deshalb Softwareentwickler als „agents of change“ bezeichnet. Aus diesem Grund sollten sich Softwareingenieure klar machen, was in der Wirklichkeit stattfindet. Das können sie aber nur, wenn sie den Kontakt zur Praxis suchen. Darum plädiere ich dafür, dass solch ein Austausch stattfindet.

Gleichzeitig möchte ich jedoch davor warnen, an diesen Austausch zwischen Wissenschaft und Praxis naiv heranzugehen. Sobald man Empirie betreibt, begibt man sich auf sozialwissenschaftliches Terrain. Dann stellt sich die Frage, wie sich das Software Engineering von sozialwissenschaftlichen Untersuchungen über Software Engineering abgrenzt. Ist es dann noch die gleiche Disziplin oder sind es zwei verschiedene Disziplinen, die zusammenarbeiten? Das ist eine ungelöste Frage, zu der man verschiedener Ansicht sein kann. Sagt man, es ist dieselbe Disziplin, dann müsste man in diese Disziplin Theorien und Methoden aus den Sozialwissenschaften importieren. Diese müsste man dann auch als Theorien und Methoden in der Informatiklehre verankern. Wir tun das ansatzweise. Denn sozialwissenschaftliche Methoden wie Ethnographie sind unter anderem für das Requirements Engineering sehr relevant. Auch in Skandinavien wird das ziemlich viel gemacht. Da gibt es auch viel Literatur.

Man muss sich ferner darüber unterhalten, was denn überhaupt adäquate Untersuchungsmethoden für die Softwareentwicklung sind. Quantitative Methoden lassen sich in der Regel nicht anwenden, da einzelne Projekte Gegenstand des Interesses sind. Bei den qualitativen Methoden stellt sich die Frage nach der Aussagekraft. Man muss also auch Methodenforschung betreiben um sich klar zu machen, wie eine Empirie stattfinden soll, wenn man daraus eine wissenschaftliche Untersuchung machen will.

Wichtige Beiträge zur Methodendiskussion in der Softwaretechnik stammen aus Skandinavien, insbesondere aus Dänemark. Ausgangspunkt ist die auf Naur zurückgehende Tagebuchmethode, welche die Historie von Projekten festhält. Seit den frühen Achtziger Jahren haben Lars Mathiassen und andere Softwareentwicklungsprojekte empirisch untersucht. Darauf aufbauend wurde

der Ansatz reflection in action entwickelt, bei dem letztlich eine gegenstandsbegründete Theorie (grounded theory) anhand sorgfältig protokollierter Beobachtungen aus Projekten erarbeitet wird.

Eine weitere Frage ist, wie man mit der Tatsache umgeht, dass jeder, der Empirie betreibt, gleichzeitig interveniert. Der Untersuchende geht nicht voraussetzungslos in eine Anwendungswelt, er bleibt nicht außen vor, neutral und mit dem berühmten objektiven Blick, um zu sehen, was passiert. Ganz im Gegenteil, jede Beobachtung ist eine Art teilnehmende Intervention, denn bereits dadurch, dass eine Untersuchung stattfindet, wird sich einiges ändern. Das Tun des Untersuchenden ist in keiner Weise unschuldig. Empirische Untersuchungen, die mit Softwaretechnik zusammenhängen, haben die Tendenz, Dinge offen zu legen, die man vielleicht nicht sagen will, so z.B., dass man eben tatsächlich chaotisch vorgeht und nicht so, wie der Chef es gerne hätte - immer nur anhand der Dokumente. Solche Dinge kommen dann heraus. Hier spielen die Frage des Vertrauens eine Rolle und auch dessen, was ich als Beobachter oder Beobachterin in solche Untersuchungen hinein trage. Dann stellt sich die Situation eher als ein Gespräch dar zwischen der Person, welche die Untersuchung gemacht hat, und den konstruktiv-orientierten Softwareingenieuren und Managern bzw. den management-orientierten Software-Ingenieuren, die dann wirklich versuchen zu verstehen, wie besser weitergearbeitet werden kann.

Zusammenfassend möchte ich für empirische Arbeiten im Software Engineering plädieren, weil ich sie auf den drei Ebenen Softwareentwicklung und -einführung, Entwicklung von Methoden (insbesondere von Vorgehensmodellen) sowie Rollenverständnis von Softwareingenieuren und -ingenieurinnen für bedeutsam halte. Ich habe versucht dieses Thema, welches mir gestellt wurde, kurz zu beleuchten. Es war ein anderes Thema als das von Herrn Kroha. Gleichzeitig sollte ich ihm aber antworten. Daher habe ich es im Anschluss an seine Ausführungen für wichtig befunden, auch meine Auffassung von Software Engineering kurz darzustellen und die Ausführungen zu meinem eigentlichen Thema damit in Verbindung zu bringen.

Personenverzeichnis

Prof. Dr. W. Dilger studierte evang. Theologie, Mathematik und Informatik an den Universitäten Tübingen, Heidelberg, Göttingen und Karlsruhe. Er promovierte 1982 an der Universität Kaiserslautern und habilitierte sich 1986 an der Universität Kaiserslautern. Er arbeitete als wissenschaftlicher Mitarbeiter an der Universität Konstanz im Sonderforschungsbereich Linguistik, am Institut für Deutsche Sprache in Mannheim, an der Universität Kaiserslautern und am Fraunhofer-Institut für Informations- und Datenverarbeitung in Karlsruhe. Von 1989 bis 1993 war er Professor für Praktische Informatik an der EUROPEAN BUSINESS SCHOOL Schloss Reichartshausen. Seit 1993 leitet er die Professur für Künstliche Intelligenz an der TU Chemnitz. Seine Forschungsschwerpunkte sind Maschinelles Lernen, Data Mining, Multiagentensysteme und Robotik.

Dr. Yvonne Dittrich hat 1997 and der Universität Hamburg zum Thema „Computeranwendungen und sprachlicher Kontext - zu den Wechselwirkungen zwischen normaler und formaler Sprache bei Einsatz und Entwicklung von Software“ promoviert. Am Blekinge Institute of Technology in Ronneby, Schweden, hat sie die Forschungsgruppe „Use Oriented Design and Development of Software“ aufgebaut. Ihr Forschungsschwerpunkt ist die Entwicklung und der Einsatz von Methoden, die Anwendungsorientierung bei der Softwareentwicklung ins Zentrum stellen. Dabei verfolgt sie einen empirischen Ansatz. In verschiedenen Projekten in Kooperation mit industriellen und unternehmerischen Partnern steht „Design for Design in Use“, die Entwicklung von flexibler Software, die an sich verändernde Nutzungskontexte angepasst werden kann, im Fokus.

Prof. Dr. Christiane Floyd studierte Mathematik und promovierte 1966 an der Universität in Wien. Sie arbeitete von 1966-68 als Systemprogrammiererin in der Compilerentwicklung bei Siemens in München. Von 1968-1973 war sie wissenschaftliche Mitarbeiterin an der Fakultät für Informatik an der Universität in Stanford (USA). Von 1973-77 arbeitete sie als Senior-Beraterin für Softwareentwicklungs-Methoden bei Softlab in München. 1978 wurde sie Professorin für Softwaretechnik an der TU Berlin. Seit 1991 leitet sie den Arbeitsbereich Softwaretechnik an der Universität in Hamburg. Ihre Forschungsschwerpunkte sind evolutionäre und partizipative Systementwicklung, menschengerechte Systemgestaltung sowie die erkenntnistheoretischen Grundlagen der Softwareentwicklung und -anwendung.

Dipl. Math. Wolfgang Häcker studierte von 1961-66 Mathematik an der Universität in Leningrad (heute St. Petersburg). Von 1966-90 war er Mitarbeiter, ab 1969 Abteilungsleiter auf dem Gebiet „Automatisierte Projektierung“ im Kombinat Robotron (incl. Vorgängereinrichtungen). Der Begriff „Automatisierte Projektierung“ wurde im RGW-Bereich anstelle von CAD, CAM verwendet. Seit Oktober 1990 ist er Bereichsleiter der Anwendungsentwicklung in der IT-Services und Solutions GmbH (incl. der Vorgängerfirmen csd und csg).

Dr. phil. Annette Henninger studierte von 1987-93 Politikwissenschaft an der Freien Universität Berlin. Ihre Promotion über Frauenförderung in der Berliner Arbeitsmarktpolitik schloss sie 1999 ab. Seit 1998 arbeitet sie als wissenschaftliche Mitarbeiterin im DFG-Projekt „Softwareentwicklung in der Praxis im Kulturvergleich“. Ihre Arbeitsschwerpunkte sind die Themen Arbeit, Arbeitsmarktpolitik, Geschlechterforschung, Technik und Regionalpolitik sowie qualitative Sozialforschung.

Prof. Dr. Petr Kroha hat nach internationalen Erfahrungen an Universitäten und in der Industrie seit 1994 an der Fakultät für Informatik der TU Chemnitz die Professur für Informationssysteme und Softwaretechnik übernommen. Seine Ansichten und Meinungen über Software Engineering hat er in seinem Buch „Softwaretechnologie" (Prentice Hall, 1997) beschrieben. Zur Zeit arbeitet er an mehreren Projekten, die sich mit Anforderungsspezifikationen (TESSI), mit parallelen Ansätzen in CASE-Werkzeugen (OPAS) und mit spezifischen Eigenschaften von Informationssystemen (WEBIS, ADONIS) beschäftigen.

Dipl.-Phys. Uwe Lünstroth, geb. 1963, studierte an der Universität Stuttgart Physik und Philosophie (experimentalphysikalische Diplomarbeit 1992) und promoviert derzeit mit einer Arbeit über Carl Friedrich von Weizsäcker am Lehrstuhl für Technikphilosophie der BTU Cottbus. Er war von Oktober 1996 bis November 2001 als wissenschaftlicher Mitarbeiter in zwei BMBF-Projekten zum demographischen Wandel und dessen Auswirkungen auf die Innovationsfähigkeit der Softwareentwicklung tätig, in denen Beschäftigungsmöglichkeiten älter werdender Softwareentwickler vorgeschlagen und mit Unternehmenspraktikern diskutiert wurden.

Dr. Gerd Paul, Jahrgang 1949, studierte in Frankfurt und Berlin Soziologie. Nach dem Examen arbeitete er zwei Jahre als Berater in einer Management-Consultant Firma und wechselte dann zum Frankfurter Institut für Sozialforschung, an dem er bis zum Ende der 90er Jahre beschäftigt war. Dort arbeitete er an mehreren empirischen Studien, die techniksoziologischen Fragen nachgingen. Er schrieb regelmäßige Beiträge zum „Jahrbuch Sozialwissenschaftliche Technikforschung". 1989 promovierte er in Frankfurt mit einer Arbeit über das berufliche Selbstverständnis von Ingenieuren. Seine letzte Studie in Frankfurt untersucht „Innovation in der Softwareindustrie" (Campus 1999). Nach einem Forschungsaufenthalt in Spanien 1999, wo er mit einem Marie Curie Stipendium der EU Gastprofessor an der Universität Almeria war, wechselte er zum Soziologischen Forschungsinstitut Göttingen über, an dem er zur Zeit eine Studie über Telekooperation durchführt.

Dipl.-Inf. Andrea Sieber studierte von 1988-94 Informatik an der TU Chemnitz und der NC State University in Raleigh. Seit 1998 arbeitet sie als wissenschaftliche Mitarbeiterin im DFG-Projekt „Softwareentwicklung in der Praxis". Sie forscht zum Schwerpunkt Informatik in der Arbeitswelt und beschäftigt sich insbesondere mit Arbeitspraktiken in der Softwareentwicklung sowie Prozessmodellen.

Prof. Dr. Sabine Sonnentag ist Professorin für Arbeits- und Organisationspsychologie an der Technischen Universität Braunschweig. Sie studierte Psychologie an der FU Berlin, promovierte an der TU Braunschweig und arbeitete danach an der Justus-Liebig Universität Giessen, wo sie sich habilitierte, der Universiteit van Amsterdam und der Universität Konstanz. Sie forscht zu Expertise und Leistungsstärke (u.a. im Bereich Softwareentwicklung), Lernen in der Arbeit, Gruppenarbeit sowie arbeitsbezogener Erholung.

Dipl. Inf. Frank Stockmann studierte von 1988-94 Informatik an der TU Chemnitz mit Vertiefung Praktische Informatik. Bereits während seines Studium beteiligte er sich an der Entwicklung eines Prototypen zur dreidimensionalen Bilddarstellung auf PC-Basis. Mit verschiedenen Projekten in diesem Bereich machte er sich zusammen mit Kommilitonen 1992 selbständig. Heute entwickelt er mit internationalen Kunden in verschiedenen medizinischen Anwendungsbereichen und ist für die Leitung und Qualifikation der Softwareentwickler der Firma zuständig.

Dipl. Inf. René Stöckel absolvierte von 1992-99 eine Informatik-Ausbildung an der TU Chemnitz mit Spezialisierung Rechnernetze/Verteilte Systeme. In seiner Diplomarbeit beschäftigte er sich mit interaktiver 3D-Visualisierung von Netzwerkstrukturen. Bereits während seines Studiums war er als Praktikant bei Hewlett Packard und Philips Industrial Electronics tätig. Nach dem Abschluss seines Studiums arbeitete er als Softwareentwickler bei der Artemedia AG. Derzeit ist er als selbstständiger Unternehmer tätig.

Dr. Friedrich Strauss schloss 1989 an der RWTH-Aachen sein Studium als Diplom-Informatiker ab, promovierte danach an der Albert-Ludwigs-Universität Freiburg im Bereich der Software-Ergonomie und Dialogsteuerungen und arbeitet seit 1995 bei sd&m. Er hat mittlerweile in allen Phasen und alle Rollen eines Projekts gearbeitet und ist seit 2000 Senior-Berater mit Schwerpunkt Benutzerschnittstellen.

Dipl.-Ing. Günter Tröger, Jahrgang 1940. Begann nach dem Studium der Angewandten Mechanik an der TU Dresden als Softwareentwickler zu arbeiten. Bis 1989 war er neun Jahre als Direktor für Forschung und Softwareentwicklung im Datenverarbeitungszentrum Chemnitz tätig. Seit 1990 ist er Geschäftsführer der Systemhaus Chemnitz GmbH. Seine Unternehmensphilosophie orientiert sich am fortgeschrittenen Stand der Softwaretechnologie. Den Schwerpunkt bildet heute die Entwicklung Web-basierter Business-Software mittels Java.