A. M. Heinecke, H. Paul (Hrsg.): Mensch & Computer 2006: Mensch und Computer im Struktur*Wandel*.
München, Oldenbourg Verlag, 2006, S. 93-102

CCS – Eine Methode zur kontext-übergreifenden Softwareentwicklung

Matthias Finck, Hartmut Obendorf, Monique Janneck, Dorina Gumm

Universität Hamburg, Department Informatik

Zusammenfassung

Arbeiten zu partizipativer Softwareentwicklung befassen sich überwiegend mit der Entwicklung von für jeweils eine Organisation maßgeschneiderten Softwaresystemen. Zunehmend drängt jedoch die Anpassung bzw. Aneignung existierender Software stärker in den Vordergrund, wodurch die gleiche Software in unterschiedlichen Anwendungskontexten genutzt wird. Verschiedene Kontexte und neue Formen der Arbeit – wie z.B. virtuelle Netzwerke – in den Prozess einzubeziehen, stellt partizipative Softwareentwicklung vor neue Herausforderungen, die einer Erweiterung des klassischen Methodenrepertoires bedürfen. Mit der *Kommentierten Sammlung von Fallbeispielen* (CCS) stellen wir eine Methode vor, die die Nutzungs- und Weiterentwicklungsphasen kontextübergreifender Softwareentwicklung adressiert und berichten von unsere Erfahrungen aus ihrem Einsatz.

1 Kontextübergreifende Softwareentwicklung als Herausforderung für partizipative Entwicklung

Partizipation der NutzerInnen am Softwareentwicklungsprozess ist ein wichtiger Bestandteil evolutionärer Entwicklung (Floyd & Züllighoven 2002) und folgt traditionell zwei Zielsetzungen (vgl. z.B. Bødker et al. 1995, Floyd et al. 1989): Zum einen soll der Einbezug der AnwenderInnen als ExpertInnen für ihren jeweiligen Arbeitskontext die *Qualität* der entwickelten Softwareunterstützung verbessern. Zum anderen spielt, ursprünglich vor dem Hintergrund gewerkschaftlicher Bewegungen und des Leitbilds der *Humanisierung der Arbeit,* die Beteiligung von ArbeitnehmerInnen an der Gestaltung ihrer Arbeitsbedingungen (*empowerment*) eine wichtige Rolle: AnwenderInnen sollen durch die Beteiligung an Softwareentwicklungsprozessen Einfluss auf Veränderungen ihrer Arbeitsorganisation nehmen können.

Diese Tradition führt dazu, dass partizipative Entwicklung (oder *participatory design*, PD) vor allem im Kontext großer Organisationen betrachtet wird (vgl. Greenbaum et al. 1994). Der Wandel der Arbeitswelt mit den resultierenden flexibleren Modellen nicht-klassischer Organisations- und Arbeitsformen, wie beispielsweise virtuellen Organisationen und Netzwerken (vgl. Wehner 2001), verlangt nach einem anderen Blickwinkel und nach einer Über-

prüfung des klassischen Repertoires der PD-Methodologie (vgl. z.B. Törpel 2001): Zum einen sind die relevanten Akteure in verteilten Kontexten mit flexiblen oder unstetigen Organisationsstrukturen u. U. nicht eindeutig identifizierbar oder stehen nur eingeschränkt zur Verfügung. Zum anderen rückt statt der Entwicklung maßgeschneiderter großer Softwaresysteme für eine Organisation die Anpassung und Aneignung existierender – teilweise frei verfügbarer – Software stärker in den Vordergrund (vgl. Pipek 2005). Für die (Weiter-)Entwicklung so genutzter Software entstehen damit Herausforderungen, die auch aus der Produktentwicklung bekannt sind, z.B. große und unbekannte sowie sehr unterschiedliche Nutzungsgruppen (Grudin 1993). „Empowerment" bedeutet in diesem Rahmen zunehmend die Vermittlung von Medienkompetenz, um die NutzerInnen zu befähigen, die geeignete Software zur Unterstützung ihrer Aufgaben auszuwählen und anzupassen. Dies verlangt den NutzerInnen im partizipativen Prozess über die möglichst genaue Beschreibung der eigenen Arbeitsabläufe hinaus ab, auch über die bisherige Nutzung von Software reflektieren und urteilen zu können. Zudem verschiebt sich der Fokus von der frühen Phase der Spezifikation *vor* der Systemeinführung hin zu späteren Phasen der Nutzung und damit auch auf Anforderungsänderungen (Lam et al. 1999). Die partizipative Gestaltung und Begleitung derartiger Anpassungs- und Aneignungsprozesse wurde im Rahmen von PD bislang wenig betrachtet.

Die zunehmende Nutzung vorhandener (off-the-shelf) Software führt dazu, dass Software sich in sehr unterschiedlichen Anwendungsdomänen verbreiten kann und eine partizipative Entwicklung somit immer neue Kontexte berücksichtigen muss. Wir schildern in unserem Beitrag als Beispiel die Entwicklung einer zunächst spezialisierten Groupware, die schließlich als Open-Source-Software im Hinblick auf neue Nutzungskontexte weiterentwickelt wurde. Von Beginn an wurden partizipative Methoden eingesetzt, deren Zielrichtung sich jedoch mit der zunehmenden Zahl von NutzerInnen und der Verbreitung in neue Anwendungsbereiche änderte: Anstatt die Handlungspraxis in einem Bereich bestmöglich zu unterstützen, müssen nun die Anforderungen und Bedürfnisse verschiedenster Nutzungsgruppen gleichermaßen berücksichtigt werden, um ein Aufspalten der Entwicklung zu verhindern.

Im Rahmen einer solch *kontextübergreifenden Softwareentwicklung* gestaltet sich Partizipation als Aushandlungsprozess zwischen EntwicklerInnen und NutzerInnen einerseits sowie NutzerInnen verschiedener Kontexte andererseits, mit dem Ziel, ein gegenseitiges Bewusstsein für die unterschiedlichen Anforderungen zu erzeugen. Da sich die tägliche Arbeits- und Handlungspraxis der Beteiligten stark unterscheiden kann, was eine Diskussion auf der Ebene konkreter Aufgabenunterstützung erschwert, muss Kommunikation zwischen den Kontexten auf einer abstrakteren Ebene von *Werten und Leitbildern*, die der jeweiligen (Arbeits-) Organisation zugrunde liegen und die Grundlage für Designideen bilden, stattfinden.

Sowohl das Ermöglichen eines Austauschs der NutzerInnen untereinander als auch die abstrakte Reflexion über Werte erfordern neue Partizipationsmethoden. Wir stellen in diesem Beitrag mit *CCS* (steht für *C*ommented Collection of *C*ase *S*tudies) eine solche Methode vor, die es NutzerInnen aus verschiedenen Kontexten ermöglicht, ihre Erfahrungen in der Nutzung sowohl untereinander auszutauschen als auch in den Entwicklungsprozess einfließen zu lassen, indem sie szenarioartig den Umgang mit der Software schildern. Vervollständigt wird die Sammlung durch einen Kommentar der EntwicklerInnen, die auf der Basis der Fallbeispiele verschiedene Designoptionen ausloten und bewerten.

In den folgenden Abschnitten veranschaulichen wir zunächst anhand unseres eigenen Entwicklungsprozesses die Notwendigkeit, neue Partizipationsmethoden zu entwickeln. An-

schließend erläutern wir CCS, die Methode der kommentierten Sammlung von Fallbeispielen, ihren Einsatz im Entwicklungsprozess und zeigen ein Anwendungsbeispiel. Eine Diskussion des Nutzens von CCS als PD-Methode beschließt den Beitrag.

2 Fallstudie: Die CommSy-Entwicklung

Die Entwicklung der Groupware CommSy begann 1999 im Rahmen einer universitären Arbeitsgruppe, die ihre eigene Arbeitsorganisation – insbesondere Austausch und Kommentierung von Arbeitsergebnissen zwischen den Präsenztreffen – webbasiert unterstützen wollte. Einige an Webprogrammierung interessierte TeilnehmerInnen implementierten einen Prototypen, der im engen Austausch mit der Arbeitsgruppe weiterentwickelt wurde. Bald begannen einige Mitglieder, die entstehende Groupware auch zur Unterstützung ihrer Lehrveranstaltungen einzusetzen und resultierende Anforderungen zu formulieren. Über persönliche Bekanntschaften verbreitete sich die Nutzung schnell über verschiedene Institute hinweg. Der – im Wesentlichen durch freiwilliges Engagement getragene – Entwicklungsprozess blieb dabei durch eine enge Beteiligung der NutzerInnen und kurze Entwicklungszyklen gekennzeichnet.

Ab 2001 wurde CommSy als Teil eines E-Learning-Forschungsprojektes weiterentwickelt und evaluiert. Der Fokus lag dabei auf der Unterstützung *projektartiger Lehrveranstaltungen* im universitären Umfeld. In diesem Rahmen konnte die Nutzung von CommSy kostenlos angeboten werden, wodurch sich die Nutzungszahlen abermals stark erhöhten. Der enge Kontakt mit den NutzerInnen, beispielsweise im Rahmen von Workshops, blieb bestehen, der partizipative Prozess wurde jedoch um Methoden ergänzt, die den Einbezug einer größeren Anzahl von NutzerInnen erlaubten, wie beispielsweise Online-Fragebögen. Auch die intensive Benutzungsbetreuung ermöglichte einen engen Kontakt. Die Nutzung blieb in dieser Phase fast ausschließlich auf die universitäre Lehre beschränkt, CommSy entwickelte sich für diesen Kontext zu einem erprobten Werkzeug. Gegen Ende des Forschungsprojektes wurde die Entwicklung zur Sicherung der Nachhaltigkeit in ein Open-Source-Projekt überführt. Durch neue Forschungsprojekte, persönliche Initiativen und die Notwendigkeit, die Bereitstellung finanzieren zu können, erweiterte sich der Nutzungskreis auf neue Kontexte.

Zunächst wurde CommSy vermehrt im *Schulunterricht* eingesetzt. Im Rahmen eines halbjährigen Pilotprojekts mit mehreren LehrerInnen wurden Nutzungsszenarien und Anforderungen erarbeitet sowie das System in einer Reihe schulischer Veranstaltungen erprobt. Hierbei zeigte sich, dass die Nutzung gerade für jüngere SchülerInnen bislang unbekannte Probleme barg. Begrifflichkeiten und Handhabungskonzepte erwiesen sich als unverständlich und mussten angepasst werden. Zudem wurde teilweise der Ruf nach stärker lehrerzentrierten Funktionalitäten laut, die der bisherigen Designphilosophie – der Unterstützung eigenverantwortlichen, selbstorganisierten Lernens – entgegenstanden.

Im Rahmen eines weiteren Forschungsprojektes ergab sich mit der Erprobung und Weiterentwicklung von CommSy in *virtuellen Netzwerken* von Freelancern, die sich für die flexible Abwicklung von Projekten, Austausch und Weiterbildung zusammenschließen, ein weiterer Anwendungsbereich. Auch in diesem Kontext entstanden eine Reihe spezieller Anforderun-

gen, z.B. bezüglich der Projektkoordination, die über die Groupware-Basisfunktionalitäten hinausgingen.

Der parallele Einsatz in den verschiedenen Nutzungskontexten und die daraus resultierenden Gestaltungsanforderungen stellten den Softwareentwicklungsprozess auf eine harte Probe. Auf der einen Seite gab es eine grundsätzliche Übereinstimmung zwischen den verschiedenen Kontexten bezüglich grundlegender Werte des Softwareeinsatzes und der Nutzung konkreter Funktionalität, sodass eine gemeinsame Entwicklung für die unterschiedlichen Kontexte sinnvoll erschien. Andererseits unterschieden sich die jeweiligen Arbeitsabläufe und Ziele des Softwareeinsatzes erheblich. Einzelne EntwicklerInnen waren ganz oder schwerpunktmäßig in bestimmten Kontexten involviert und vertraten engagiert deren Standpunkte. Komplexe Konfigurationsmechanismen zur kontextspezifischen Anpassung der Software erwiesen sich nicht als tragfähig, da dies die Wartbarkeit der Software stark verschlechterte und zudem die Konsistenz und damit die Benutzbarkeit des Systems spürbar zu beeinträchtigen begann. Daher wurde nach Methoden gesucht, die ein Auseinanderbrechen der Entwicklung verhindern und die Gemeinsamkeiten der verschiedenen Kontexte betonen konnten.

3 Die Kommentierte Sammlung von Fallbeispielen

Ein erster Schritt, um seitens der EntwicklerInnen als auch der NutzerInnen ein Bewusstsein für die unterschiedlichen Anforderungen und Bedürfnisse zu bilden, waren Workshops, die gezielt Personen aus den verschiedenen Kontexten zusammenbrachten und einen Austausch von Nutzungserfahrungen sowie eine gemeinsame Diskussion von Gestaltungsoptionen ermöglichten. Im Gegensatz zu klassischen PD-Techniken wie Future-Workshops (vgl. Greenbaum & Kyng 1991) stand aber nicht die Analyse und Reorganisation von Arbeitsabläufen eines einzelnen Kontexts im Vordergrund, sondern der Erfahrungsaustausch über die Nutzung konkreter Funktionalität und die Reflexion über verbindende Werte der Softwarenutzung – mit dem Ziel, eine kontextübergreifende Entwicklungsgemeinschaft aufzubauen.

Die CCS-Methode adressiert den von NutzerInnen geäußerten Bedarf nach einem kontextübergreifenden, dauerhaften und damit von veranstalteten Workshops unabhängigen Austausch. Mit ihr werden einerseits Nutzungsbeispiele gesammelt, die sowohl den EntwicklerInnen als auch den NutzerInnen ein kontextübergreifendes Bild von Anforderungen vermittelt; andererseits werden die Beispiele von EntwicklerInnen kommentiert, um Design-Entscheidungen transparent zu machen. Es entsteht eine schriftliche Dokumentation, die gleichermaßen für die EntwicklerInnen wie für AnwenderInnen aus den unterschiedlichen Kontexten Gültigkeit hat.

3.1 Der Erstellungsprozess der Fallbeispiele

Ein wesentliches Ziel der Fallbeispiele ist es, authentische Praktiken in verschiedenen Nutzungskontexten zu beschreiben. Im Sinne von PD sind die NutzerInnen als *die* ExpertInnen in ihrer Anwendungsdomaine zu verstehen (vgl. Greenbaum & Kyng 1991); die Fallbeispiele sollten daher von NutzerInnen beschrieben und geschrieben werden.

Die Beschreibung eines Fallbeispiels stellt jedoch hohe Ansprüche an die NutzerInnen: sie müssen eine – häufig ungewohnte – Beobachterrolle auf die eigenen Tätigkeiten einnehmen und ihre Praxis reflektieren. Aus der PD-Literatur ist bekannt, dass dies (z.B. in Future Workshops) unter Moderation erfahrener EntwicklerInnen oftmals zu einem größeren Verständnis über den Arbeitsprozess auf beiden Seiten führt (vgl. Kensing & Madsen 1991). Bei der Erstellung der Fallbeispielsammlung ist eine direkte Moderation allerdings nicht erwünscht. Die entstehenden Texte sind „ungefiltert", da sie ohne den Zwang zu einer gemeinsamen Sprache, den ein Workshop erzeugt, entstanden sind. Die Verwendung der Sprache der jeweiligen Anwendungswelt kann ein großer Vorteil sein: Die Breite der Anwendungskontexte wird unmittelbar sichtbar und der Transfer zwischen NutzerInnen läuft ohne Mittlerfunktion der Entwicklersprache ab. Hier zeigt sich auch der Unterschied zu anderen Szenariotechniken: weder sollen die NutzerInnen ihre Arbeitspraxis exklusiv für die EntwicklerInnen beschreiben (vgl. Jacobsen et al. 1992), noch wird gemeinsam eine Systemnutzungsvision erstellt (vgl. Kensing & Madsen 1991), noch werden die Beschreibungen mehrerer NutzerInnen konsolidiert und so ein abstrakteres Modell der Nutzung gewonnen (vgl. Carroll et al. 1998). Vielmehr dient das einzelne Fallbeispiel im Zusammenspiel der kommentierten Sammlung der Fallbeispiele einer Arbeits- und Prozessreflexion für NutzerInnen wie EntwicklerInnen über die Grenzen einzelner Kontexte hinaus (vgl. 4).

Da die Erstellung eines Fallbeispiels einen relativ hohen Aufwand erfordert, müssen zunächst NutzerInnen identifiziert werden, die bereit sind, sich auf diese Weise in die Entwicklung einzubringen. Zunächst wurden NutzerInnen, die ohnehin schon in engem Kontakt mit den EntwicklerInnen standen, persönlich angesprochen; weitere AutorInnen wurden durch allgemeine Aufrufe angesprochen. Vor allem die Reaktion auf persönliche Ansprache war sehr positiv und eine ausreichende Menge von AutorInnen schnell identifiziert, von denen alle auch einen vollständigen Beitrag erstellt haben. In der gerade aufgelegten zweiten Fassung gelingt dies auch in vorwiegend nichtuniversitären Kontexten.

Um die Fallbeispiele besser lesbar zu gestalten und die Einordnung und den Vergleich zu vereinfachen, haben wir mit der Einladung, ein Fallbeispiel zu schreiben, eine Dokumentvorlage mitgeschickt, die eine sehr freie Gliederung vorgab. Wir haben uns gegen eine stärkere inhaltliche Führung, wie sie etwa in einem Leitfadeninterview eingesetzt wird (vgl. Flick 1999), aus der Überzeugung entschieden, dass die NutzerInnen Gegenstand und Schwerpunkt ihrer Beispielbeschreibung frei wählen können sollten, um für den jeweiligen Kontext wichtige Gesichtspunkte zu betonen. Zwei universitätsintern entstandene und eng betreute Beispiele sehr unterschiedlicher Beschreibungen wurden mit der Einladung verschickt, um das Format und unser Erkenntnisinteresse auch inhaltlich zu verdeutlichen. Der Austausch der Dokumente erfolgte ausschließlich über Email, und die redaktionelle Arbeit und die Zusammenführung der Beiträge wurde von EntwicklerInnen übernommen.

Ein Fallbeispiel beginnt mit einer vorstrukturierten Kontextbeschreibung, die zentrale Themen umfasst: zum einen eine Beschreibung, wie die Arbeit mit der Software organisiert wurde, welche gruppeninternen Regeln festgelegt und wie Prozesse vorbereitet und mit anderen Hilfsmitteln unterstützt wurden; zum anderen, wie die konkrete Nutzung der einzelnen Funktionen des Systems aussah – darunter fällt die Beschreibung sinnvoll „umgenutzter" Funktionalität ebenso wie die konkrete Schilderung, welche Funktionen von wem und wofür genutzt wurden (vgl. Abb. 1).

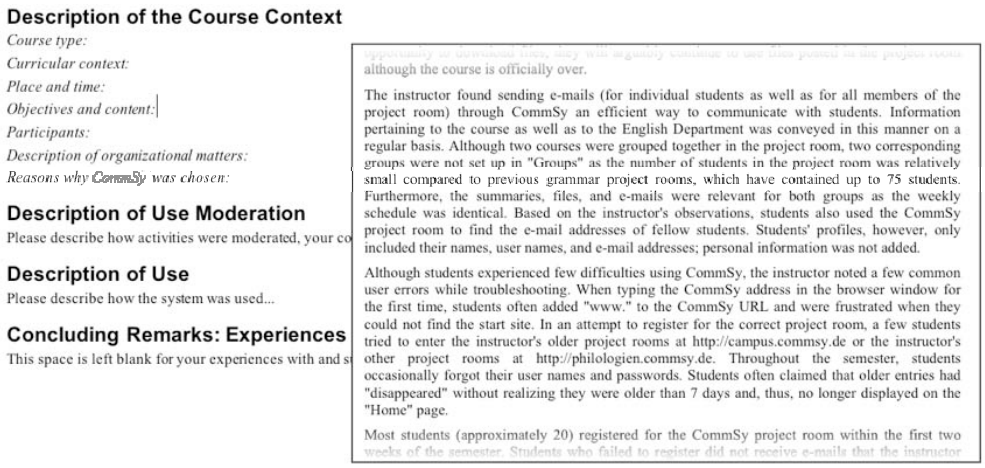

Abbildung 1: Informell gestaltete Vorlage und ein den Einsatz in der universitären Lehre beschreibendes Fragment.

3.2 Der Aufbau der kommentierten Sammlung

Schon eine einfache Sammlung von Fallbeispielen kann den Reflexionsprozess der AutorInnen, den Austausch von Nutzungserfahrungen und die Gegenüberstellung der verschiedenen Nutzungskontexte unterstützen – und dies im Gegensatz zu Workshops in einer dauerhaften Form, die auch weniger involvierten NutzerInnen zugänglich ist und den AutorInnen als Bezugspunkt in ihrer Funktion als MultiplikatorInnen dienen kann. In dieser Rohform fehlen aber noch wichtige Aspekte, die eine dauerhafte und über den Kreis der AutorInnen und EntwicklerInnen hinausgehende Nutzung ermöglichen: Zunächst kann eine so ausführliche Dokumentation auf manche NutzerInnen abschreckend wirken – das Auffinden der für die Lesenden relevanten Beispiele muss vereinfacht werden. Zweitens ist im Gegensatz zu der Diskussion auf einem Workshop nicht offensichtlich, welche Softwareversion und damit welche Funktionalität Grundlage für die beschriebene Nutzung ist; dies muss als Bezugspunkt ergänzt werden. Schließlich bleiben die von den AutorInnen angerissenen offenen Fragen unbeantwortet und die „Umnutzung“ von Funktionalität kann die von den EntwicklerInnen damit beabsichtigte Nutzung verdecken; dies ist dann ungünstig, wenn die Umnutzung als normal empfunden und nicht explizit beschrieben wird. Daher sollten die EntwicklerInnen wiederum die Fallbeispiele reflektieren. Der Sammlung von Fallbeispielen werden daher drei weitere Elemente als „Kommentar“ seitens der EntwicklerInnen beigefügt (vgl. auch Tabelle 1):

Der Zugriffsschlüssel beschreibt die unterschiedlichen Kontexte, zu denen Erfahrungsberichte vorliegen, sowie die Motivation für die Nutzung des Softwaresystems. Die Fallbeispiele werden hinsichtlich der Erwähnung bestimmter Funktionalität oder bestimmter Nutzungssituationen kategorisiert. Damit bekommen die LeserInnen einen Überblick über die Einsatzkontexte und können selektiv die Fallbeispiele lesen, die für sie besonders interessant sind.

Die Systembeschreibung: Die Systembeschreibung dient als Bezugspunkt für die Sammlung der Fallbeispiele. Hier werden die Designprinzipien und damit die gemeinsamen kontextübergreifenden Werte kommuniziert, die hinter der Nutzung stehen und in den Fallbeispielen nur implizit mitschwingen. Aber auch die Entwicklung der Software wird beschrieben anhand von Funktionalitäten, die im Vergleich zur vorherigen Version hinzugekommen sind.

Die Designreflexion: In dem begründenden Abschluss der Sammlung nehmen die EntwicklerInnen Stellung zu genannten Problemen und Weiterentwicklungswünschen. Sie führen eine Wertediskussion anhand der Gestaltung verschiedener in den Fallbeispielen erwähnter Features. Außerdem erläutern sie den Umgang mit beschriebenen Wünschen zur zukünftigen Entwicklung.

Tabelle 1: Systematische Darstellung der Struktur der kommentierten Sammlung von Fallbeispielen

Von der Dokumentation angesprochene Ebenen	Zugriffs-schlüssel	System-beschreibung	Fallbeispiele	Design-Reflektion
Funktionalität		●	●	●
Einsatz / Vorgehen	●		●	
Ziel / Wert	●	○	○	●

Damit ergänzt der Kommentar der EntwicklerInnen die gesammelten Fallbeispiele nicht nur um eine einfache und vereinheitlichte Zugriffsstruktur, sondern abstrahiert auch von den geschilderten konkreten Arbeitsabläufen und identifiziert Ziele und Werte, die über Kontextgrenzen hinaus Gültigkeit haben.

4 Nutzen von CCS als PD-Methode

Mit der CCS-Methode wird ein Bereich von PD adressiert, der bisher wenig Beachtung findet: die kontextübergreifende Entwicklung. Bei dieser Form der Softwareentwicklung verfügen die NutzerInnen nicht mehr über ein gemeinsames Repertoire an Arbeitsabläufen und Zielen des Softwareeinsatzes. Mit Hilfe der CCS-Methode wird eine gemeinsame Basis geschaffen – und zwar nicht auf der Ebene der Arbeitsabläufe, sondern zum einen auf der konkreten Ebene der Systemnutzung und zum anderen auf der abstrakten Ebene gemeinsamer Werte, die die verschiedenen Kontexte verbindet. Dies geschieht, indem zunächst zwischen den NutzerInnen ein gemeinsamer Erfahrungshorizont und eine Sensibilität für die unterschiedlichen Sichten auf den Einsatzzweck der Software geschaffen werden, um dann über Möglichkeiten der Softwareaneignung zu reflektieren. Die gemeinsame Auseinandersetzung über die Fallbeispiele erzeugt eine verbindende Klammer und fördert die Bildung einer kontextübergreifenden Interessensgemeinschaft. Im Vergleich zu klassischen PD-Methoden verschiebt sich dabei der Fokus von der Betrachtung konkreter organisatorischer Abläufe in einem Kontext zur Diskussion von Werten und Zielen bei der Softwarenutzung

(Abb. 2). Mit der Bildung einer kontextübergreifenden Interessensgemeinschaft auf der Basis gemeinsamer Werte und Ziele der Softwarenutzung gelingt es, das Einsatzfeld der Software weiterhin klar zu begrenzen – nicht auf spezielle Arbeitsabläufe, sondern auf gemeinsame Werte der Softwarenutzung. Für die Entwicklung von CommSy bedeutet dies z.B., dass sich die Entwicklung nicht mehr auf universitäre Lehre beschränkt, wohl aber auf den Einsatz des Systems in kooperativen Arbeits- oder Lernsituationen kleinerer Gruppen, die vertrauensvoll zusammenarbeiten und das System als Ergänzung zur Präsenzarbeit verwenden. Damit kann trotz einer Diversifizierung des Nutzungskontextes die Entwicklung zusätzlicher Funktionalität eingedämmt werden, sodass keine „eierlegende Wollmilchsau" als Konsequenz der Erweiterung von konkreten Nutzungskontexten entsteht.

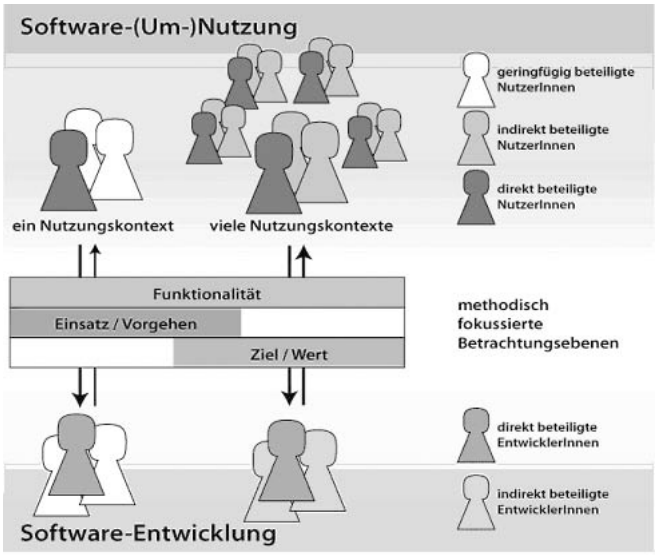

Abbildung 2: Unterstützung der Softwareaneignung für viele Nutzungskontexte mit der CCS-Methode.

Unsere Erfahrung mit dem Einsatz von CCS zeigt, dass – anders als bei NutzerInnen-Workshops – durch die Fallbeispiele tatsächlich eine breitere Masse an NutzerInnen erreicht und somit zumindest indirekt an der Designdiskussion beteiligt werden kann. So nutzten sowohl die AutorInnen der Fallbeispiele sowie auch hieran gänzlich unbeteiligte NutzerInnen die Sammlung, um anderen Anregungen für ihre Systemnutzung zu geben oder die Funktionsweise und Designphilosophie der Software zu verdeutlichen. Das Interesse an der Sammlung war so groß, dass AnwenderInnen aus dem Kontext virtueller Organisationen und Netzwerke selbst die Initiative zur Herausgabe einer weiteren, aktualisierten Auflage der Sammlung ergriffen. Erneut war die Bereitschaft der NutzerInnen groß, sich mit eigenen Beiträgen an dieser zweiten Sammlung zu beteiligen.

Auch die EntwicklerInnen profitieren von dem ganzheitlichen Bild der Softwarenutzung, das sie durch die Fallbeispiele erhalten. Häufig fehlte es seitens der EntwicklerInnen, die vornehmlich mit NutzerInnen aus einem Kontext in Verbindung stehen, an Verständnis für die

Anforderungen anderer Nutzungsgruppen. Die lebendigen und anekdotischen Schilderungen aus der Sammlung von Fallbeispielen ermöglichen ihnen, sich in die unbekannten Nutzungssituationen hineinzuversetzen. Das gemeinsame Kommentieren der Fallbeispiele und damit die Reflexion der getroffenen Design-Entscheidungen helfen dem Entwicklungsteam, seine Produktvision zu schärfen.

Mit der kontextübergreifenden Softwareentwicklung verschieben sich die Ziele partizipativer Softwareentwicklung von der Umgestaltung von Arbeitsabläufen zu einer Suche nach gemeinsamen Werten und Interessen. Auf einer konkreten Ebene müssen die begrenzten Ressourcen und gegensätzlichen Anforderungen aufgearbeitet und nach gangbaren Kompromissen gesucht werden. Die Frage nach der optimalen (Um-)Nutzung der Software bekommt eine wichtige Bedeutung: Was kann mit einer vorhandenen Software bereits unterstützt werden? Auf einer abstrakten Ebene können diese Kompromisse mit den identifizierten Werten abgeglichen werden. Die Auseinandersetzung mit der Gültigkeit der Werte in den unterschiedlichen Kontexten erlaubt eine abstrakte Diskussion über den Nutzen konkreter Funktionalität. Dieser Abstraktionsgrad entsteht durch die Gegenüberstellung der Nutzung desselben Systems in verschiedenen Kontexten, die in keinem unmittelbaren Zusammenhang stehen – statt über gegensätzliche Anforderungen und mögliche Weiterentwicklungen konkreter Funktionalität zu diskutieren, werden Gemeinsamkeiten sichtbar. Sowohl die NutzerInnen als auch die EntwicklerInnen werden gefordert, mit größerem Abstand die eigene Systemvorstellung zu reflektieren und gemeinsame Interessensschwerpunkte verschiedener Kontexte aufzudecken.

Die CCS-Methode ist nur dann sinnvoll einsetzbar, wenn es genug NutzerInnen in unterschiedlichen Kontexten gibt, die bereit und fähig sind, selbstständig ihre Nutzung zu reflektieren. Auch die notwendige Vorlaufzeit ist mit mehreren Wochen größer als bei einem Workshop. Deshalb ist ein Einsatz nur in größeren Projekten sinnvoll, in denen eine gewisse Funktionskonstanz erreicht wurde – größere Versionssprünge können alte Fallbeispiele obsolet machen. Sind diese Rahmenbedingungen gegeben, verspricht die CCS-Methode vorteilhafte Effekte für EntwicklerInnen und NutzerInnen in dreierlei Hinsicht:

- Die Akzeptanz der NutzerInnen für Gestaltungsentscheidungen kann steigen, wenn sie mehr über deren Gründe erfahren. Umgekehrt können die EntwicklerInnen die Übereinstimmung ihrer Designphilosophie mit den von den NutzerInnen formulierten Werten überprüfen und so Design-Entscheidungen legitimieren oder revidieren.
- Durch die gemeinsame Reflexion nimmt die Befähigung der NutzerInnen zu, die Software zur Unterstützung ihrer Aufgaben anzupassen; es steigt aber auch die Tiefe des Verständnisses der einzelnen Kontexte seitens der EntwicklerInnen.
- NutzerInnen können mit dem differenzierteren Verständnis der Nutzungsvision die Software eigenständiger in ihre Arbeitsabläufe integrieren. Für die EntwicklerInnen bietet sich schließlich die Möglichkeit, eine konsistente kontextübergreifende Systemvision zu erstellen.

Literaturverzeichnis

Carroll, J. M.; Rosson, M. B.; Koenemann, J. (1998): Requirements Development in Scenario-Based Design. IEEE Transactions on Software Engineering, Vol. 24, No. 12, December 1998.

Czyzewski, P.; Johnson, J.; Roberts, E. (1990): Introduction – Purpose of PDC ’90. Proc PDC 1990, CPSR, i-ii.

Flick, U. (1999): Qualitative Forschung: Theorien, Methoden, Anwendung in Psychologie und Sozialwissenschaften. Rowohlt-Taschenbuch-Verlag, 4. Auflage, Hamburg 1999.

Floyd, C. (1994): Software-Engineering – und dann?, Informatik Spektrum, Band 17, Heft 1, Springer-Verlag, Berlin, Heidelberg, New York, Tokio, 1994, S. 29-37.

Floyd, C.; Reisin, F.-M.; Schmidt, G. (1989): STEPS to software development with users. In: Proc. ESEC 1989, Springer, S. 48-64.

Floyd, C.; Züllighoven, H. (2002): Softwaretechnik, In: Rechenberg, P.; Pomberger, G. (Hrsg.): Informatik-Handbuch. 3. akt. und erw. Auflage, Hanser Verlag, München, Wien, S. 763-790.

Greenbaum, J.; Kyng, M. (1991): Design at Work: Cooperative Design of Computers Systems, Lawrence Erlbaum Ass, New Jersey.

Greenbaum, J.; Snelling, L.; Jolly, C.; On’, J. (1994): The limits of PD? Contingent jobs and work reorganization. In: Proc. PDC 1994, CPSR, S. 173-174.

Grudin, J. (1993): Obstacles to participatory design in large product development organizations. In: Schuler, D.; Namioka, A. (Hrsg.), Participatory Design: Principles and Practices, LEA, S. 99-119.

Lam, W; Loomes, M; Shankararaman, V (1999): Managing Requirements Change: A Set of Good Practices, In: Proceedings of the 3rd European Conference on Software Maintenance and Reengineering, S. 85-97.

Jacobson, I.; Christerson, M.; Jonsson, P.; Övergaard, G. (1992): Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, Reading, Massachusetts, 1992.

Kensing, F.; Madsen, K. H. (1991): Generating visions: Future workshops and metaphorical design. In: J. Greenbaum & M. Kyng (Hrsg.), Design at work: Cooperative design of computer systems. Hillsdale NJ US: Erlbaum.

Pipek, V. (2005): From tailoring to appropriation support: Negotiating groupware usage. PhD Thesis, Faculty of Science, Department of Information Processing Science (ACTA UNIVERSITATIS OULUENSIS A 430), University of Oulu, Oulu, Finland.

Törpel, B. (2001): Groupwareentwicklung in einem Dienstleistungsnetzwerk und die Tradition der Beteiligungsorientierten Systementwicklung. In: Matuschek, I., A. Henninger and F. Kleemann (Hrsg.): Neue Medien im Arbeitsalltag. Empirische Befunde, Gestaltungskonzepte, Theoretische Perspektiven. Opladen: Westdeutscher Verlag.

Wehner, J. (2001). Projektnetzwerke – Neue Unternehmensstrukturen und neue Qualifizierungen. In: Rohde, M., Rittenbruch, M., Wulf, V. (Hrsg.): Auf dem Weg zur virtuellen Organisation. Heidelberg: Physica-Verlag, S. 33-53.

Wulf, V.; Rohde, M. (1995): Towards an Integrated Organization and Technology Development. In: Proc of the Symposium on Designing Interactive Systems, New York, S. 55-64.