Ein patternorientierter Ansatz für die endbenutzerzentrierte Entwicklung kooperativer Systeme

Till Schümmer

Lehrgebiet für kooperative Systeme
Fernuniversität in Hagen, Universitätsstraße 1, 58084 Hagen
till.schuemmer@fernuni-hagen.de

Abstract: In diesem Beitrag wird ein Entwurfsprozess für kooperative Systeme vorgestellt, der sich primär durch die Nutzung von Entwurfsmustern für kooperative Systeme auszeichnet. Es wird gezeigt, wie ein iterativer Prozess Benutzerpartizipation fördern kann und wie die Benutzer in die Lage versetzt werden, aktive Designentscheidungen für kooperative Systeme zu treffen.

1 Einleitung

Die Entwicklung von kooperativen Systemen ist eine anspruchsvolle Aufgabe. Einer der Gründe hierfür ist, dass sowohl Anwender als auch Entwickler häufig keinen Überblick über die Möglichkeiten kooperativer Anwendungen haben.

Ferner geht es bei der Entwicklung kooperativer Systeme sowohl darum, die Interaktion eines Benutzers mit dem Computersystem zu strukturieren, als auch darum, den Gruppenprozess zu definieren und zu unterstützen. Die Mensch-Maschine-Mensch-Interaktion (Human-Computer-Human-Interaction – HCHI) rückt hierbei in den Vordergrund, wofür es aktuell jedoch nur unzureichende Unterstützungsformen im Sinne von Prozessen oder praktischen Richtlinien gibt.

Diese Faktoren machen die Definition von Anforderungen schwierig, was ein großes Problem für Entwickler und Anwender bei der Erstellung kooperativer Systeme ist. In der hier zusammengefassten Dissertation [Sch05] wurde deshalb ein Vorgehensmodell und eine Sammlung von Entwurfsmustern (Patterns) vorgestellt, durch die Benutzer und Entwickler bei der Gestaltung kooperativer Systeme unterstützt werden. In diesem Artikel sollen die Haupterkenntnisse daraus zusammengefasst werden.

Der verbleibende Teil dieses Artikels gliedert sich wie folgt: Zunächst werden einige theoretische Aspekte des Designbegriffs im Kontext von kooperativen Systemen diskutiert und die daraus resultierenden Anforderungen hergeleitet. Danach wird der Stand der Forschung dargestellt. Abschnitt 4 stellt den Oregon Software Development Process vor. Über Erfahrungen mit diesem Prozess wird exemplarisch in Abschnitt 5 berichtet. Eine Zusammenfassung und ein Ausblick auf zukünftige Forschungsarbeiten schließen diesen Artikel ab.

2 Eine Annäherung an die Theorie des Designs

Eines der Hauptprobleme beim Entwurf kooperativer Systeme ist die Definition von Anforderungen an das zu entwickelnde kooperative System (als Werkzeug für die kooperierenden Menschen). Dabei spielen Fragestellungen eine Rolle, die für den generellen Gebrauch und Entwurf von Werkzeugen relevant sind. Der Philosoph Martin Heidegger [Hei27] macht deutlich, dass Werkzeuge oft nicht explizit wahrgenommen werden. Er bezeichnet sie deshalb als *Zuhandenes*, also als einen Gegenstand, der Teil des agierenden Individuums geworden ist. Das Zuhandene ist außerhalb der Wahrnehmung, solange es in den Kontext passt. Passen die vom Designer des Werkzeugs angenommenen Rahmenbedingungen nicht mehr optimal auf die aktuelle Situation, so kommt es zu einem *Bruch*. In der Situation eines Bruchs nimmt der Nutzer das Werkzeug wahr und erkennt die Anforderungen. Ohne die direkte Erfahrung des Bruchs wird die Beschreibung von Anforderungen schwierig und es bleibt ein großer Unsicherheitsfaktor, ob sich das Werkzeug in zukünftigen Situationen bewähren wird (vgl. auch [Sta93]).

Der Architekt Alexander [Ale64] nimmt an, dass sich die so wahrgenommenen Anforderungen als Kräfte beschreiben lassen. Anforderungen stehen im Wechselspiel mit dem Kontext und beeinflussen sich gegenseitig. Eine lokale Betrachtung der Anforderungen ist zunächst möglich (Modularisierung), bei der Zusammenführung verschiedener Anforderungen (Komposition) müssen diese als Kräfte im potentiell unendlichen Kontext jedoch zusammen betrachtet werden. Sobald ein Problem lokal durch Einfügen neuer Artefakte in das Kräftefeld gelöst wurde, muss das Zusammenspiel des Gesamtsystems betrachtet werden. Rittel und Webber [RW73] bezeichnen solche Probleme als *Wicked Problems*. Das Zusammenspiel des Gesamtsystems muss folglich stets neu überprüft werden. Schön [Sch83] prägte hierzu den Begriff der *Reflection in Action*. Benutzer eines Werkzeuges sollen, sobald sie einen Bruch erfahren, zur Reflexion über das Werkzeug in ihrem aktuellen Kontext motiviert werden. Dies resultiert zum einen in einem Verständnis der widersprüchlichen Anforderungen und zum anderen in einem angepassten Design des genutzten Werkzeugs, dem sogenannten *Tailoring*.

Damit der Benutzer die Kontrolle über seine Umgebung behalten kann, ist es erforderlich, dass er mit kognitiven Werkzeugen ausgerüstet wird, die es ihm erlauben, gut durchdachte Entwurfsentscheidungen zu treffen. Entwurfsmuster (Pattern) [Ale79] sind ein solches Werkzeug. Sie bilden den Benutzer, indem sie wiederkehrende Probleme und deren Lösungen so darstellen, dass sowohl die Lösung als auch die Begründung für die Lösung einsichtig wird. Dabei beschreiben sie die ein Designproblem, einen Designkontext und die in dem Problem im Widerspruch stehenden Anforderungen. Die Lösung beschreibt, wie diese Anforderungen ausgeglichen werden können, um das System in ein Gleichgewicht zu bringen und wie sich der Kontext durch Implementierung des Patterns verändern wird. Im Gegensatz zu fertigen Komponenten, die oft intern nicht mehr verändert werden können, geht ein Entwurfsmuster davon aus, dass zwei Implementierungen des Musters niemals gleich sein werden, da sich der Kontext stets unterscheidet.

Alexander zeigt weiter, dass Patterns im Idealfall die Struktur der Lösung verstärken. Kompatible Anforderungen werden hervorgehoben während widersprüchliche Anforderungen in ein Gleichgewicht gebracht werden. In einem Entwicklungsprozess sollte man,

so Alexander, eine Sequenz von Patterns finden, die diese Eigenschaft nicht nur für ein Pattern sondern für eine ganze Folge von Patterns berücksichtigt. Jedes Pattern in dieser Sequenz stellt dann eine *strukturerhaltende Transformation* dar [Ale03]. Entwickelt man das System komplett durch strukturerhaltende Transformationen, so ergibt sich hierbei eine holistische Sicht auf die Systemgestaltung, da jede Transformation das Gesamtsystem berücksichtigen muss.

Doch wie kommt man zu einer Folge von strukturerhaltenden Transformationen? Im Sinne von Entwurfsprozessen laufen alle diese Erkenntnisse auf einen partizipativen Entwurfsprozess hinaus, wie er in der Skandinavischen Schule weit verbreitet ist [BB95]. Ziel dieser Prozesse ist es, den Benutzer so gut wie möglich in allen Teilen der Gestaltung von Systemen mit einzubeziehen. Die Benutzer sollen die Muster auswählen und im Kontext des Gesamtsystems bewerten. In der Architektur findet sich hierzu das Oregon Experiment [ASA$^+$80], in dem exemplarisch die Beteiligung der Nutzer bei der Gestaltung der Gebäude der Universität von Oregon organisiert wurde.

2.1 Anforderungen an einen Prozess zur Entwicklung von Groupware

Vor dem Hintergrund der diskutierten Interpretationen des Gestaltungsprozesses ergeben sich fünf Anforderungskomplexe an einen Entwurfsprozess für kooperative Systeme:

Ganzheitliche Sicht auf die Anforderungen: Anforderungen für kooperative Systeme können nicht losgelöst vom soziotechnischen Kontext der Nutzer betrachtet werden.

Evolutionäres Design: Kooperative Systeme sollen verstärkt in iterativen Prozessen entworfen werden. Gruppenteilnehmer müssen zur Reflexion über die Gruppeninteraktion und gegebenenfalls zu deren Anpassung ermuntert werden.

Modularität: Der Entwurfsprozess sollte dazu beitragen, den Fokus auf ein Problem zur Zeit zu legen, ohne jedoch die Auswirkungen (im Sinne von Kräften) auf das Gesamtsystem zu ignorieren. Wichtig ist hierbei die Situiertheit der Anforderungsanalyse in einem Bruch.

Nutzerorientiertes Design: Die Benutzer sollen in die Lage versetzt werden, das kooperative System selbst an die aktuellen Bedürfnisse der Gruppe anzupassen. Dies bedingt einen Wissenstransfer von Designwissen zu Benutzern und Entwicklern.

Austausch von Wissen: Erprobte Praktiken zur Anpassung von kooperativen Systemen sollen in der Gemeinschaft der Nutzer kommuniziert und bewertet werden, um die Qualität der Anpassungsfertigkeiten der Benutzer zu verbessern.

3 Stand der Forschung

Zum Stand der Forschung sind zwei verschiedene Klassen von Entwicklungsmethodiken zu betrachten. Erstens Vorgehensmodelle und Hilfsmittel für die Entwicklung beliebiger

Anwendungen und zweitens speziell für kooperative Systeme entwickelte Vorgehensmodelle.

Im Bereich der Software-Entwicklungsprozesse stehen die folgenden Modelle stellvertretend für die am weitesten verbreiteten Entwicklungsmethodiken:

- sequentielle Modelle (z.B. das Wasserfall-Modell [Roy70]),
- iterative Modelle (z.B. der Unified Process [JBR99]), in denen die Entwicklung des Systems in mehreren Iterationsschritten durchlaufen wird, bei denen jede Iteration eine zusätzliche Annäherung an die Ziele der Benutzer bringt,
- agile Methoden (z.B. eXtreme Programming [Bec99]), die auf extrem kurze Iterationen und eine vermehrte Kommunikation zwischen Benutzern und Entwicklern ausgerichtet sind und
- partizipative Vorgehensmodelle [MK93], bei denen die Interaktion mit dem Benutzer und seine Einbindung in den Prozess im Vordergrund stehen.

Daneben sind Entwurfsmuster in vielen Teilen der Softwareentwicklung inzwischen Stand der Technik. Dabei liegt der Fokus der Forschung primär auf technikzentrierten Entwurfsmustern für objektorientiertes Software-Design. Für die Mensch-Maschine-Interaktion sind erste Pattern-Sammlungen entstanden (bspw. [Gra02]).

Alle genannten Modelle erfüllen jeweils nur Teilaspekte der Anforderungen. So setzen sequenzielle, iterative und agile Modelle einen Fokus auf die Unterstützung der technischen Aufgaben im Lebenszyklus eines Entwicklungsprojektes. Partizipative Vorgehensmodelle und Entwurfsmuster legen hingegen besonderen Wert auf die Einbeziehung der Benutzer.

Entwurfsmuster unterstützen den Benutzer bei der Erschließung von Lösungskonzepten und spielen daher eine wichtige didaktische Rolle. Diese auch für kooperative Systeme zu nutzen ist eines der Ziele des in Abschnitt 4 vorgestellten Lösungskonzeptes.

Speziell für das Anwendungsfeld von kooperativen Systemen finden sich nur wenige Vorgehensmodelle in der Literatur. Zu erwähnen sind hier vor allem das *Organizational and Technical Development* Modell (OTD) [WKS+99] und das *SER* Modell (seeding, evolutionary growth, and reseeding) [FGM+01]. Beide Modelle legen einen Schwerpunkt auf die Einbeziehung der Benutzer bei der Anpassung des Systems während dessen Einsatzes mit dem Ziel, dass die Benutzer die nötigen Anpassungen selbst vornehmen (*Tailoring*). Der Erfahrungsaustausch bezüglich der Anpassungen wird in den Modellen unterschiedlich detailliert beschrieben. Rittenbruch u.a. [RMW+02] berichten über den Einsatz und die Anpassung der eXtreme Programming-Methodologie für die Entwicklung von kooperativen Systemen. Darin haben die Autoren vor allem die Rolle des Kunden neu überdacht und an die Interaktion mit Benutzergruppen angepasst. Ein besonderer Fokus liegt auf der Koordination der verschiedenen Benutzeranforderungen.

Auf dem Feld der Entwurfsmuster für kooperative Systeme gibt es neben den in der Dissertation [Sch05] veröffentlichten Mustern keine Sammlungen, die einen umfassenden Blick auf kooperative Systeme werfen. Insbesondere gibt es noch keinen Ansatz, der Muster für kooperative Systeme mit einem Entwicklungsprozess in Verbindung bringt.

4 Lösungsansatz

Wie die Übersicht über existierende Ansätze gezeigt hat, existiert bisher noch kein Prozessmodell, welches alle in den Anforderungen genannten Aspekte berücksichtigt. Aus diesem Grund wurde der Oregon Software Development Process entworfen. In diesem Abschnitt sollen zunächst der Prozess vorgestellt und danach Groupware Patterns als zentrales didaktisches Mittel diskutiert werden.

4.1 Der Oregon Software Development Process

Der Oregon Software Development Process (OSDP) propagiert ein iteratives Vorgehen in drei verschiedenen Arten von Iterationen (vgl. Abb. 1). Jede dieser Iterationen kann mehrfach durchlaufen werden. Verschiedene Beteiligte können parallel verschiedene Arten von Iterationen ausführen.

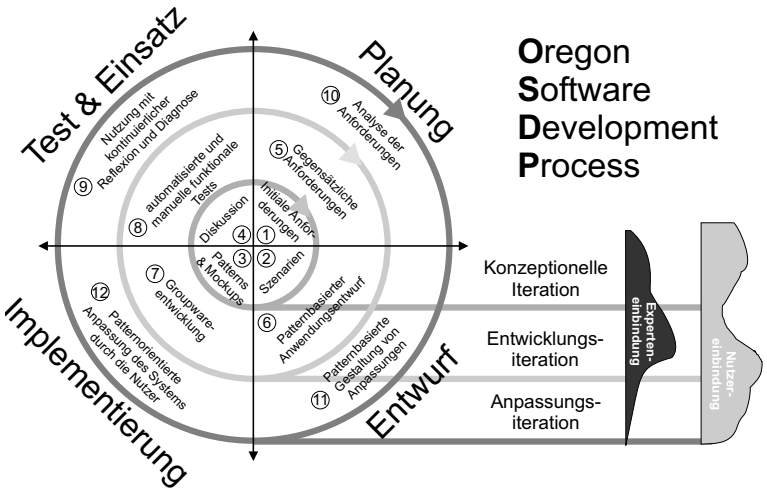

Abbildung 1: Oregon Software Development Process (OSDP) [SS04, Sch05]

Die innere *Konzeptionelle Iteration* verfolgt zwei primäre Ziele: (a) Die Benutzer und Entwickler sollen ein erstes Verständnis über das zu entwickelnde kooperative System und den dazugehörigen Kooperationskontext erhalten. (b) Es werden *Scenario Interest Groups* (SIGs) gebildet, in denen sich Benutzer zusammenfinden, die ein gemeinsames Interesse an den gleichen Kooperationsszenarien haben.

Zu Beginn der Iteration bildet sich die Projektgruppe und Entwickler und Endanwender kommen zusammen, um sich gemeinsam ein klares Bild über die Anforderungen zu verschaffen (1). Die Anforderungen werden hauptsächlich durch die Art von Kollaboration, die unterstützt werden soll, bestimmt. Die Endanwender erstellen danach zusammen

mit den Entwicklern in unterschiedlichen SIGs Anwendungsszenarien (2). Anhand dieser Anwendungsszenarien werden eine Reihe von *anwendungsorientierten Patterns* aus einer größeren *Groupware Pattern Language* ausgewählt. Diese Patterns dienen den Entwicklern als Startpunkt, um erste *Mockups*, also rudimentäre Prototypen der Benutzungsschnittstelle, für das zu entwickelnde System zu erstellen (3). Entwickler und Endanwender diskutieren gemeinsam die Prototypen, die in den einzelnen SIGs entwickelt worden sind (4). Sie entscheiden, welche der Szenarien für die Entwicklung die höchste Relevanz haben. Das Ergebnis der *Konzeptionellen Iteration* ist somit eine Liste von geordneten Anwendungsszenarien und eine Menge von SIGs, die im weiteren Verlauf des Projektes eng miteinander kooperieren werden.

Um schnelle Korrekturen von Fehlern und eine flexible Anpassung an sich ändernde Anforderungen zu unterstützen, setzt OSDP auf kurze *Entwicklungsiterationen*. Jede Entwicklungsiteration besteht aus

(5) der Feststellung von miteinander in Konflikt stehenden Anforderungen,

(6) einem auf *implementierungsnahen Patterns* basierenden objektorientierten Entwurf,

(7) der eigentlichen Entwicklung der Groupware-Anwendung, bei der durchaus auf objektorientierte Groupware-Plattformen zurückgegriffen werden kann und

(8) funktionalen Tests der Anwendung.

Die Interaktion zwischen Entwickler und Endanwender im Kontext der SIGs wird durch *Task Cards* strukturiert, welche die Aufgaben für die Entwickler unter Nutzung von Patterns beschreiben (6). Der initiale Katalog von Patterns kann in dieser Iteration jederzeit erweitert werden, sofern dies neue Anforderungen erfordern (5). Da mehrere SIGs an mehreren Szenarien gleichzeitig arbeiten, übernimmt ein Entwickler die Rolle eines *Gardener*, wie sie von Rittenbruch u.a. [RMW+02] vorgeschlagen wird. Der *Gardener* hat die Aufgabe, alle *Task Cards* zu sammeln und diese anhand ihrer Relevanz, wie sie durch die SIGs bestimmt wird, zu ordnen. Bei den Iterationen geben die Entwickler immer wieder Zwischenversionen der Anwendung frei und diskutieren diese mit den Endanwendern.

In der *Tailoring Iteration* benutzen Anwender die entwickelte Groupware-Anwendung. Sobald sie einen Bruch erleben, sind sie aufgefordert über ihre Kooperation zu reflektieren und die Notwendigkeit einer Anpassung zu bedenken (9). Diese Reflexion ist der Ausgangspunkt für das Tailoring der Anwendung. In jeder Iteration analysieren die Endanwender ihre Anforderungen (10). Danach wählen die Benutzer relevante anwendungsorientierte Patterns aus und beschreiben damit eine Anpassung des kooperativen Systems (11). Die in den Patterns beschriebenen Lösungen können durch die Anwender häufig selbst umgesetzt werden, um die Anwendung anzupassen. Anwender konfigurieren dazu die Anwendung durch die Komposition einzelner funktionaler Komponenten, um die kollaborierende Gruppe besser zu unterstützen (12). Wenn es den Endanwendern nicht möglich ist die Anwendung eigenständig anzupassen, beschreiben sie das Problem auf einer *Task Card* und geben diese an den *Gardener* weiter. Der *Gardener* sortiert daraufhin die Sammlung aller *Task Cards* neu, um alle Entwickler und Endanwender auf einem aktuellen Stand über den Entwicklungsplan zu halten.

Während der gesamten *Tailoring Iteration* übernimmt ein Gruppenmitglied die Aufgabe des *Pattern Scouts*. Der *Pattern Scout* hat die Aufgabe Lösungen zu identifizieren, die im Rahmen des Projektes für gute Ergebnisse sorgten. Dazu beobachtet der *Pattern Scout* das Verhalten der Anwender im System, diskutiert gefundene Lösungen mit den Anwendern und dokumentiert diese als Patterns. Die so gefundenen Patterns werden der *Pattern Language* hinzugefügt und stehen somit bei neuen Projekten zur Verfügung.

4.2 Eine Patternsprache für Groupware

Ein essentieller Bestandteil von OSDP ist die Verwendung von Patterns. Patterns werden in OSDP eingesetzt, um vorhandenes Entwurfswissen zu erfassen und zu kommunizieren. OSDP ordnet Patterns auf einer Skala zwischen *anwendungsnahen* und *implementierungsnahen* Pattens an. *Anwendungsnahe Patterns* sind für Endanwender gedacht und beziehen sich auf das Verhalten einer Anwendung, wie sie durch den Endanwender wahrgenommen wird. *Implementierungsnahe Patterns* zielen auf den Entwickler und beschäftigen sich bspw. mit Klassenstrukturen oder Netzwerkkommunikation.

Wichtig ist, dass alle Patterns in einer sowohl für den Anwender als auch für den Entwickler verständlichen Sprache verfasst sind. Eine eher prosaische Beschreibung mit vielen Elementen, die zum besseren Erinnern der Patterns beitragen, hat sich in der erstellten Groupware-Pattern-Sammlung bewährt. Aktuell ist ein holistischer Katalog mit etwa 200 solcher Patterns in Arbeit [SL07]. Eine erste Version dieses Katalogs konnte im Rahmen der Dissertation [Sch05] mit Nutzern erprobt werden. Die prosaische Form der Muster impliziert einen relativ großen Umfang, weshalb aus Platzgründen in diesem Artikel keine vollständigen Muster abgedruckt werden können. Da die Muster jedoch eine hohe Struktur aufweisen, kann man einen Eindruck vom Inhalt der Muster bereits durch die Lektüre der Problemstellung und des Lösungsabschnitts erhalten. Dies soll an zwei Beispielen demonstriert werden: Das Room Pattern als ein anwendungsnahes Pattern und das Lovely Bags Pattern als ein implementierungsnahes Pattern.

Room. *Problem:* Während des Lebenszyklus einer verteilten Gruppe müssen die Teilnehmer oft technische Aspekte der Kooperation lösen: Kommunikationskanäle müssen initiiert und gemeinsam genutztes Material muss zur Verfügung gestellt werden. Die Ergebnisse der Kooperation müssen für spätere Zecke archiviert werden. Bei diesen Aktionen können Fehler auftreten. Sie benötigen technische Detailkenntnisse bei den Benutzern, die diese jedoch oft nicht haben.

Lösung: Modelliere einen virtuellen Kooperationsplatz als einen Raum. Dieser kann Dokumente und Teilnehmer verwalten. Stelle sicher, dass Benutzer miteinander mittels eines automatisch etablierten Kommunikationskanals kommunizieren können, sofern sie zur gleichen Zeit im gleichen Raum sind. Trage Sorge dafür, dass alle Benutzer auf die Dokumente eines Raumes in den aktuellen Versionen zugreifen können und mache diese Dokumente persistent.

Lovely Bags. *Problem:* Schreibzugriffe auf gemeinsam genutzte Container-Objekte ändern den Inhalt dieser Objekte durch Hinzufügung und Entfernung von Elementen. Die meisten dieser Zugriffe haben ein schlechtes Nebenläufigkeitsverhalten. Das lässt synchrone Kooperation mit diesen Objekten oft unmöglich erscheinen.

Lösung: Modelliere das Container-Objekt als Bag, sofern ein hoher Grad an Nebenläufigkeit benötigt wird. Sofern eine Ordnung der Elemente nötig ist, versuche diese anhand eines Ordnungskriteriums innerhalb der einzelnen Objekte lokal wiederherzustellen, aber nutze für das gemeinsam genutzte Datenobjekt nach wie vor ein Bag.

5 Erfahrungen

Die Kombination von OSDP und Entwurfsmustern für kooperative Systeme wurde im Rahmen des Projektes CURE evaluiert, indem eine kooperative Lernplattform für die FernUniversität in Hagen entwickelt wurde, die inzwischen im täglichen Einsatz von mehr als 1500 registrierten Studenten genutzt wird.

In konzeptionellen Iterationen bildeten sich SIGs zu Themenbereichen wie *Virtuelle Seminare* oder *Kooperative Gruppenübungen*. Beteiligt waren an diesen SIGs Lehrende und Studierende, die ihre Wünsche und Vorstellungen für das verteilte Lernen einbrachten. Am Ende der konzeptionellen Iterationen standen eine Reihe von Szenarien und Designstudien. Ein Beispiel für ein Szenario ist die Vorbereitung für ein virtuelles Seminar. Die Teilnehmer identifizierten die nötigen Schritte im Kooperationsprozess und attributierten diese Schritte mit passenden Patterns (bspw. dem Room Pattern im Fall des virtuellen Seminars).

In den Entwicklungsiterationen interagierten die Mitglieder der SIGs mit den Entwicklern und identifizierten benötigte Funktionen, die von Benutzern und Entwicklern gemeinsam auf Patterns abgebildet wurden. Die Funktionen wurden auf Task-Cards vermerkt und geordnet. Der Gardener hat sich als wichtige Rolle herausgestellt, da er die Sequenzen von Task-Cards der verschiedenen SIGs miteinander verwob und so einen Plan für die Entwicklung des Gesamtsystems pflegte. In der Implementierung stellte sich die Sequenz der eingesetzten Patterns als sinnvolle strukturerhaltende Transformation dar. Der holistische Ansatz sorgte dafür, dass das System koherent von verschiedenen SIGs genutzt werden konnte.

Die Benutzer begannen sehr schnell, kreativ mit dem System umzugehen. Anpassungen wurden sowohl durch die Entwickler (in neuen Entwicklungsiterationen) als auch durch die Benutzer in *Anpassungsiterationen* vorgenommen. Vor allem Änderungen am Gruppenprozess konnten beobachtet werden. Mechanismen zur Anpassung der Inhalte des Lernsystems unter Nutzung von Inhaltsvorlagen konnten ebenfalls beobachtet werden und halfen, die Gruppeninteraktion besser zu unterstützen. Erste neue Entwurfsmuster wie zum Beispiel ein Muster zur Gestaltung von Literaturrecherchen entstanden und wurden zwischen den Benutzern ausgetauscht.

6 Zusammenfassung und Ausblick

In diesem Artikel wurde die Gestaltung kooperativer Systeme diskutiert. Im Vergleich zum Stand der Forschung sind folgende Fortschritte hervorzuheben:

- der *Oregon Software Development Process*, in dem vor allem Benutzerbeteiligung unterstützt wird,
- ein *Format für Entwurfsmuster*, das vorrangig für die Nutzung durch Endbenutzer ausgelegt ist,
- eine Sammlung von *Entwurfsmustern für die Entwicklung kooperativer Systeme* mit vollständiger Beschreibung ausgewählter Entwurfsmuster, sowie
- eine *Fallstudie* über den Einsatz der Entwurfsmuster an einem konkreten Beispiel, der kooperativen Lernplattform CURE.

Es konnte dargestellt werden, wie Benutzer zusammen mit Entwicklern in der Lage sind, kohärente Systeme unter der Nutzung von strukturerhaltenden Transformationen zu erstellen. Aktuell wird der Katalog der dabei benutzten Entwurfsmuster erweitert, um Benutzer in allen Aspekten kooperativer Systeme zu unterstützen. Inwieweit strukturerhaltende Transformationen so auch in anderen Kontexten hilfreich sein können, ist eine relevante Frage für zukünftige Forschungsarbeiten.

Literatur

[Ale64] Christopher Alexander. *Notes on the Synthesis of Form*. Harvard University Press, Cambridge, Massachusetts, 7 (2002). Auflage, 1964.

[Ale79] Christopher Alexander. *The timeless way of building*. Oxford University Press, New York, 1979.

[Ale03] Christopher Alexander. *The phenomenom of life*. Center for Environmental Studies, Berkeley, California, USA, 2003.

[ASA+80] Christopher Alexander, Murray Silverstein, Shlomo Angel, Sara Ishikawa und Denny Abrams. *The Oregon Experiment*. Oxford University Press, New York, 1980.

[BB95] Gro Bjerknes und Tone Bratteteig. User participation and democracy: a discussion of Scandinavian research on systems development. *Scand. J. Inf. Syst.*, 7(1):73–98, 1995.

[Bec99] K. Beck. *eXtreme Programming Explained*. Addison Wesley, 1999.

[FGM+01] G. Fischer, J. Grudin, R. McCall, J. Ostwald, D. Redmiles, B. Reeves und F. Shipman. Seeding, Evolutionary Growth and Reseeding: The Incremental Development of Collaborative Design Environments. In G. Olson, T. Malone und J. Smith, Hrsg., *Coordination Theory and Collaboration Technology*, Seiten 447–472. Lawrence Erlbaum Associates, 2001.

[Gra02] Ian Graham. *A Pattern Language for Web Usability*. Addison-Wesley, 2002.

[Hei27] Martin Heidegger. *Sein und Zeit*. Niemeyer, Tübingen, 17 (1993). Auflage, 1927.

[JBR99] Ivar Jacobson, Grady Booch und James Rumbaugh. *Unified Software Development Process*. Addison-Wesley, 1999.

[MK93] Michael J. Muller und Sarah Kuhn. Participatory design. *Communications of the ACM*, 36(6):24–28, 1993.

[RMW+02] Markus Rittenbruch, Gregor McEwan, Nigel Ward, Tim Mansfield und Dominik Bartenstein. Extreme participation - moving extreme programming towards participatory design. In *Proceedings of PDC 2002*, Malmo, Sweden, June 2002.

[Roy70] Winston Royce. Managing the Development of Large Software System. In *Proceedings of IEEE WESCON*, August 1970.

[RW73] Horst W. J. Rittel und Melvin M. Webber. Dilemmas in a general theory of planning. *Policy Sciences*, 4:155–169, 1973.

[Sch83] Donald A. Schön. *The Reflective Practictioner: How Professionals Think in Action*. Basic Books, New York, 1983.

[Sch05] Till Schümmer. *A Pattern Approach for End-User Centered Groupware Development*. Schriften zu Kooperations- und Mediensystemen - Band 3. JOSEF EUL VERLAG GmbH, Lohmar - Köln, August 2005.

[SL07] Till Schümmer und Stephan Lukosch. *Patterns for Computer-Mediated Interaction*. John Wiley and Sons Ltd., 2007. to appear.

[SS04] Till Schümmer und Robert Slagter. The Oregon Software Development Process. In *Proceedings of XP2004*, 2004.

[Sta93] Gerry Stahl. *Interpretation in Design: The Problem of Tacit and Explicit Understanding in Computer Support of Cooperative Design*. Dissertation, Univ. of Colorado, 1993.

[WKS+99] Volker Wulf, Matthias Krings, Oliver Stiemerling, Giulio Iacucci, Paul Fuchs-Fronhofen, Joachim Hinrichs, Martin Maidhof, Bernhard Nett und Ralph Peters. Improving Inter-Organizational Processes with Integrated Organization and Technology Development. *Journal of Universal Computer Science*, 5(6):339 – 365, 1999.

Till Schümmer, geboren 1973, studierte von 1994 bis 1999 Informatik an der TU Darmstadt. Im Rahmen dieses Studiums kristallisierte sich bereits das Feld der kooperativen Systeme als einer seiner Interessensschwerpunkte heraus. In den Jahren 1999 bis 2000 war er Stipendiat am DFG Graduiertenkolleg “Infrastrukturen für elektronische Märkte” an der TU Darmstadt. Die Schwerpunkte lagen hier bei der Unterstützung verteilter Kooperation im B2C e-Commerce. 2001 wechselte er an die Fernuniversität in Hagen, wo er seitdem neue kooperative Lernsysteme gestaltet und im Kontext der Fernuniversität im Feld evaluiert. Entwurfsmuster haben sich für ihn dabei als wichtiges Mittel zur Einbindung von Endbenutzern erwiesen. Till Schümmer ist Autor zahlreicher Entwurfsmuster für kooperative Systeme und Organisator verschiedener Workshops zu diesem Thema. 2005 fasste er die Ergebnisse dieser musterorientierten Sicht in seiner Promotion zusammen.