EUD-SMES
End User Development in Small and Medium Enterprise Software Systems

Roadmap Entwicklung hochanpassbarer Systeme

gefördert vom Bundesministerium für Bildung und Forschung

EUD-SMES
End User Development in Small and Medium Enterprise Software Systems

Roadmap Entwicklung hochanpassbarer Systeme

Externer Meilenstein des EUDISMES-Projekt

Veröffentlichungsdatum: 14.12.2006

Das Bundesministerium für Bildung und Forschung (BMBF) unterstützt das Projekt EUDISMES mit der Forschungsoffensive "Software Engineering 2006".
(Förderkennzeichen: 01 IS E03 A)

BUHL
STRÄHLE+HESS
Viβer
STERNIAKOB
SAP
Universität Siegen
Wirtschaftsinformatik
Wirtschaftsinformatik und Neue Medien

Inhaltsverzeichnis

1 Roadmap „Entwicklung hochanpassbarer Systeme“

1.1 Einführung

Ziel der Roadmap ist die Vorgabe einer Projektmethodik, an der sich weitere Planungs- und Umsetzungsarbeiten hinsichtlich prototypischer Implementierungen orientieren. Die in der Roadmap aufgezeigten Grundlagen (EUD-Pattern) und Anwendungsfelder (Demonstratoren) bilden dabei die konzeptionelle Basis. Durch die vorangegangene Empirie konnten Übersichts- und Problemaspekte in den Unternehmen identifiziert und kategorisiert werden. Im Whitepaper „Endbenutzerorientierte Anpassbarkeit für ERP Systeme in KMU“ sind die wichtigsten – für KMU generalisierbare – Prozessszenarien aufgezeigt. Innerhalb dieser Szenarien lassen sich einzelne Prozessschritte identifizieren, die sich insbesondere durch EUD-Maßnahmen effizienter umsetzen und gestalten lassen. Die Roadmap gibt deshalb nicht nur die technologischen Grundlagen (Pattern und Demonstratoren) vor, sondern zeigt auch spezifische Prozessschritte auf, die im weiteren Projektverlauf mit den identifizierten EUD-Maßnahmen optimiert werden sollen.

Die Roadmap ist ein dynamisches Dokument in dem Sinne, dass sie im weiteren Projektverlauf an neue Erkenntnisse angepasst und verfeinert werden kann. Die Roadmap spiegelt dabei den aktuellen Stand der EUD-Grundlagen (EUD-Technologien) wieder und stellt Ansatzpunkte zur Optimierung in den Prozessketten dar.

1.2 Inhalt

Bevor man Prozesse durch EUD-Maßnahmen optimieren kann, muss man einerseits die technischen und konzeptionellen Möglichkeiten verstehen, um Systeme „hochanpassbar“ zu gestalten und andererseits die Prozesse verstehen, auf die man mit den Maßnahmen Einfluss nehmen möchte. Als eine der frühen Projektarbeiten wurde deshalb auch die Technologielandkarte entwickelt, die eine Vielzahl von EUD-Methoden identifiziert und nach Kategorien geordnet in Patternform darstellt. Neben diesen EUD-Pattern sind auch eine Vielzahl von Demonstratoren beschrieben, die eine Umsetzung des jeweiligen Pattern darstellen.

Auf Grundlage der Technologielandkarte werden die im Whitepaper identifizierten Prozessschritte diskutiert und deren Potential für Aneignungs- und Unterstützungsmaßnahmen hervorgehoben. Dabei sind Chancen, Grenzen und Einschränkungen wichtige Rahmenbedingungen. Auf Grundlage der Unterstützungspotentiale in den definierten Prozessketten, können prototypische Entwicklungen gezielt und definiert für allgemein relevante Anwendungsfälle (Prozessszenarien) konzipiert und entwickelt werden.

Nachdem das Kapitel 2 eine wissenschaftliche Einführung zu dem Thema EUD gibt und in Kapitel 3 kurz auf das Thema Usability eingegangen wird, wird in Kapitel 4 die Methodik beschrieben, die hinter der Erstellung der Technologielandkarte liegt. Bevor in Kapitel 6 die einzelnen identifizierten EUD-Pattern beschrieben werden, ist in Kapitel 5 die Gliederung der Patterns in definierte Kategorien dargestellt. Kapitel 8 beschreibt die identifizierten Demonstratoren. In Kapitel 9 werden die im Whitepaper identifizierten EUD-Ansatzpunkte zur Prozessunterstützung erörtert. Abschließend wird im letzten Kapitel noch einmal zusammengefasst, was allgemein für die Entwicklung von Roadmaps zur Realisierung der beschriebenen Maßnahmen gesagt werden kann.

2 EUD-Forschungsschwerpunkte

2.1 Einführung in die EUD-Forschung

„End User Development is a set of activities or techniques that allow people, who are non-professional developers, at some point to create or modify a software artifact“(Paternò, Klann et al. 2003)

Im Rahmen des *Network of Excellent* zum Thema End User Development (EUD-Net) der Europäischen Union wurde ein interdisziplinäres Dach geschaffen, um eine integrierte Sichtweise auf das Thema zu entwickeln. Die gemeinsame Leitfrage ist, wie man Endbenutzer dabei unterstützen kann, ihre Softwareartefakte besser in ihre Arbeitspraxis einzubetten und an ihre Bedürfnisse anzupassen. Einen guten Überblick über die verschiedenen Vertreter der EUD-Forscher und deren Arbeiten findet man in dem Buch *End User Development* von (Liebermann et al. 2006).

Versucht man das Forschungsfeld zu strukturieren, bietet es sich in einer ersten Annäherung an, die Arbeiten aufgrund ihres gewählten Forschungsfokus zu klassifizieren. Insbesondere unterscheiden sich die Arbeiten dahingehend, wie stark sie den sozio-technischen Anwendungskontext mit in ihre Überlegungen einbeziehen.

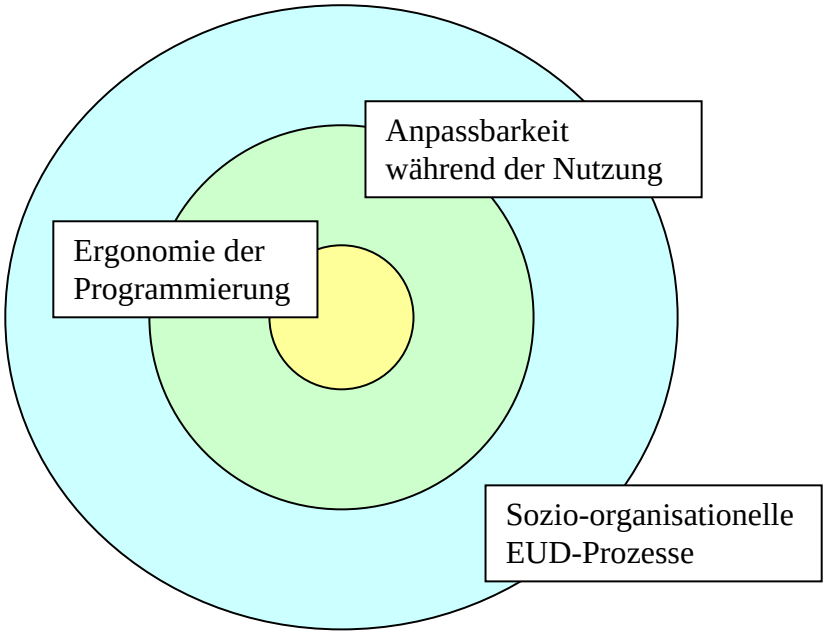

Abbildung 1 Fokusse der EUD-Forschung

2.2 Ergonomie der Programmierung

Die Forschung zur Ergonomie der Programmierung gehört zu den ältesten Bereichen der EUD-Forschung. Die Anfänge finden sich in den 70er Jahren des letzten Jahrhunderts Jahrhunderts (z.B. Miller 1974). Das vorherrschende Methoden- und Forschungsparadigma ist stark von einer quantitativ orientierten Psychologie bestimmt. International wird deshalb auch häufig von *Psychology of Programming* gesprochen. Insbesondere in den 80er Jahren wurden viele kontrollierte Laborstudien unternommen, um die Ergonomie von neu aufkommenden Programmiersprachen und speziellen Programmkonstrukten (z.B. Programmblöcke mit ‘BEGIN’-’END’) durch so genannte Novice Programmers zu bestimmen (meist Kinder oder

Undergraduated Computer Science Students). Empirisch gestützte ergonomische Untersuchungen zu EUD werden aber auch heutzutage noch durchgeführt. Dies liegt zum einen am Aufkommen neuer Programmierkonzepte, wie z.B. der visuellen Programmierung (Green and Perte 1996), zum anderen an neu aufkommenden Forschungsfragen. So versucht z.B. das *Natural Programming* (Pane, Ratanamahatana et al. 2001; Pane and Myers 2006) nicht die Ergonomie einzelner Programmkonstrukte und Notationssysteme zu bewerten, sondern aus, von Computerlaien angefertigten, natürlich-sprachlichen Algorithmusbeschreibungen ergonomische Aspekte für Programmsysteme abzuleiten. Ein anderes Beispiel sind die ergonomischen Studien von Burnett zum *End User Software Engineering* (Burnett, Cook et al. 2004). Hier wird nicht nur der Task des Codings unter ergonomischen Aspekten studiert, sondern es werden andere, während der Softwareentwicklung anfallende Aktivitäten, wie das Testen oder das Debuggen (Rothermel, Cook et al. 2000) untersucht.

Daneben lassen sich diesem Forschungszweig auch die Arbeiten zu den methodischen Grundlagen der Untersuchung ergonomischer Aspekte zuordnen. Insbesondere das Modell der *Cognitive dimensions of notations*(Green 1989) stellt einen Kriterienkatalog zur Bestimmung der Gebrauchstauglichkeit von Programmiersystemen zur Verfügung, mit dem die Besonderheiten von Programmiersystemen besser erfasst werden können, als dies z.B. bei der DIN EN ISO 9241 Teil 11 der Fall ist.

Einen Überblick über den aktuellen Stand der *Psychology of Programming* von EUD findet man bei (Blackwell 2006). Eine Aufarbeitung der software-ergonomischen Arbeiten zur Programmierung und die Zusammenstellung der darin vorgeschlagenen Empfehlungen finden sich bei (Pane and Myers 1996).

2.3 Anpassbarkeit während der Nutzung

„If the modifications that are being made are to the subject of matter of the tool then we think of it as use; if the modifications are the tool itself, then it is tailoring“(Henderson and Kyng 1991, S. 224)

Die Arbeiten aus diesem Zweig der EUD-Forschung studieren EUD aus der Perspektive des Nutzungskontextes. Die meisten Arbeiten in diesen Bereich beziehen sich dabei auf die oben zitierte Definition von (Henderson and Kyng 1991). Sie, als auch (Oberquelle 1994) verstehen dabei Anpassung als eine auf das Werkzeug bezogene sekundäre Tätigkeit des Nutzers. Dagegen stellt das Verändern einer Applikation für dessen Entwickler seine primäre Arbeitsaufgabe dar und fällt deshalb nicht unter den Begriff des Anpassens. Aus dieser Perspektive ergeben sich entsprechende Grenzfälle in Anwendungsdomänen bei denen *ein bisschen* Programmieren Teil der alltäglichen Arbeitspraxis ausmacht. (Letondal 2006) hat diesen Grenzfall von EUD am Beispiel von Bioinformatikern untersucht.
Einen kritischen Punkt beim Anpassen einer Applikation stellt dabei der Übergang zwischen Nutzung und Anpassung dar. Dieser Übergang wird üblicherweise dadurch erschwert, dass heutige Anwendungssysteme auf technischer Ebene noch unzureichend auf diesen Übergang vorbereitet sind. So konstatieren (MacLean, Carter et al. 1990), als auch (Bentley and Dourish 1995), dass bei den meisten bestehenden Systemen ein *customization gulf* zwischen einer oberflächlichen und einer tief greifenden Anpassung besteht, die sich in der Aufteilung der Systemarchitektur in Interface und Funktionalität widerspiegelt. Darüber hinaus finden sich abrupte Sprünge in der Komplexität der Anpassungsmöglichkeiten, die es den Endnutzer erschweren, sich immer mächtigere Anpassungsmöglichkeiten im Laufe der Zeit anzueignen (Mørch 1997). Teilweise wird deshalb auch die Gestaltung eines *gentle slope of complexity*

als *das* Ziel der EUD-Forschung angesehen (Myers, Hudson et al. 2003; Fischer, Giaccardi et al. 2004; Pane and Myers 2006).

In der EUD Forschung wurden verschieden Ansätze entwickelt, um den *customization gulf* zu verringern. So versucht der Ansatz der *Direct Activation* (Wulf and Golombek 2001) die Integration in den Nutzungskontext zu verbessern; das Konzept der *Programming by Demonstration* (Liebermann 2001) knüpft bei der Programmierung des Systems an die Interaktionsformen der Nutzung an; das Konzept der *Self Disclosure Interfaces* (DiGiano 1996) koppelt graphische und Kommandozeilen orientierte Interfaces, um so den Übergang zum Skripting des Systems sanfter zu gestalten.

Daneben setzen sich eine Reihe von Arbeiten mit der Frage auseinander, wie die Anpassbarkeit der zugrunde liegende Systems erhöht werden kann (Malone, Lai et al. 1994; Stiemerling 2000; Teege 2000). Die Anpassbarkeit des System kann dabei nach (Stiemerling 2000) durch die drei folgenden Parameter charakterisiert werden: Die Repräsentation der Systemeigenschaften, die Funktionalität diese zu Ändern und die Verbindung zwischen der Repräsentation und den Systemeigenschaften.

2.4 Sozio-organisationale EUD-Prozesse

„The problems of software systems customization are not simply restricted to questions of how flexible software systems should be. Instead, those problems are both fundamental and endemic to the nature of computer-supported cooperative work. Customization is part of the fabric of collaborative activity.“(Dourish, Lamping et al. 1999)

Eine dritte Forschungsrichtung lässt sich in Arbeiten zu EUD erkennen, die versuchen EUD aus ihrem sozio-technischen Kontext heraus zu verstehen. Neben dieser inhaltlichen Verlagerung des Fokus findet sich auch auf der methodischen Ebene ein Unterschied zu den Forschungsarbeiten zur Ergonomie der Programmierung. In diesen Zweig der EUD-Forschung folgt man meist dem Paradigma qualitativer empirischer Forschung, wobei häufig auf ethnographische Methoden zurückgegriffen wird.

Verschiedene Studien haben dabei versucht, die Anpassungsgewohnheiten und die in der Praxis anzutreffenden Hindernisse und Anlässe zur Anpassung und Aneignung innerhalb von Organisationen empirisch zu bestimmen (Mackay 1991; Gantt and Nardi 1992; Wulf 1999). Hierbei hat sich gezeigt, dass Anpassungen zumeist in Netzwerken von Akteuren mit unterschiedlicher Qualifikation und Neigung eingebettet sind. Dabei spielen sowohl Gurus und lokale Entwickler als auch das Ausbilden einer *Tailoring Culture* eine entscheidende Rolle. Bezug nehmend auf die Klassifikation von Gruppenanpassbarkeit (Oberquelle 1994) haben (Kahler 2001; Pipek and Kahler 2005) untersucht, wie und durch welche technischen EUD-Maßnahmen sich die verschiedene Formen kooperativen Anpassens unterstützen lassen.

Verschiedene Autoren haben sich auch mit den Konsequenzen von EUD für den gesamten Softwareentwicklungsprozess auseinandergesetzt. So wird sowohl im erweiterten STEPS Modell (Wulf and Rohde 1995) als auch im SER-Modell (Fischer and Ostwald 2002) explizit berücksichtigt, dass Software im Nutzungskontext durch den Endbenutzer weiterentwickelt wird. Die Prozessmodelle greifen diesen Umstand auf und propagieren ein zyklisches Modell von Entwicklung auf der einen Seite und Nutzung/Anpassung auf der anderen Seite, wobei bei der (Weiter-)Entwicklung die in der Praxis vollzogenen Anpassungen berücksichtigt werden.

Ein erweitertes Verständnis von EUD findet man bei (Pipek 2005). Er legt dar, dass die EUD-Forschung zu kurz greift, wenn sie sich nur auf die Umgestaltung der Artefakte konzentriert. Stattdessen stellt die Anpassung der Systemeigenschaften aus Nutzersicht nur eine Form dar, mittels der das Artefakt kreativ in die eigene Arbeitspraxis integriert werden kann. Eine Veränderung des Artefakts aus der Nutzerperspektive kann sich aber auch in einer stabilen Änderung der Nutzung bzw. Umnutzung von Features des Artefakts ausdrücken. (Pipek 2005) spricht hierbei auch von Aneignungsprozessen, die über die Anpassung hinausgehen. Aus dieser Perspektive hat Pipek eine Reihe von Maßnahmen angeregt um solche Aneignungsprozesse zu unterstützen. (Draxler and Stevens 2006; Stevens and Draxler 2006) regen an, auch die Reflektionsprozesse über die Softwareartefakte und den Arbeitskontext als die notwendige Gestaltungsaktivität des *Problem Framing* in Sinne von (Schön 1983) zu betrachten und entsprechende Unterstützungshilfen für Endbenutzer zu entwickeln.

2.5 EUD als Melting pot von Forscher und Praktiker unterschiedlichster Herkunft

"*Opportunities to learn and to inform generate enthusiasm-which is needed to overcome inevitable obstacles. It is not always apparent why another person's perspective and priorities differ. It takes patience to understand conflicts in priorities and to find mutually advantageous modes of operation*"(Grudin 1994)

Zusammenfassend lässt sich also feststellen, dass das Forschungsfeld End User Development sich durch sehr unterschiedliche Perspektiven auf das Phänomen der Umgestaltung von Softwareartefakten durch den Nutzer bezieht. Es zeichnet sich jedoch genau dadurch aus, dass es sich nicht auf eine der Perspektiven einschränken lässt, sondern vielmehr einen multi-disziplinären *Melting pot* bzw. *Market place* darstellt, an dem Forscher und Praktiker mit unterschiedlichem Hintergrund und Begriffsgebäuden partizipieren.

Diese Vielfalt der Perspektiven innerhalb der EUD-Forschung bietet dabei die Chance von den Erkenntnissen anderer Disziplinen zu lernen und bestehende Ansätze gewinnbringend neu zu kombinieren. Jedoch merkt (Grudin 1994) zu Recht an, dass Multidiziplinarität auch eine hohe Anforderung des gezeigten Verstehens auf Seiten der beteiligten Akteure erfordert. Aus diesem Grunde besteht innerhalb des Forschungsfelds des EUD der Bedarf an einer *EUD-Landkarte*, die den Akteuren hilft die jeweiligen Forschungsergebnisse einordnen zu können.

Das Mittel „Technologielandkarte" hilft dabei, sich einen Überblick zu verschaffen und die EUD-Maßnahmen miteinander in Relation zu setzen. Für die praktische Anwendbarkeit bei der Gestaltung von Anwendungssystemen gilt es darüber hinaus den Problemkontext und das Lösungspotential der einzelnen EUD-Maßnahmen herauszuarbeiten. Im Software-Engineering hat sich dabei bewährt, das gesammelte Wissen als so genannte Pattern (Alexander 1977; Gamma, Helm et al. 1995) aufzubereiten, um so die Anwendbarkeit des Wissens zu erhöhen.

3 Usability

3.1 Einordnung

Usability lässt sich nicht als eine EUD-Maßnahme (vgl. EUD-Patterns in Kapitel 6) einordnen. Vielmehr bezeichnet das Thema Usability ein Querschnittsthema, das insbesondere auch im Zusammenhang mit EUD-Strategien eine wichtige Bedeutung besitzt. Usability – oder auch Gebrauchstauglichkeit – spielt an der Schnittstelle zwischen Nutzer und System eine entscheidende Rolle. Es ist nicht ausreichend, dass Systeme eine Vielzahl an Möglichkeiten zur Anpassung bieten. Vielmehr ist es wichtig, dass geeignete Anpassungsoptionen von den Nutzern auch intuitiv bedient und genutzt werden können. Es ist daher zwischen dem Konzept einer Unterstützungsmaßnahme (EUD-Pattern) und einer einfachen und intuitiv zu bedienenden Realisierung bzgl. Usability-Aspekten zu unterscheiden.

Das Thema Usability ist auch betriebswirtschaftlich von großer Bedeutung. Auch die beste und am Funktionsumfang gemessene vollständigste IT-Lösung ist fehl am Platze, wenn der Zugang zu benötigten Funktionen nur schwer oder umständlich möglich ist. Mitarbeiter müssen schnell mit einer Lösung „vertraut“ sein. Intuitive Bedienung ist Voraussetzung für schnelle Einarbeitung und den folgenden Produktivitäts-Output. Das Interface einer SW-Lösung sollte selbstbeschreibend sein in dem Sinne, dass die Nutzung von Hilfe- und Supportmaßnahmen auf ein niedriges Maß reduziert werden kann. Aus diesem Grund sind auch alle EUD-Maßnahmen unter Usability-Gesichtspunkten zu betrachten. Das EUDISMES-Projekt fokussiert sich dabei nicht nur auf konzeptionelle und technische EUD-Methoden, sondern berücksichtigt auch Aspekte der Bedienbarkeit an der Schnittstelle zum Nutzer.

Die im Projekt entwickelten EUD-Maßnahmen sollen an geeigneter Stelle auch unter Usability-Aspekten betrachtet werden. Aus diesem Grund gibt der folgende Abschnitt zunächst eine kurze Beschreibung zum Thema Usability, bevor in Abschnitt 3.3 Usability-Methoden beschrieben werden. Diese Methoden bilden zusammen mit den technischen EUD-Maßnahmen die Projekt-Expertise für folgende Entwicklungen und Evaluationen.

3.2 Bedeutung von Usability-Maßnahmen für das EUD in KMU

Im Kontext des Themas End-User Development sollten Usability-Maßnahmen zu einem kontinuierlichen Verifizierungsinstrument von EUD-Massnahmen entwickelt werden. Die besondere Problemlage in EUD-Ansätzen läuft auf eine Involvierung vieler unterschiedlicher Typen von Benutzerrollen und Benutzerexpertisen und möglicherweise auch vielen unterschiedlichen Benutzerschnittstellen (abhängig von der funktionalen Dekomposition der betrachteten Anwendungen) hinaus. Der Bereich Usability muss dabei zwei Aufgaben abdecken. Zum einen ist sicherzustellen, dass auch in dieser Heterogenität von Benutzern und Schnittstellen es immer noch möglich ist, effizient und effektiv die in Betracht gezogenen Infrastrukturveränderungen auch umzusetzen. Da nicht davon ausgegangen werden kann, alle Schattierungen von Benutzerkenntnissen abzutesten, muss hier eine sinnvolle Auswahl getroffen werden. Zum anderen kommt den Kriterien der Selbsterklärungsfähigkeit und Lernförderlichkeit eine besondere Rolle zu. Eine adäquate Überwachung dieser Interfacesaspekte garantiert, dass es Benutzern ermöglicht wird, sich sukzessive in die Konfiguration ihrer Infrastrukturen einzuarbeiten ohne über Gebühr von ihrem Arbeitskontext abgelenkt zu werden. Dies ist insbesondere auch hinsichtlich weicher, nicht-technischer Aspekte relevant: Es muss bei Endbenutzern der Eindruck entstehen, sich immer mit ein wenig mehr Arbeitsaufwand in der Konfiguration der Infrastrukturen ein wenig besser auszukennen. Nur wenn dies ein Eindruck ist, der sich bei der überwiegenden Anzahl der

Endbenutzer durchgesetzt hat, kann eine nachhaltige Kultur eines reflektiven Umgangs mit technischen Infrastrukturen am Arbeitsplatz entstehen.

Abhängig von den technischen Möglichkeiten der betrachteten Grundsysteme kann auch noch ein dritter Aspekt für Usabilityuntersuchungen hinzukommen. Wenn die möglichen Umkonfigurierungen von KMU-Infrastrukturen auch die Benutzbarkeit beeinflussen, kann es notwendig sein, auch Endbenutzern effiziente Möglichkeiten zu geben, die Usability der von ihnen modifizierten Benutzerschnittstellen zu untersuchen.

Für kleine KMU steigt wiederum die Wichtigkeit von Usability-Untersuchungen, da bei ihnen Szenarien, in denen ein Benutzer wirklich allein mit Rechner und Handbuch (d.h. ohne die Hilfe von Kollegen) Konfigurierungen vor sich nehmen muss, häufiger vorkommen dürften als bei den größeren KMU.

3.3 Beschreibung

Usability ist ein englisches Kunstwort, das sich aus den Begriffen „to use“ (etwas gebrauchen, verwenden) und „ability“ (Fähigkeit, Tauglichkeit) zusammensetzt. Übersetzungen, die in der aktuellen Literatur häufiger zu finden ist, lauten „Gebrauchstauglichkeit“ oder „Benutzerfreundlichkeit“. Der Begriff stammt aus der Forschung zur Mensch-Computer-Interaktion (human computer inteaction, HCI) und hat sich inzwischen auch in der deutschen Literatur durchgesetzt.

Gegenstand des Usability-Testing ist die Gebrauchstauglichkeit oder Benutzerfreundlichkeit von bestimmten Systemen. Der Gebrauchs- oder Bedienungsprozess eines mehr oder weniger technischen Systems durch die Benutzer wird dabei analysiert und auf Fehlerquellen hin überprüft. Eine gängige Definition stammt von der Internationalen Organisation für Standardisierung, die ein System als benutzerfreundlich kennzeichnet, sofern es die folgenden Kriterien erfüllt:

- Effektivität
- Effizienz
- Zufriedenheit

Zunächst stellt sich die Frage, wie die beiden erstgenannten Kriterien zu verstehen sind. Am besten lassen sie sich darüber beschreiben, wie sie meist operationalisiert werden:

- das Ausmaß der Zielerreichung (Effektivität) und
- das Verhältnis von Aufwand und Ausmaß der Zielerreichung (Effizienz).

Es lässt sich daher notwendigerweise ableiten, dass zur Beschreibung der Usability eines Systems

- eine Beschreibung der Ziele des Benutzungsprozesses,
- eine Beschreibung der (potentiellen) Benutzer,
- eine Beschreibung der zur Zielerreichung notwendigen Aktionen,
- eine Beschreibung der notwendigen Ausstattung (insbesondere Software und Hardware)

sowie

- eine Beschreibung der Umgebung oder des Kontextes, in dem die Benutzung stattfindet

wichtig sein werden.

Zusammenfassend lässt sich festhalten, dass Usability eine hypothetische Eigenschaft ist, die (technischen) Systemen zugeschrieben wird, wenn sie "benutzerfreundlich", "angenehm zu bedienen", "geeignet zur Erfüllung einer bestimmten Aufgabe" und dergleichen mehr sind. Gegenstand des Usability-Testing ist folglich die Überprüfung dieser Eigenschaften auf einer methodisch sicheren Basis; während lange Zeit die Qualität nach dem eher darwinistischen Prinzip „was sich auf dem Markt durchsetzt, ist gut!“ beurteilt wurde, sind heute methodisch saubere Testverfahren gefragt, die einem Hersteller konkrete Anhaltspunkte zur Verbesserung des entwickelten Systems bieten – noch bevor das Produkt auf den Markt kommt.

3.4 Methoden

Es existieren verschiedene Usability-Methoden. Die wichtigsten sind im Folgenden aufgeführt:

User Tracking:
- Beobachten des Verhaltens

Online Quick-Checks:
- Über Fragebögen

Remote Usability Testing:
- Fragebögen
- Verhalten

Usability-Testing im Lab:
- Interview
- Beobachtung
- Eye Tracking
- Verhalten

Fragebogen

Tiefencheck:
- Im Labor
- Tiefenpsychologisches Interview

Der Nutzer im realen Kontext:
- Interview
- Beobachtung
- Verhalten
- Kontextdaten

Die folgenden Abschnitte beschreiben beispielhaft wichtige Usability-Maßnahmen:

Usability Testing im Lab
Beim klassischen Usability Testing im Lab erhält man vergleichbare Ergebnisse unter kontrollierten Bedingungen. Interviews und Verhaltensbeobachtung sind möglich. Die direkte Interaktion zwischen Nutzer und Interviewer ermöglicht tiefer gehende Analysen.

Usability Testing Remote
Ausgewählte Nutzer führen zu Hause einen Remote Usability Test durch und bearbeiten Aufgaben und Fragebögen. Es sind systematische Experimente mit unterschiedlichen Design-Varianten möglich. Von den Remote Testern werden Clicks, Mausbewegungen, Scrollbewegungen, und Eingaben erfasst (jeweils mit Timeline).

Tiefencheck im Lab
Experten führen Tiefeninterviews mit Ihren Nutzern durch, um Nutzungshintergründe, Bedürfnisstrukturen und psychologische Motivationen aufzudecken.

Usability Testing mit Eye Tracking
Der Eye Tracker ist in den Bildschirm eingebaut. Das Remote Eye Tracking System registriert den Blickverlauf mit Hilfe von Infrarot-Technik und Mustererkennung.

Bei allen Usability-Maßnahmen können in einer zweiten Ordnungsstrafe noch Testszenarien unterschieden werden:

- Einen Usability **Walk-Through** zur schnellen Aufnahme von Schwachstellen, insbes. bei frühen Releases.
- Einen **Metrik-Test** zur objektiven Bewertung des Usability-Status Quo
- Einen **Konzept-Test** zum Überprüfen von neuen Usability-Konzepten

Walk-Through

Mit Hilfe dieses Verfahrens sollen insbesondere bei Beta-Release-Ständen oder auch noch früheren Releases Lücken festgestellt werden und durch den Walk-Through bewertet werden. Insbesondere steht die reine Beobachtung und Befragung des Probanden bei dieser Testform im Vordergrund. Es sollen anhand von gefundenen Usability-Schwachpunkten Lösungsansätze zur Verbesserung der Usability gefunden werden.

Metrik-Tests

Metrik-Tests dienen dazu eine Software und deren Usability objektiv zu bewerten. Die Usability-Metriken werden dabei anhand von normierten Use-Cases definiert. Diese Use-Cases sind ein Subset von allen möglichen Use-Cases, die ein Nutzer mit der Software ausführen kann. Das Subset wird dabei auf 10-15 wesentliche Use-Cases konzentriert. Die wesentlichen Use-Cases zielen dabei auf die Kernfunktionalität des Produktes ab. Folgende Kriterien dienen bei der Abarbeitung von Use-Cases zur Bewertung der Usability:

- hat der Nutzer den Use-Case erfolgreich abgearbeitet?
- falls eine erfolgreiche Abarbeitung erfolgte: in welcher Zeit wurde der Use-Case abgearbeitet?
- falls eine erfolgreiche Abarbeitung erfolgte: wie ist der Zeitvergleich mit einer optimalen (Best-Practice) Abarbeitung?
- Welche Hilfsmittel wurden benötigt um den Use-Case abarbeiten zu können (z.B. Bedienhilfe, Online-Hilfe, Moderator gefragt, etc.).

Mit Hilfe eines gewichteten Punktesystems wird anhand dieser Metriken eine Bewertung abgegeben, die einmalig kalibriert werden muss. Die Kalibrierung erfolgt durch einen initialen Test.

Konzept-Tests

Bei diesen Tests wird ein Prototyp der zu testenden SW gebaut. Neu überarbeitete Usability-Konzepte werden nun mit einem eingeschränkten Probandenkreis getestet. Hier kommt besonders die Interview-Technik mit Beobachtung zum Zuge. Metriken werden hier auch genutzt, wenn Use-Cases der in der Metrik genutzten Use-Cases optimiert werden müssen.

Grundsätzlich sieht der Testablauf bei allen Testtypen wie folgt aus:

- Es wird eine Entscheidung über das Testszenario getroffen (s.o.)
- Es wird festgelegt welche Use-Cases zu testen sind. Die Use-Cases sind bei einem Metrik-Test immer gleich, bzw. beinhalten ein festes Use-Case Set, das aus Gründen der Vergleichbarkeit immer enthalten ist.
- Die Use-Cases werden in einer Use-Case Dokumentation so beschrieben, wie sie sich aus Kundensicht darstellen. Soll zum Beispiel eine Konteneinrichtung getestet werden, erhält der Nutzer die Vorlage seiner Bank zur Anmeldung von Online-Banking mit genau den Daten, die der Nutzer bei seiner Bank erhält um Online-fähige Konten anzulegen.
- Die gesuchten Probandenprofile werden durch den Usability-Manager und den zuständigen Produkt-Manager festgelegt (z.B. Firmeninhaber mit max. 3 Beschäftigten mit Steuerberater und EÜR, wenige IT-Kenntnisse, etc.)
- Geeignete Probanden werden durch eine Ausschreibung gesucht.
- Die Probanden werden nach einem festgelegten Zeitplan nacheinander innerhalb von 2-3 Tagen durch den Test geführt. Pro Proband wird zwischen 1-1,5 Std. veranschlagt.
- Der Test wird durchgeführt und aufgezeichnet. Parallele Beobachtung ist optional möglich. Üblicherweise wird der Kunde während der Tests in Metrik-Tests nicht befragt, sonst schon.
- Testauswertung anhand der Aufzeichnungen
- Erstellung eines Usability-Reports
- Vorstellung des Reports
- Klassifikation der Usability-Probleme
- Erarbeitung von Lösungen zur Vermeidung der klassifizierten Probleme
- Umsetzung der Lösung
- Erneuter Test (Konzept-Test) zur Feststellung der Güte der Lösung
- Finale Umsetzung
- Erneuter optionaler Walk-Through
- Metrik-Test (kann auch mit dem Walk-Through zusammen durchgeführt werden).

3.5 Ansatzpotentiale für Verbesserungen

Die bzgl. dem Konzept der Servicekaskadierung (siehe „Whitepaper: Endbenutzerorientierte Anpassbarkeit für ERP-Systeme in KMU“) gefundenen Usability-Probleme wurden in einem internen Meilenstein dokumentiert und bewertet. Dabei wurden unter anderen zwei Problemklassen identifiziert, die für den domänenspezifischen Prozess der Servicekaskadierung besonders relevant sind. Im Folgenden sind zwei Problemklassen mit „Stammdaten“ und „Coach“ bezeichnet. Diese relevanten Kategorien sind exemplarisch kurz beschrieben.

Stammdaten

Sehr viele der identifizierten Probleme sind „kleine Hakeligkeiten“ bei der Eingabe und Pflege von Stammdaten, die zwar jeder für sich betrachtet nicht sehr gravierend sind, in der Summe aber doch ein massives Problem darstellen. Darunter fallen z.B. Probleme mit nicht eindeutigen Feld-Beschriftungen bei – eigentlich – so einfachen Bereichen wie einer Adress-Eingabe, oder auch das Prinzip von Ansichts- und Änderungsmodus, das nicht auf Anhieb verstanden wurde, und das auch auf seine Sinnhaftigkeit hinterfragt werden sollte.

Es gibt in jedem Test viele Anwender, die wegen eines Usability-Problems eine Aufgabe nicht erledigen können, dafür aber nicht der Software die „ Schuld“ geben, sondern sich selbst („zu hause hätte ich natürlich das Handbuch gelesen“). Ebenso hört man häufig nach Tests Aussagen wie „ Nachdem ich das mal verstanden habe, war es eigentlich ganz einfach“. Das

zeigt sich besonders deutlich in Programmbereichen, die sich mit einem komplexen Thema befassen.

Ein Anwender hat also in der Regel kein großes Problem damit, sich für einen komplizierten Sachverhalt in eine Software einzuarbeiten und mit relativ viel Aufwand herauszufinden, wie er damit umzugehen hat. Wenn das aber bei einfachsten Bereichen nicht klappt, hat er dafür kein Verständnis und es produziert Frust und eine negative Einstellung der Software gegenüber.

Darum sind gerade Probleme, die sich bei eigentlich sehr einfachen Eingabebereichen gezeigt haben, besonders gefährlich: Von einem Bereich, den man auch als unerfahrener Anwender von der Sache her begreift (= die Struktur einer Adresse), und von dessen Umsetzung in der Software, schließt man sehr leicht auf die anderen Bereiche („ Na, wenn das schon mit der einfachen Adresseingabe so losgeht, wie mag dann erst der Rest aussehen, wenn es kompliziert wird“).

Coach

Der Coach führt den Nutzer durch die einzelnen Aufgaben. Der Coach wurde generell als eine gute Idee betrachtet. Allerdings sollten zwei Grundsätze beachten werden:

- Ein Coach muss bei schwierigeren Vorgängen Hilfe bieten; wenn hingegen eine einfache Adresseingabe einen Coach benötigt, ist etwas an der Adresseingabe schon nicht gut gelöst.

Der Coach darf selbst nicht mehr Probleme verursachen als er löst, z.B. durch missverständliche Symbolik, zu kleine Schritte oder Schwierigkeiten in der Navigation.

4 Methodik EUD-Maßnahmenkatalog

Wie in Kapitel 2 bereits beschrieben, ist das Thema EUD breit in verschiedene Forschungsschwerpunkte aufgefächert. Das Wissen zu und über einzelne Schwerpunkte verteilt sich auf verschiedene Wissensträger. Aus diesem Grund wurden in ersten Workshops die Expertisen der mit dem Thema EUD Vertrauten gesammelt und in der Gruppe diskutiert. Als Output entstand so eine Wissenssammlung (Katalog) über verschiedene EUD-Methoden (siehe Abbildung 2). Die Sammlung wurde in einem Wiki-System abgelegt. Das System erlaubt umfangreiche Editierfunktionen beim verteilten asynchronen Arbeiten mit mehreren Personen.

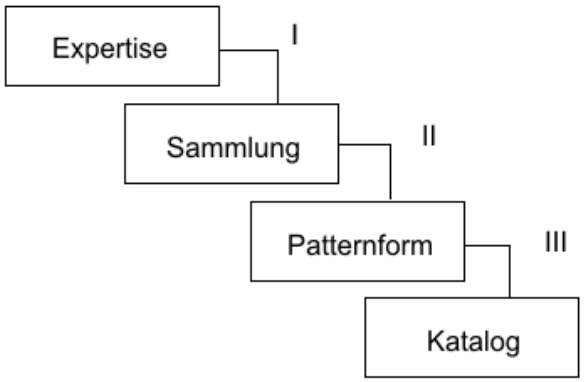

Abbildung 2: Prozessphasen bei der Katalogerstellung

In einem weiteren Schritt ging es darum, die Wissensbausteine in eine einheitliche Form zu überführen. Dazu erfolgte ein Rückgriff auf die so genannte Patternform. Pattern – zu Deutsch Entwurfsmuster – beschreiben wiederkehrende Probleme und geben eine allgemeine Lösungsvorschrift. Durch solche Vorschriften lassen sich zu gefundenen Problembereiche schnell und effizient Lösungsansätze entwickeln.

"Entwurfsmuster vereinfachen die Wiederverwendung von erfolgreichen Entwürfen und Architekturen. Die Darstellung bewährter Techniken als Entwurfsmuster macht die Techniken leichter verständlich, so dass Entwickler neuer Systeme einfacher auf sie zurückgreifen können. Entwurfsmuster helfen zwischen Entwurfsalternativen zu wählen..." (Gamma, Helm et al. 1995)

Zum Zwecke der Überführung in Patternform wurden zu jeder EUD-Methode die Kategorien „Definition“, „Problembereich“, „Lösung“, „Konsequenz“ und ggf. „Ähnliche Pattern“ definiert. Durch diese Fragmentierung der EUD-Methoden erfolgt eine klare Strukturierung, die zu einer hohen Wiederauffindbarkeit bei ähnlichen Problemen führt.

Die Ausformulierung der Patternstruktur erfolgte in einem asynchron verteilten Prozess durch die Personen mit der höchsten Expertise über die jeweilige EUD-Methode. Das Wiki-System ermöglichte dabei den parallelen Zugriff aller Beteiligten. Nach Ausformulierung der Patterns erfolgte in einem weiteren Workshop die Evaluierung des Pattern-Verständnisses, indem alle Beteiligten ihre Anmerkungen zu den Patterns äußerten. Durch diese Phase konnten Missverständnisse ausgeräumt und die Patterns durch den jeweiligen Bearbeiter verbessert werden. Letztlich entstand aus den einzelnen Patterns ein umfangreicher Pattern-Katalog. Zum Zwecke der Übersichtlichkeit erfolgte die Aufgliederung der Patterns in thematische Kategorien. Die einzelnen Kategorien sind im folgenden Abschnitt näher beschrieben.

5 Gliederungskategorien der Pattern

Abbildung 3 zeigt die identifizierten Kategorien der EUD-Patterns, in denen thematisch ähnliche Ansätze zusammengefasst sind. In der Abbildung ist am unteren Rand die Kategorie „Architekturkonzept“ angeordnet. Dabei handelt es sich im weiteren Sinne bereits um Realisierungsansätze, wie „Web Services“ oder „Komponenten Basierte Programmierung“. EUD-Patterns dieser Kategorie befinden sich bereits sehr nah an einer konkreten Umsetzung, wenngleich das Architekturkonzept die konkrete Implementierung nicht vorschreibt.

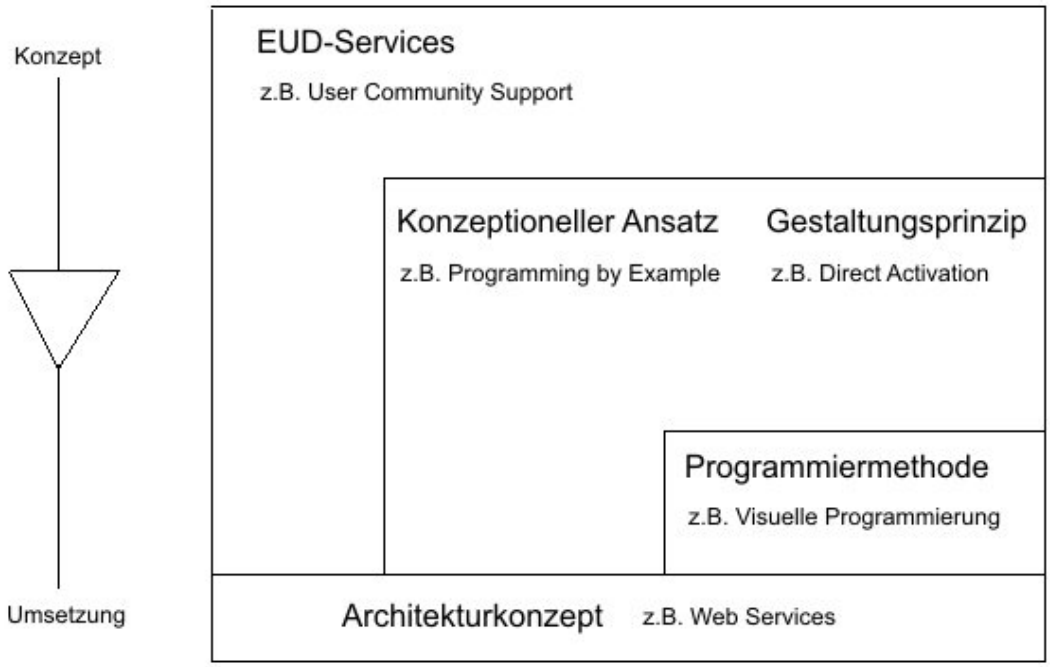

Abbildung 3: Übersicht EUD-Patternkategorisierung

Eine weitere Kategorie die sich sehr nah an das Architekturkonzept anschließt, ist die Kategorie „Programmiermethode“. Konzepte der Natürlichen, der Visuellen oder des Extreme- Programmierung stellen unterschiedliche Ansätze dar, eine Architektur zu erstellen bzw. im Nachgang durch EUD-Methoden zu beeinflussen.

Auf konzeptionell nächst höherer Ebene sind die Kategorien „Konzeptioneller Ansatz“ und „Gestaltungsprinzip“ angesiedelt. In Pattern des „Konzeptionellen Ansatzes“ sind EUD-Methoden beschrieben, die bereits auf hohem abstraktem Level eine EUD-Lösungsvorschrift zu gegebenen Problemen vorgeben. So können durch „Programming by Example“ beispielsweise immer wiederkehrende Aktionsfolgen des Nutzers aufgezeichnet und für eine automatisierte Interaktionsfolge gespeichert werden. Dieser Ansatz ist zunächst unabhängig von der zugrunde liegenden Architektur (Komponentenbasiert o.ä.) und unabhängig von der angewandten Programmiermethode zu deren Realisierung. Auf der gleichen Ebene befindet sich die Kategorie „Gestaltungsprinzip“. Bei dieser Kategorie handelt es sich im Grunde ebenfalls um einen „Konzeptionellen Ansatz“, nur mit dem Unterschied, dass hier ein stärkerer Fokus auf dem Design der Nutzerschnittstelle liegt.

Die Kategorie, die am weitesten von der konkreten Realisierung entfernt liegt, ist mit „EUD-Services“ bezeichnet. EUD-Pattern dieser Kategorie beschreiben sehr abstrakt und High-Level eine EUD-Lösungsvorschrift. So kann durch „User Community Support“ eine Benutzergemeinschaft durch entsprechende Community-Tools in deren Arbeit unterstützt

werden. Dabei können unterschiedliche Konzeptionelle Ansätze und Gestaltungsprinzipien zur Anwendung kommen, weswegen die Kategorie der „EUD-Services“ auch an oberster Stelle vorzufinden ist.

Das folgende Kapitel enthält die nach den einzelnen Kategorien unterteilten EUD-Patterns. Dabei werden zunächst die Pattern beschrieben, die sich bereits sehr nahe an einer konkreten Realisierung orientieren („Architekturkonzepte“). Entsprechend Abbildung 3 erfolgt dann die Beschreibung weg von der Realisierung („Programmiermethode“) und hin zu den stärker konzeptionellen Ansätzen („Konzeptioneller Ansatz“, „Gestaltungsprinzip“ und „EUD-Services“).

6 Pattern von EUD-Methoden

6.1 Architekturkonzept

6.1.1 Web Services

6.1.1.1 Definition

Ein Web Service ist eine Funktion, die zur Unterstützung von Maschine-Maschine Interaktionen über XML-Funktionsaufrufen und -Schnittstellen angesprochen wird. Funktionsaufrufe anderer Systeme erfolgen über den Austausch XML-basierter SOAP-Nachrichten. Das Funktionsinterface wird über ein so genanntes WSDL-Dokument beschrieben.

6.1.1.2 Problembereich

Es gibt einen Mangel an standardisierten Protokollen zum Austausch von verteilt produzierten Funktionsergebnissen zwischen Anwendungen, die auf HTTP aufsetzen. Die Dienste im Netz wurden ursprünglich für Endkunden entwickelt (Nachrichtenaustausch, Shopanwendung, Routenplaner etc.). Diese Dienste sind bezüglich Umfang und Interface an Bedürfnisse des Einzelnen ausgerichtet. Es besteht jedoch der Bedarf Dienstleistungen auch beispielsweise anderen Unternehmen zugänglich zu machen (B2B), damit diese dann entsprechend komplexe und angepasste Dienste entwickeln können (Geschäftsprozesse). Voraussetzung für eine solche variable Dienstintegration sind einheitliche Schnittstellen und Kommunikationsprotokolle. Es besteht der Bedarf nach wieder verwendbaren und flexiblen Softwarekomponenten, die plattformunabhängig einsetzbar sind.

6.1.1.3 Lösung

Dienste werden über plattformunabhängige XML-Schnittstellen bereitgestellt, die von anderen Diensten oder Applikationen dynamisch eingebunden und genutzt werden können. Allgemein existieren Dienstanbieter (Instanz, die einen oder mehrere Dienste anbietet bzw. veröffentlicht), Dienstkonsument (Instanz, welche den Dienst für eigene Zwecke einbindet und nutzt) und teilweise ein Vermittler (Service Broker, der eine Datenbank über alle registrierten Dienste, die vom Dienstkonsument genutzt werden können, enthält). Zur Unterstützung dieser 3 Entitäten existieren folgende 3 Basisstandards

- ***SOAP***: Kommunikationsprotokoll zum Methodenaufruf (XML-RPC) und Datenaustausch (Document)
- ***WSDL*** (Web Service Description Language): Dieser Standard definiert unter anderem die Schnittstelle eines Dienstes (wie lauten die einzelnen Funktionen, welche Parameter werden erwartet, welche Rückgabewerte werden geliefert) und Adresse des Dienstes
- ***UDDI*** (Universal Description, Discovery and Integration): Standard realisiert eine Lookup Funktionalität, die die Registrierung und das Auffinden von Web Services ermöglicht.

6.1.1.4 Konsequenz

Da die Schnittstellen und Funktionsaufrufe in XML formuliert sind, sind Web Service Plattform- und Programmiersprachen unabhängig einsetzbar. HTTP als Transportprotokoll wird durch seine weite Verbreitung von Firewalls selten blockiert. Nachteilig ist die Übertragung im XML-Format, welches geparst werden muss. Dies kann zu Performanceengpässen führen. Zudem ist ein gewisses Basiswissen für die Nutzung erforderlich.

6.1.2 Ambient Services

6.1.2.1 Definition

Ambient Services sind Services, welche mit dem physischen Umfeld des End-Users in Beziehung stehen und in diesem Kontext für diesen von Nutzen sind. Sie können daher auch als Location-Based-Services (LBS) verstanden werden.

6.1.2.2 Problembereich

Im Gegensatz zu stationären Desktop-Systemen bewegen sich Mobile Endgeräte mit ihren Benutzern in ständig wechselnden physischen Kontexten. Dienste / Services sind bestimmten Kontexten zugeordnet. Das mobile Endgerät ist die Zugangsmöglichkeit zu diesen Services. Dabei ergibt sich eine Menge von Problemen: Es muss geklärt werden, wie erkannt wird, wann sich ein Benutzer in einem neuen physischen Umfeld befindet und wie verschiedene Umfelder voneinander abgegrenzt werden können. Dabei könnten Teile einer Anwendung in einem bestimmten Kontext relevant sein und in anderen Kontexten stören. Außerdem muss der Benutzer die Möglichkeit bekommen Services in Umfeldern explorieren zu können.

6.1.2.3 Lösung

Diese Probleme werden durch die Ambient Services gelöst, indem physische Kontexte definiert werden, die Funktionalität in Form von Services dekomponiert wird und die Kontexte den Services zugeordnet werden. Des Weiteren werden Werkzeuge bereitgestellt, die es dem End-Benutzer erlauben Regelsysteme festzulegen, die das spätere Verhalten des Systems bei Wechseln von Kontexten determinieren.

6.1.2.4 Konsequenz

Applikationen sind keine statischen Artefakte mehr, die auf eine bestimmte Anwendungsdomäne hin erstellt wurden, sondern existieren temporär in Form von Kompositionen individuell in Abhängigkeit des Kontextes zur Zeit der Nutzung. Die Funktionalität ist also bereits vorhanden, und der End-User arrangiert und komponiert sein benötigtes Anwendungssystem während der Nutzung. Dies zieht aber auch eine neue und damit ungewohnte Art der Nutzung von Software-Artefakten nach sich. Es handelt sich dabei um End-User-Composing, wobei keinerlei neue Funktionalitäten implementiert werden.

6.1.2.5 Ähnliche EUD-Methoden

Ähnliche EUD-Methoden sind Salient Services, Shared Initiative sowie Direct Activation.

6.1.3 Component-based Programming – Komponentenbasierte Programmierung

6.1.3.1 Definition

Software-Komponenten werden folgendermaßen definiert: "Eine Software-Komponente ist ein Software-Element, das konform zu einem Komponentenmodell ist und gemäß eines Composition Standard ohne Änderungen mit anderen Komponenten verknüpft und ausgeführt werden kann." (Heineman, Council 2001)

Komponentenbasierte Programmierung adressiert somit zwei Themen: Die Erstellung von Software-Komponenten und das Zusammenfügen von Software-Komponenten zu komponentenbasierten Systemen.

6.1.3.2 Problembereich

Ausgehend von monolithischen System-Architekturen lässt sich folgendes beobachten: Je größer und komplexer Softwaresysteme sind, umso schwieriger ist es diese zu warten oder den sich verändernden Anforderungen anzupassen. End-Benutzer können ihr monolithisches Software-System nicht selbst durch weitere Module/Plug-Ins/Komponenten erweitern.

6.1.3.3 Lösung

Der Entwurf, die Erstellung und die Wartung von Softwaresystemen erfordern bei steigender Komplexität softwaretechnische Maßnahmen, die eine Dekomposition des Gesamtsystems in funktional abgeschlossene System-Komponenten ermöglichen. Dazu sind mindestens zwei Voraussetzungen zu schaffen, zum einen müssen Software-Komponenten in ihrer Struktur und in ihren Schnittstellen nach außen einer Komponenten-Spezifikation entsprechen. Zum anderen muss ein Komponenten-Framework bereitgestellt werden, welches die so spezifizierten Komponenten aufnimmt und ausführbar macht. Das Framework ist somit eine Art "Betriebssystem" für die einzelnen Komponenten. Es bietet wiederum selbst Funktionalität an, die von allen Komponenten gemeinsam genutzt werden kann.

Abhängigkeiten zwischen Software-Komponenten dürfen nicht hart im Code verdrahtet sein, vielmehr sollten sich die Komponenten nach außen in Ihrer Funktionalität selbst beschreiben und Schnittstellen anbieten, welche eine lose Kopplung mit anderen Komponenten ermöglichen. Software-Komponenten verbergen Implementierungsdetails und repräsentieren Funktionalität. Dies bedeutet, dass Endbenutzer ohne Programmierkenntnisse ein System anpassen oder erweitern können, indem sie ganze Komponenten hinzufügen oder entfernen, ohne Kenntnis von der eigentlichen Implementierung der Komponenten und des Gesamtsystems.

6.1.3.4 Konsequenz

Jede Komponente repräsentiert einen Teil der Gesamt-Funktionalität. Das Gesamtsystem wird in einzelne, wiederverwendbare Komponenten zerlegt. Dadurch ist ein einfacheres Hinzufügen und Entfernen von Funktonalität möglich und die System-Architektur leicht skalierbar. Die Anpassung der einzelnen Komponenten erfordert allerdings eine Programmierung und kann Abhängigkeiten anderer Komponenten beeinträchtigen

6.1.3.5 Ähnliche EUD-Methoden

Eine ähnliche EUD-Methode ist der Ansatz der Serviceorientierten Architekturen.

6.1.4 Customization Gulf - Anpassungsgraben

6.1.4.1 Definition

Die Trennung zwischen der Benutzerschnittstelle und der Applikationsfunktionalität zeigt sich in den meisten der aktuell verfügbaren Softwaresysteme durch einen Graben zwischen der Oberfläche und der tiefen Anpassung. Dieser Graben, der Customization Gulf genannt wird, ist das Ergebnis von Software-Technik-Praxis und widerspricht den eigentlichen Anforderungen an Anpassbarkeit durch den System-Benutzer.

6.1.4.2 Problembereich

Der Customisation Gulf ist charakterisiert durch zwei eng miteinander verbundene Problembereiche:
Auf der einen Seite durch den Grad der möglichen Anpassung: Häufig sind nur oberflächliche Anpassungen im Sinne von Konfigurationen möglich.
Auf der anderen Seite ist der Customisation Gulf durch die Sprache der Anpassung gekennzeichnet. Traditionell bieten Softwaresysteme nur eingeschränkte Möglichkeiten, mit

denen Benutzer Anpassungsanforderungen so ausdrücken können, wie es ihre Kompetenz zulässt.
Diese beiden Probleme erlauben es dem Benutzer nicht, tief in die Funktionalität des Systems einzugreifen, um dort Anpassungen vorzunehmen. Diese Art der Anpassung geht weit über die bloße Anpassung der Benutzerschnittstellen hinaus.

6.1.4.3 Lösung

Dies lässt sich durch die Entwicklung von Sprachen/Ausdrucksweisen lösen, die es dem Benutzer ermöglichen, in der unter der Benutzerschnittstelle liegenden Software-Architektur Anpassungen vorzunehmen. Zudem muss schon bei dem softwaretechnischen Entwurf von Systemen eine direkte Integration der Benutzer-Schnittstellen-Architektur und der Architektur der Applikationslogik vorgesehen werden.

6.1.4.4 Konsequenz

Dadurch wird die Anpassung des Benutzers in der Tiefe des Systems ermöglicht. Dabei findet wirklich eine Anpassung der Funktionalität und nicht eine Anpassung des Zugangs zur Funktionalität statt. Dieser Ansatz ist aber praktisch nur möglich bei Neuentwicklungen.

6.1.4.5 Ähnliche EUD-Methoden

Eine ähnliche EUD-Methode ist die Direkt Activation.

6.1.5 Domain Oriented Design Environments

6.1.5.1 Definition

Domain Oriented Design Environments (DODE) sind anwendungsbezogene Entwicklungsumgebungen, die eine integrierte Nutzung und Entwicklung von Softwareartefakten erlauben (Fischer 1994).

6.1.5.2 Problembereich

Universelle Entwicklungsumgebungen stellen Endbenutzer vor das Problem, dass sie mehr Gestaltungsoptionen bieten als im Anwendungskontext gebraucht werden.

6.1.5.3 Lösung

DODE integrieren Nutzung und Entwicklung in eine Umgebung. Daneben greifen DODE Konzepte aus der Anwendungsdomain in der Gestaltungsumgebung mit auf. Die Domainorientierung verringert dabei die Kluft, die Endbenutzer davon abhält, aktiv in die Gestaltung der Softwareartefakte einzugreifen. Darüber hinaus lässt sich DODE in einen größeren sozio-technischen Kontext stellen (Fischer 1996). Dabei lassen sich drei große Bereiche herausstellen:

- Eine facettenreiche, domainunabhängige Architektur samt eines Baukasten von domainspezifischen Erweiterung, aus denen spezifische DODEs zusammengestellt werden;
- Mechanismen für die Integration der speziellen Bestandteile;
- ein evolutionäres, partizipatives Prozessmodell, wie das “Seeding, Evolutionary growth, and Reseeding” (SER)-Modell.

6.1.5.4 Konsequenz

Es findet eine Übertragung von Konzepten aus den Nutzungskontexten in die Gestaltungsumgebung statt. Dies führt zu einer Integration von Gestaltung und Nutzung. Der Ansatz erhebt einen ganzheitlichen Anspruch. Da der Ansatz sehr allgemein formuliert ist, ist es schwierig das Prinzip auf einen konkreten Fall anzuwenden.

6.2 Programmiermethoden

6.2.1 Extreme Programming

6.2.1.1 Definition

Extreme Programming ist ein inkrementelles Softwareentwicklungsmodell. Im Gegensatz zum Wasserfallmodell werden immer wieder neue (kleinere) Teilziele formuliert und umgesetzt. Da Kunden zu Beginn der Entwicklung ihre Anforderungen noch nicht vollständig kennen, ermöglicht das Modell die Umsetzung von Anforderungen auch zu späteren Zeitpunkten.

6.2.1.2 Problembereich

Die Entwicklung eines Softwaresystems als ganzes ist kosten- und zeitaufwendig. Oftmals stellen sich Probleme im Umgang mit dem System erst dann heraus, wenn das System fertiggestellt und im Einsatz ist.

6.2.1.3 Lösung

Anstatt ein Softwaresystem von Beginn an als Ganzes zu realisieren, erfolgt zuerst die Implementierung eines Grundgerüstes, auf das dann schrittweise weitere Funktionalitäten aufgebaut werden. Dabei wird in Folge eine Menge von kleineren (inkrementellen) Zielen abgearbeitet. Zur Erreichung dieser Ziele kommen folgende Methoden zur Anwendung:

- incremental programming
- incremental planing
- incremental design
- incremental requirements gathering
- incremental deployment

6.2.1.4 Konsequenz

Vorteile dieser Methode sind eine schnellere Entwicklung lauffähiger Softwaresysteme und eine frühere Identifizierung von Problemen (teilweise vergleichbar mit Rapid Prototyping). Weiterhin können neue Anforderungen an das System auch noch während des Entwicklungsprozesses in die Entwicklung einfließen.

Nachteilig sind die hohen Anforderungen an die Programmierer bzgl. Dokumentation und Kommentaren (ein inkrementelles Teilziel kann beispielsweise von ganz anderen Programmierern bearbeitet werden – hier ist eine vollständige Architektur- und Codeübersicht zwingend erforderlich).

6.2.1.5 Ähnliche EUD-Methoden

Eine verwandte EUD-Methode ist das inkrementelle Programmieren.

6.2.2 Incremental Programming, inkrementelles Programmieren

6.2.2.1 Definition

Inkrementelles Programmieren bezieht sich auf sehr kleinteilige, eng begrenzte Veränderungen an existierenden Softwaresystemen: "Incremental Programming is close to traditional programming but limited to change a small part of a programm, such as a method in a class. It is easier than programming from scratch." (Lieberman et al. 2006)

6.2.2.2 Problembereich

Die Entwicklung eines Software-Systems als ganzes ist kosten- und zeitaufwendig. Oftmals stellen sich Probleme im Umgang mit dem System erst dann heraus, wenn das System fertig und im Einsatz ist.

6.2.2.3 Lösung

Incremental Programming ist eine Vorgehensweise, die unter anderem auch bei Extreme Programming zum Einsatz kommt. Anstatt ein Softwaresystem von Beginn an als Ganzes zu realisieren, erfolgt zuerst die Implementierung eines Grundgerüstes, auf das dann schrittweise weitere Funktionalitäten aufgebaut werden. Dabei wird in Folge eine Menge von kleineren (inkrementellen) Zielen abgearbeitet.

6.2.2.4 Konsequenz

Vorteile dieser Methode sind eine schnellere Entwicklung lauffähiger Softwaresysteme, frühere Identifizierung von Problemen (teilweise vergleichbar mit Rapid Prototyping) und dass neue Anforderungen an das System auch noch während des Entwicklungsprozesses einfließen können.

Nachteilig sind die hohen Anforderungen an die Programmierer bzgl. Dokumentation und Kommentare (ein inkrementelles Teilziel kann beispielsweise von ganz anderen Programmieren bearbeitet werden – hier ist eine vollständige Architektur- und Codeübersicht zwingend erforderlich).

6.2.3 Natural Programming - Natürliche Programmierung

6.2.3.1 Definition

Natural Programming beschäftigt sich mit generellen Prinzipien, Methoden und Programmiersprachen, welche den Aufwand zum Erlernen der Programmierung für nicht professionelle Programmierer signifikant senken sollen. Durch Studien an Nicht-Programmierern soll herausgefunden werden, wie diese Aufgaben formulieren, die ein Computer für sie bearbeiten soll. Dadurch sollen mehr "natürliche" Programmiersprachen entwickelt werden, welche dann erheblich leichter zu erlernen sind.

6.2.3.2 Problembereich

Natural Programming adressiert Anwendungsbereiche, in denen von Menschen, die keine professionellen Programmierer sind, erwartet wird, dass sie programmieren.

- ***Programmieren für Kinder:*** Entwicklung von neuen Programmiersprachen, welche es Kindern und ihren Lehrern erleichtern Spiele oder Lern-Simulationen zu erstellen.

- ***Anpassbare Systeme:*** Anpassung wird hier definiert als die Tätigkeit der Veränderung einer Computer-Anwendung im Kontext deren Nutzung. Anpassung adressiert somit zwei Felder: Zum Einen die Entwicklung, zum Anderen die Nutzung. Anpassung ist somit beides, im Sinne von Entwicklung während der Nutzung, um Anforderungen gerecht zu werden, die während der Entwicklung nicht berücksichtigt wurden oder nicht bekannt waren. Eine wohldefinierte und benutzbare Programmiersprache ist der Schlüssel für effektive Anpassbarkeit.

- ***General scripting:*** Es gibt viele Situationen, in denen wiederkehrende Aktionen ausgeführt werden müssen. Das Schreiben eines Skripts oder eines Makros, um diese wiederkehrenden Aufgaben zu erledigen, erlaubt es dem Benutzer effektiver zu arbeiten. Solche Script-Sprachen sind bekannt von der Unix-Shell oder aus diversen

Office-Produkten. Microsoft-Office-Produkte nutzen Visual Basic als Skriptsprache. Die Fähigkeit, solche wiederverwendbaren Skripte zu schreiben, kann für den Benutzer sehr hilfreich sein.

- ***World-Wide-Web:*** Eine Reihe von interaktiven Tools zum Erstellen von statischen Web-Seiten sind auf dem Markt vorhanden (z.B. Microsoft-Powerpoint, Macromedia Dreamweaver, Adobe PageMill). Dennoch ist es für die Erstellung von dynamischen interaktiven Web-Inhalten nach wie vor erforderlich Skriptsprachen wie Perl, Java-Script, PHP, usw. zu erlernen. Das Vorhandensein von besser benutzbaren und leichter erlernbaren Sprachen würde es einer größeren Spannweite an Menschen ermöglichen, solche dynamischen Webseiten zu konstruieren.

Weitere eher speziellere Anwendungsfelder sind Simulation Setup, Multimedia Authoring, Educational Software, Controlling Robots and Process Control und Intelligent Agents.
"Tatsächlich würden alle Anwender von der Möglichkeit profitieren, durch Scripting ihre wiederkehrenden Aufgaben zu automatisieren und ihre Anwendungen an ihre Anforderungen anzupassen."(vgl. Myers 1998)

6.2.3.3 Konsequenz

Mit mehr natürlichen Programmiersprachen und -umgebungen ist eine steigende Zahl von Menschen zu erwarten, die in der Lage sein werden, zu programmieren. Somit ist ein schnellerer und effizienterer Implementierungsprozess zu erwarten. Zudem sind die so erstellten Programme weniger fehlerbehaftet und leichter lesbar und damit leichter wartbar, als Programme, welche mit heutigen Programmiersprachen programmiert wurden. Somit können auch Laien in Open-Source-Communities integriert werden.

6.2.4 Parametrisierung

6.2.4.1 Definition

Unter Parametrisierung versteht man die Auswahl von Implementierungsvarianten eines Anwendungssystems durch Setzen bestimmter Attribute, sogenannter Parameter, im Anwendungssystem. Diese Parameter müssen innerhalb eines vordefinierten Wertespektrums gewählt werden. Ein parametrisierbares Business-Objekt führt zu jedem Variation Point alle vom Systementwickler vorgesehenen Varianten mit sich. Es wird parametrisiert, indem allen Parametern, die die Ausprägung von Variation Points bestimmen, ein Wert zugewiesen wird.
Parametrisierung ist neben der Konfigurierung und der Programmierung eine Form des Customizings von Anwendungssystemen. Sie hat Auswirkungen auf die Funktionsweise der Software und der Benutzeroberfläche.

Auch in Verbindung mit Extensions kann Parametrisierung eingesetzt werden: Das Business-Objekt ermittelt dann mit Hilfe von Parametern, mit welchen Extension-Varianten es sich verknüpfen soll und damit, wessen Methoden es nutzen muss. Besonders nutzbringend ist dies, falls dieselbe Variante an verschiedenen Stellen innerhalb des Anwendungssystems verwandt wird oder die Wahl einer bestimmten Variante die Verwendung von Varianten an anderen Variation Points desselben Business-Objekts oder sogar anderer Business-Objekte bedingt. In diesem Fall kann ein Parameter die konsistente Variantenwahl innerhalb des Anwendungssystems steuern. Die Parametrisierung unterstützt das Formulieren von Dependency Links. Mit deren Hilfe können sinnvolle Variantenkombinationen verschiedener Variation Points aggregiert und mittels Setzen eines Parameters gemeinsam ausgewählt werden.

6.2.4.2 Problembereich

Standardsoftware bietet oft Funktionsüberhang und keine spezielle Anpassung z.B. für bestimmte Branchen.

6.2.4.3 Lösung

(De-)Aktivieren von Funktionalität und Anpassen der Erscheinungsweise des Programms durch das Setzen von, durch die Entwickler vorgedachten, Parametern.

6.2.4.4 Konsequenz

Das Setzen der Parameter ist durch die End-User einfach zu realisieren, da er nur noch eine Variante auswählt. Die verschiedenen Konfigurationen müssen dabei aber vom Programmierer vorgedacht werden.

6.2.4.5 Ähnliche EUD-Methoden

Dieses Konzept ist eine Komponente des Customizings (Konfigurierung und Programmierung)

6.2.5 Visuelle Programmierung

6.2.5.1 Definition

Unter visueller Programmierung wird die Anwendung von visuellen Techniken zur Spezifikation von Programmen verstanden. Eine visuelle Programmierumgebung ist eine integrierte Entwicklungsumgebung (IDE) mit einer visuellen Entwicklungsoberfläche, die es erlaubt, die Benutzeroberfläche der zu erstellenden Anwendung graphisch zu bearbeiten. In der Regel wird dabei eine Oberfläche nach dem "Baukastenprinzip" erstellt, und während der Bearbeitung genauso oder ähnlich angezeigt, wie sie später im lauffähigen Programm erscheint (WYSIWYG). Darüber hinaus können auch Funktionalitäten durch visuelle Techniken modelliert werden. Dieses Vorgehen wird auch als RAD (Rapid Application Development, schnelle Anwendungsentwicklung) bezeichnet.

6.2.5.2 Problembereich

Herkömmliche Programmierung erfordert hohen Zeit- und Arbeitsaufwand, da Programmiercode von Hand eingetragen werden muss. Darüber hinaus muss der Entwickler über gute Programmierkenntnisse verfügen.
Durch RAD lassen sich mit geringen Programmierkenntnissen in sehr kurzer Zeit Benutzeroberflächen und Funktionalitäten visuell modellieren.

6.2.5.3 Lösung

Manche Anwendungen, besonders Benutzeroberflächen (GUI), können mit Hilfe von RAD in einem Bruchteil der Zeit, die für konventionelle Programmierung erforderlich wäre, erstellt werden. Auch automatische Vervollständigung des Codes noch bei der Eingabe ist häufig Teil des RAD-Konzepts. Ebenfalls zur Visuellen Programmierung dienen so genannte Autorensysteme, die aber meist weniger auf Programmierer als viel mehr auf Graphiker ausgerichtet sind und zur Erstellung interaktiver Multimedia-Anwendungen dienen. Dabei wird meist der erzeugte Programmcode gänzlich vor dem Ersteller verborgen. Daneben wird der Begriff Visuelle Programmierung auch für Programmiersprachen benutzt, bei denen graphisch z.B. mit Statechart-Diagrammen programmiert wird.

6.2.5.4 Konsequenz

Durch die Anwendung des WYSIWYG lassen sich mit geringen Programmierkenntnissen schnelle Ergebnisse in der Anwendungsentwicklung erzielen. Meist sind solche Systeme aber in ihrer Flexibilität eingeschränkt.

6.2.6 Makroprogrammierung

6.2.6.1 Definition

Ein Makro ist ein Programmcode-Fragment, das vom Interpreter durch eine vorher definierte größere Programmstruktur ersetzt wird. Durch diese Ersetzungsstrategie können Nutzer häufig wiederkehrende Problemlösungsstrukturen leichter und schneller in die eigene Arbeitsumgebung einbetten.

6.2.6.2 Problem

Der End User will die Funktionsweise der Arbeitsumgebung speziell auf seine Bedürfnisse abstimmen und Verfahrensabläufe speichern und wiederholbar machen.

6.2.6.3 Lösung

Durch Aufzeichnung von Makros lassen sich, sich wiederholende gleiche Aktionen aufzeichnen und wiederholt (automatisch) ausführen. Dies schließt die Bearbeitung des Textes, aber auch die Bedienung des Programms ein. Die erstellten Makros lassen sich in die Menüleiste, aber auch in Textelemente einbetten und sind so schnell zugänglich.

6.2.6.4 Konsequenz

Durch die mögliche Anpassung an die persönlichen Arbeitsabläufe lässt sich die tägliche Arbeit beschleunigen und vereinfachen. Die Erstellung der Makros ist sehr intuitiv, da der Nutzer nur seine gewohnten Aufgaben durchführt und diese vom System dann aufgezeichnet werden. Ein weiterer Vorteil der Makros liegt in der Möglichkeit zur Weitergabe an Nutzer mit gleichen oder ähnlichen Abläufen und Präferenzen. Problematisch ist allerdings die Anpassung der Makros. Entweder müssen diese dann komplett neu aufgezeichnet werden oder für die Anpassung sind Programmierkenntnisse nötig.

6.2.7 Skriptsprachen

6.2.7.1 Definition

Typische Elemente von Skriptsprachen sind: Die implizierte Deklaration von Variablen und dynamischen Funktionsnamen, dynamische und automatische Typumwandlung, ein automatisches Speichermanagement und eine dynamische Klassenzugehörigkeit oder prototypenbasierte Vererbung. Die Skripte werden häufig ohne getrennte Übersetzungsphase ausgeführt, was zu einer späten syntaktischen Fehlererkennung führt, da Fehler erst bei der Ausführung erkannt werden. Die Binaries einer Skriptsprache werden meist im Textformat gespeichert.

6.2.7.2 Problembereich

Jeder User hat unterschiedliche Anforderungen bezüglich individueller Funktionalität, Aussehen und Verhaltensweise des Programms.

6.2.7.3 Lösung

Durch den Einsatz von Skriptsprachen lassen sich die Programme auf relativ hoher Ebene mit relativ geringem Einarbeitungsaufwand individualisieren. Dies ist dadurch bedingt, dass Skriptsprachen meist eine große Sammlung von vorhandenen High-Level-Operationen

anbieten und den Programmieren Low-Level-Programmieraufgaben wie z.B. Speichermanagement und komplexe Schleifenkonstrukte abnimmt.

6.2.7.4 Konsequenz

Da die low-level-Aufgaben (wie z.B. Speichermanagement, etc.) vom Interpreter/Compiler übernommen werden, sind diese Sprachen meist recht einfach zu erlernen. Da zudem oft viele High-Level-Operationen bereitgestellt werden, ist eine schnellere Programmierung möglich. Ein Nachteil von Skriptsprachen ist die langsamere Ausführung, da das Skript meist erst noch interpretiert werden muss.

6.3 Konzeptioneller Ansatz

6.3.1 Cognitive Criteria of Programming Enviroments / Cognitive Dimensions

6.3.1.1 Definition

Cognitive Dimensions of Notification (CD), zu deutsch kognitive Dimensionen der Benachrichtigung, sind ein Framework um die Usability von Notationssystemen (z.B. Word-Prozessoren, computerunterstützte Design-Tools oder Musik-Notation) sowie von Informationsartefakten (z.B. Uhren, Radios oder central heating controllers) zu beschreiben.
Ziel des CD-Frameworks ist das zur Verfügung stellen eines Vokabulars, welches von den Entwicklern genutzt werden kann, um die kognitiven Implikationen ihrer Design-Entscheidung zu untersuchen, zu erreichen.

Designer von Notations-Systemen haben realisiert, dass ihre Entscheidungen eine Auswirkung auf die Usability haben, und dass die Usability-Probleme mit den Notationen kognitive Rückwirkungen auf die Nutzer haben. Viele Designer wissen dies aber nur auf intuitiver Ebene. Das macht es für sie umso schwieriger Usability-Aspekte zu diskutieren. Besonders, da sie selten eine formale Ausbildung in Kognitiver Psychologie besitzen. (Blackwell 2006)

6.3.1.2 Problembereich

Cognitive Dimensions widmet sich dem Problem, wie man die Güte von Programmierumgebungen evaluieren kann.

6.3.1.3 Lösung

Mit CD wurde ein Satz von Kriterien entwickelt, mittels derer sich Güte messen lässt.

6.3.1.4 Beispiele

Mittels Cognitive Dimensions wurden insbesondere visuelle Programmierumgebungen evaluiert. Der Anwendungsbereich des Frameworks wurde aber auch auf andere Bereiche ausgeweitet.

6.3.1.5 Konsequenz

Bei diesem Konzept lernt der Nutzer während der Nutzung die Ablaufmechanismen kennen. Allerdings bekommt er, auch wenn nicht angepasst wird, zusätzliche Informationen aufgebürdet.

6.3.1.6 Ähnliche Pattern

Ähnliche Pattern sind der Gentle Slope of Complexity, sowie der Customization Gulf.

6.3.2 Ambient Prototyping

6.3.2.1 Definition

Mit Ambient Prototyping werden Methoden bezeichnet, die es ermöglichen, Programmieren und Konfigurieren mit Kreativtechniken durchzuführen, die keinen direkten Bezug zur Computerwelt haben, sondern in der Realwelt (Ambient Environment) durchgeführt werden.

6.3.2.2 Problembereich

Die Methoden bieten sich dann an, wenn man mit Endbenutzern zusammenarbeitet, die nicht mit computerbasierten Benutzerinterfaces arbeiten können oder wollen (z.B. Kinder, Behinderte, Ältere Menschen). Auch kann die haptische Dimension der Kreativitätstechniken ihren Einsatz motivieren.

6.3.2.3 Lösung

Lösung sind Programmier- und Konfigurationskonzepte, die ein realweltliches Arbeiten ermöglichen.

6.3.2.4 Konsequenz

Es fehlen zurzeit noch veröffentlichte Machbarkeitsstudien und Evaluationen dieser Ideen.

6.3.2.5 Ähnliche EUD-Methoden

Je nach Ausrichtung können Konzepte des Visual Programming oder von Domain-Oriented Design Environments verwendet werden.

6.3.3 Augmented Reality Tailoring

6.3.3.1 Definition

„Augmented Reality“ lässt sich als "virtuell bereicherte Realität" beschreiben. Das Konzept des Tailoring ist eine Erweiterung, das die "virtuellen Ergänzungen" persönlich auf den Einzelnen zugeschnitten bzw. angepasst werden können.

6.3.3.2 Problembereich

Die durch Software produzierten Daten und Lösungen müssen den Nutzern im Kontext der jeweiligen Aufgabe präsentiert werden. In vielen Anwendungsdomänen stößt man mit herkömmlichen textuellen Beschreibungen schnell an die Grenzen. Anwendungen müssen hochflexibel sein und auf den Einsatz zugeschnittene und angepasste Ausgaben liefern. Für detailgetreue und plastische Repräsentation sind andere als herkömmliche Ausgabeformate notwendig – z.B. 3D-Visualisierungen oder grafische Animationen.

6.3.3.3 Lösung

"Virtuelle Realität" ermöglicht die plastische und realistische Wiedergabe von Sachverhalten und die Interaktion mit den Objekten. Erfolgt die Präsentation nicht in einem globalen Kontext (für alle Nutzer gleich), sondern angepasst an die Bedürfnisse eines Einzelnen, so sind vielfältige Unterstützungsmöglichkeiten denkbar.

6.3.3.4 Konsequenz

Die sich daraus ergebenden Vorteile sind zum einen eine realistischere Wiedergabe sowie personalisierte Ergebnisse. Ein möglicher Nachteil ist die zum Teil hohe Rechenleistung, die für virtuelle Repräsentationen notwendig ist.

6.3.4 Exploration Environments

6.3.4.1 Definition

Exploration Environments unterstützen den Benutzer beim selbstgesteuerten Erlernen von Groupware. Die Simulation der Auswirkungen eigener Aktionen, auf die Arbeitsumgebung anderer Nutzer, ermöglicht den Benutzern die Arbeitsweise von Groupware zu erkennen und zu verstehen.

6.3.4.2 Problembereich

Es gibt zwei grundlegende Formen des Erlernens von Anwendungen: Anwendungen lassen sich fremdgesteuert zum Beispiel durch Schulungen und Manuals, oder selbstgesteuert durch Exploration im Arbeitskontext erlernen. Problematisch ist die selbstgesteuerte Exploration bei Groupware, da dort eine Exploration nur aus eigener Sicht möglich ist. Auswirkungen eigener Aktionen auf andere Arbeitsbereiche sind dagegen nicht sichtbar, und so nur schwer nachvollziehbar.

6.3.4.3 Lösung

Explorationsumgebungen simulieren Benutzerschnittstellen und ermöglichen somit das Testen und Erproben eigener Aktionen hinsichtlich der Auswirkungen auf die Arbeitsbereiche anderer Nutzer.

6.3.4.4 Konsequenz

Explorationsumgebungen helfen somit das Verständnis von Groupware-Funktionen, sowie die Aufgabenausführung zu verbessern und zu beschleunigen.

6.3.5 Programming by Example

6.3.5.1 Definition

Programming by Example-Systeme zeichnen das Interaktionsverhalten des Nutzers mit dem System auf und generieren daraus eine Routine. Das Konzept kann mit Programming by Demonstration gleichgesetzt werden (Lieberman, H.; Faaborg, A 2006).
Bei Lieberman wird Programming by Example als Oberbegriff für folgende Techniken definiert:

Automatic Programming (auch mit Programming by Input/Output Examples bezeichnet): Dies bezeichnet die Fähigkeit eines Systems von einem Beispiel zu generalisieren. Dabei gibt der Programmierer Beispiele vor und das Programm muss dann selbst die richtige Methode finden, um den Input in den Output umzuwandeln.

Programming by Demonstration: Der Programmierer (oder Endnutzer) vollzieht Aktionen mit Beispieldaten die das Programm aufzeichnet. Dadurch lassen sich Aktionen generalisieren.

Programming by Teaching (auch mit Instructible Systems oder Agents bezeichnet): Erweiterung des Programming by Demonstration Ansatz um verbale und gesturale Hinweise oder Eingaben durch Nutzer.

6.3.5.2 Problembereich

Zur Erledigung von Aufgaben führen Nutzer Aktionsfolgen mit Software-Systemen durch: A1→A2→A3→…→An. Für ähnliche Ziele sind die Aktionsfolgen oft gleich, jedoch müssen Nutzer ohne Unterstützung immer wieder die gleichen Aktionsfolgen durchführen. Ein Beispiel dafür ist die Aktionsfolge bei einem Kauf in einem Online Shop: Artikel suchen →

Artikel auswählen → Details anzeigen → Artikel in Warenkorb legen → Warenkorb auswählen → persönliche Daten eingeben → etc.

6.3.5.3 Lösung

Die Aktionsfolgen zur Erreichung bestimmter "High-Level Ziele" werden aufgezeichnet. Die aufgezeichnete Aktionsfolge kann dann zu einer generalisiserten Aktion zusammengefasst werden. Nutzer können so ihre Ziele deutlich schneller erreichen. Programming by Example unterscheidet sich dabei dahingehen von Makro-Aufzeichnung, dass die Eingaben keine Konstanten sondern Variablen darstellen: Eine Makro-Aufzeichnung würde beispielsweise immer nur Buch X bestellen. Um, wie in unserem Beispiel gewünscht, auch andere Artikel kaufen kaufen zu können, werden Variablen benötigt (beispielsweise den Suchbegriff nach Artikel in einem Bestellprozess).

6.3.5.4 Konsequenz

Programming by Example hilft den Nutzer ihre Ziele schneller zu erreichen. Zudem bietet das Konzept Parametrisierung und, im Vergleich zu Makros, eine höhere Dynamik. Es lassen sich allerdings nicht alle Aufgaben mit Programming by Example generalisieren – zum Beispiel ständig wechselnder Input in allen Aktionsschritten ist nicht umsetzbar.

6.3.6 Salient Services, Saliente Dienste

6.3.6.1 Definition

Der Begriff der "Salient Services" bezeichnet einzelne Anwendungskomponenten, welche von Nutzern fall- und bedarfsweise als Dienste innerhalb ihrer eigenen Anwendung genutzt, abonniert, wieder abbestellt, werden können, etc..

6.3.6.2 Problembereich

"Salient" bedeutet hervorstechend, in der Sozialpsychologie bezeichnet das Konzept der Salienz "individuelle situative Relevanz": So mag es für einen Menschen in vielen Situationen unerheblich sein, welches Geschlecht er/sie hat, aber beim Besuch einer öffentlichen Sauna am "Frauentag" wird diese Geschlechtszugehörigkeit plötzlich relevant (→ salient).
Während jedoch das Konzept der Ambient Services die lokale Eingebettetheit dieser Dienste in einen konkreten räumlichen Kontext betont, fehlt dem Konzept der Salient Services dieser lokale oder räumliche Bezug. Stattdessen wird mit der Salienz ein situativer Bezug zu aktuellen (individuellen) Bedürfnissen und Präferenzen hergestellt.

6.3.6.3 Lösung

Das Angebot bestimmter Ambient Services ändert sich, wenn ich einen Umkleideraum verlasse und stattdessen den Poolbereich eines öffentlichen Schwimmbades betrete. Das Angebot von Salient Services ist hingegen weitgehend ortsunabhängig, es wird vielmehr definiert durch Kontextbedingungen, die in einer konkreten Situation für den Nutzer relevant sind, z.B. eine soziale Rolle, die ein Nutzer in einer aktuellen Interaktionssituation innehat: Als Vater, der gerade seine kleine Tochter irgendwo abholt, benötigt er möglicherweise andere Dienste (technische Unterstützung) als als Turnier-Kegelsportler, der mit seinem Kegelverein um den Aufstieg in die nächste Liga kämpft.

6.3.6.4 Konsequenz

Die angebotenen Dienste sind an "reale" und situationsbedingte Bedürfnisse des Einzelnen anpassbar. Dadurch wird ein situativer Bezug der Funktionalität hergestellt. Als problematisch erweist sich bei der Realisierung der salienten Dienste der praktische Umgang mit der Situationsregistrierung.

6.3.6.5 Ähnliche EUD-Methoden

Einen ähnlichen Ansatz verfolgen die Ambient Services.

6.3.7 Shared Initiative als gemeinsame Anstrengung von Adaptivity and Adaptability

6.3.7.1 Definition

Flexible Softwaresysteme passen sich entweder selbst automatisch an Kontextbedingungen an (Adaptivity) oder lassen sich durch den Nutzer anpassen (Adaptability). Shared Initiative ist die Kombination dieser beiden Anpassungsprinzipien in einer Anwendung.

6.3.7.2 Problembereich

Es ist unmöglich, während der Systementwicklung alle möglichen Einsatzkontexte und Bedürfnisse potentieller Nutzer vorherzusehen.

6.3.7.3 Lösung

Shared Initiatives als flexible Softwaresysteme, ermöglichen die situationsspezifische Anpassung während der Laufzeit. Die Anpassung erfolgt sowohl in Abhängigkeit vom Kontext (zum Teil automatisch) und durch Nutzerinput (explizit).

6.3.7.4 Konsequenz

Software-System berücksichtigt Kontext und Nutzerinput nun gleichermaßen. Dadurch bietet sich die Möglichkeit der Erstellung feingranularer Dienste (Kontext und Input lassen sich situationsbedingt und priorisiert verarbeiten). Die Schwierigkeit dabei liegt in der Priorisierung von Kontext und Input.

6.3.7.5 Ähnliche EUD-Methoden

Einen verwandten Ansatz dazu verfolgen Ambient Services.

6.3.8 User-Initiated Design, nutzer-initiierte Gestaltung

6.3.8.1 Definition

User-Initiated Design bezieht sich auf Gestaltungsaktivitäten von Softwaresystemen, die von den Nutzern, nicht von den Entwicklern der Systeme ausgehen. Es existieren zudem auch Formen delegierter EUD-Projekte, in denen Nutzer zwar Anpassungsprozesse initiieren, die eigentliche Entwicklungstätigkeit dann aber an (semi-) professionelle Experten delegieren.

6.3.8.2 Problembereich

Im Gegensatz zu herkömmlicher Systementwicklung und auch der Mehrzahl der Participatory Design-Ansätze (PD) geht End-User Development in erster Linie von den Endnutzern und nicht von professionellen Entwicklern oder Designern aus. Obwohl auch bei PD die Partizipation von Nutzern am Entwicklungsprozess eine entscheidende Rolle spielt, werden entsprechende Projekte doch in der Regel von professionellen Entwicklern initiiert, geplant sowie die Prozesse von diesen koordiniert, moderiert und gesteuert.

6.3.8.3 Lösung

Prozesse der Aneignung von Systemen hingegen, ebenso wie Aktivitäten des Tailoring und der Anpassung von Systemen - und somit auch End-User Development - werden mindestens von den Nutzern in einer konkreten Nutzungssituation aus einem existierenden Problem oder Bedürfnis heraus selbst initiiert, in der Regel dann auch selbständig durchgeführt, koordiniert und gesteuert.

6.3.8.4 Konsequenz

Der Anpassungsprozess wird durch Nutzer initiiert (nicht durch Experte, Programmierer, o.ä.). Dadurch kann der Nutzer den Anpassungsprozess kontrollieren, sowie den Dienst an eigene Bedürfnisse anpassen. Allerdings müssen zur erfolgreichen Anwendung die Anpassungsmöglichkeiten dem Nutzer zuerst vermittelt werden (auch der Anpassungsprozess muss erstmalig den Nutzern angeeignet werden). Zudem besitzen diese Ansätze einen komplexeren Realisierungsaufwand als bei einer "starren" vorgegebenen Struktur.

6.4 Gestaltungsprinzip

6.4.1 Direct Activation (Direkte Aktivierbarkeit)

6.4.1.1 Definition

Das Konzept der „Direkten Aktivierbarkeit“ (Wulf; On Search..; 1999) unterstützt beim Erlernen von Anpassungsfunktionen durch ein erleichtertes Finden dieser, und spielt sowohl bei Einzelplatzsystemen als auch bei Groupware eine Rolle. Die Umsetzung bezieht sich vorrangig auf das Design der Benutzungsschnittstelle, wobei insbesondere die Anordnung der Anpassungsfunktionen relativ zu den zugehörigen Nutzungsfunktionen berücksichtigt wird. Da sich diese aus dem Anwendungskontext ergibt, ist nur eine konzeptionelle Beschreibung der Anforderungen möglich, jedoch keine konkrete Vorgabe, wie die Integration in bestimmte Softwareprojekte aussehen kann.

6.4.1.2 Konzept

In verschiedenen Studien wurde untersucht, in welchen Situationen Nutzer Anpassungsfunktionen suchen. Die Untersuchungsergebnisse – Durchführung von Anpassungen bei der Einführung neuer Software (Mackay; Users and customizable Software 1990) (Tyre und Orlikowski 1994) sowie der häufige Bedarf von Anpassungsfunktionen (Page et al. 1996) während der Nutzung einer Textverarbeitung – zeigen den Bedarf von Nutzern, Anpassungen vorzunehmen, auf. Aus der Studie von Wulf (Wulf, On Search.. 1999) geht hervor, dass besonders die fehlende Kenntnis von Anpassungsfunktionen durch die Nutzer einen Hinderungsgrund zur Durchführung darstellt. Gründe für diese fehlende Kenntnis sieht Wulf darin, dass die Interaktionspunkte zu den Anpassungen nicht bekannt sind oder nicht gefunden werden.

Interaktionspunkte für Anpassungen in Anwendungen können nach ihren Bezugsmitteln gruppiert werden: Zum Einen nach dem Bezugsmittel der Räumlichen Nähe (hierzu gehören z. B. Schaltflächen und Menüs die eine Anpassungsfunktion auslösen) zum Anderen nach dem Bezugsmittel der Konsistenz der Aktivierungsmodi (wozu man z.B. Vereinbarungen über eine durchgängig realisierte identische Tastenkombination zählt). Ein Beispiel hierfür ist die Funktion der „F1-Taste“ die erwartungsgemäß die Hilfe aufruft.

Auch eine mehrstufige Aktivierung ist als konsistente Realisierung zu betrachten, wenn die Handlungsschritte bis zum Interaktionspunkt der Anpassung identisch sind. Das Kontextmenü zu einer Funktion, das in einigen Anwendungen durchgängig über die rechte Maustaste aufgerufen werden kann und der dort vorhandene Menüpunkt „Eigenschaften“ ist ein Beispiel einer mehrstufigen Aktivierung.

Einen Sonderfall stellen getriggerte Funktionen, welche nicht vom Benutzer selbst, sondern vom System angestoßen werden, dar. In diesem Fall ist kein Interaktionspunkt vorhanden, da die Ausführung einer getriggerten Funktion vom Erreichen eines vordefinierten Zustands abhängig ist. Eine Nutzung der beiden Bezugsmittel räumliche Nähe und konsistenter Aktivierungsmodus auf die Anpassung getriggerter Funktion setzt voraus, dass der Nutzer die Zustände kennt, in denen die getriggerte Funktion aktiviert wird. Ein konsistenter Aktivierungsmodus im Umfeld getriggerter Funktionen ist nicht möglich. Die Zustände können nur durch einen visuellen Bezug also mit dem Bezugsmittel der räumlichen Nähe sichtbar gemacht werden. Hat zum Beispiel ein Nutzer verschiedene Ordner für eingehende E-Mails angelegt und stellt fest, dass eine E-Mail durch seine Filter falsch einsortiert wurde, so könnte durch einen direkt erreichbaren Interaktionspunkt am Ordner eine Anpassung durchgeführt werden und so direkt die Kriterien verändert werden, nach denen die Sortierung der E-Mails erfolgt.

6.4.1.3 Konsequenz

Direkte Aktivierbarkeit bietet somit ein Erleichtertes Finden von Anpassungsfunktionen in Groupware, dadurch eine Verkürzung von Bearbeitungszeiten von Anpassungsaufgaben und eine Verbesserung des Ergebnisses dieser Anpassungen, wenn die entsprechenden Funktionen gefunden wurden.

6.4.1.4 Verwandte EUD-Methoden

Ähnliche Ansätze verfolgen die Ambient Services sowie die Direct Manipulation.

6.4.2 Affordances for change

6.4.2.1 Definition

Affordances ist die von einem Gegenstand (offensichtlich vorhandene oder tatsächlich gegebene) angebotene Gebrauchseigenschaft. Ein Stuhl hat den Angebotscharakter sich darauf zu setzen, stellen oder auch zum werfen. Ursprünglich wurde der Begriff von Gibson (Gibson 1979) benutzt. Er bezeichnet als "Affordance" den Angebotscharakter eines Objektes: Ein Objekt kommuniziert durch seine Beschaffenheit, wie es zu nutzen sei. Die Übertragung auf die Gestaltung digitaler Artefakte fand bei Von Don Norman statt.
Der Begriff Affordances for change bedeutet, das Softwareartefakt entsprechend seinem Angebotscharakter zu verändern.

6.4.2.2 Problem

Computeranwendung stellen sich dem Nutzer häufig als monolithische Applikationen dar, in dessen Ablauf er nicht eingreifen kann. Dadurch ergibt sich der Wunsch des Nutzers das System an seine Bedürfnisse anzupassen.

6.4.2.3 Lösung

Computeranwendungen sind so zu gestalten, dass sie ihre Anpassbarkeit und Erweiterbarkeit mitkommunizieren. Eine gelungene Affordance for change stellen zum Beispiel Wiki-Systeme dar. Das Anklicken einer nicht vorhandenen Website führt nicht wie gewohnt zu einem Fehler (HTML-Fehler 404), sondern öffnet eine editierbare Seite und bietet dem Nutzer an das System zu erweitern und anzupassen. Dieses Prinzip wurde auf die Erweiterbarkeit von Anwendungssystemen übertragen (Twidale et al. 1992): Falls keine oder eine schlechte Übersetzung vorliegt wird der Nutzer eingeladen, Menü-Einträge und Hilfetexte eines Anwendungssystems selbst in seine Landessprache zu übersetzen.

6.4.2.4 Bewertung

Affordances sind ein bekannter und erfolgreicher Gestaltungsansatz aus der Interfacegestaltung. Es gibt aber bisher keine genauen Gestaltungshinweise, wie sich Affordances for Change realisieren lassen kann.

6.4.2.5 Ähnliche Pattern

Ein ähnliches Prinzip verfolgt Direct Activation.

6.4.3 Gentle Slope of Complexity – Weicher Komplexitätsübergang

6.4.3.1 Definition

Die Veränderung von Informationssystemen kann alles zwischen dem Setzen eines Häkchens in einem Konfigurationsfenster und einer Neuprogrammierung der Anwendung sein. Diese Veränderungen sind immer ein Kompromiss von Mächtigkeit, Reichweite (im Sinne vom Ausmaß der Veränderung von Aussehen und Funktionalität der Software) und Komplexität

(im Sinne von zeitlichem und kognitivem Aufwand, der zur Veränderung der Software aufgebracht werden muss). Ein Ziel von EUD-Strategien ist es, eine große Bandbreite von Veränderungsmöglichkeiten vorzuhalten (von wenig Mächtigkeit/niedrige Komplexität bis große Mächtigkeit/hohe Komplexität). So wird ein weicher Anstieg der Komplexität ('Gentle Slope of Complexity') für Benutzer mit unterschiedlichen Kompetenzen möglich. Sowohl Henderson und Kyng (Henderson, Kyng 1991) als auch Morch (Morch 1997) beschreiben dreistufige Konzepte, um unterschiedliche Anpassungskomplexitäten zu beschreiben.

6.4.3.2 Problembereich

Die Strategie reagiert auf das Problem, oft mit sehr heterogen ausgebildeten Benutzerkompetenzen konfrontiert zu sein.

6.4.3.3 Lösung

Einen 'Gentle Slope of Complexity' zu erreichen, bedeutet, in den Werkzeugen möglichst viele Anpassungmöglichkeiten zu integrieren, die eine große Bandbreite bezüglich des zu ihrer Bedienung notwendigen kognitiven Aufwands abdecken.

6.4.3.4 Konsequenz

Die Grundannahme des Ansatzes, dass hohe Änderungsmächtigkeit auch immer hohe Komplexität bedeutet, ist zwar plausibel, aber tatsächlich nicht empirisch belegt. Es gibt auch Experten, die eher dahin argumentieren, Programmieren als Kulturtechnik zu begreifen (entsprechend dem Lesen), und entsprechende Kompetenzbildung bereits in den Grundschulen zu betreiben.

6.5 EUD Services

6.5.1 Community Help in Context

6.5.1.1 Definition

Bei diesem Konzept soll der Aneignungsprozess durch effektive Community-basierte Hilfesysteme unterstützen werden. Dabei bilden zwei Ansätze den Kernbereich der Arbeit: Communities und Context-Awareness.

6.5.1.2 Problembereich

Ein fundamentales Problem heutiger Hilfe-Systeme liegt im Problem der Dekontextualisierung. Dies entsteht dadurch, dass die meisten Hilfe-Systeme den Arbeitskontext in dem sich der Benutzer aktuell befindet, nicht erkennen. So muss der Benutzer zuerst nach geeigneten Hilfebeiträgen suchen, die zu seiner Situation passen.
Darüber hinaus soll dieses Konzept die Lücke zwischen Experten und Novizen schließen, um einen effizienten und effektiven Wissenstransfer zu gewährleisten. (Ackermann, Pipek, Wulf 2003)

6.5.1.3 Lösung

Communities erzielen die Möglichkeit, dass sich Benutzer gegenseitig Hilfe leisten können. Dabei erweisen sich die derzeit viel diskutieren Wiki-Systeme als nützlich, den Benutzer mit in die Hilfe zu integrieren, indem jeder Benutzer die Möglichkeit besitzt, Hilfebeiträge zu erstellen und zu editieren.

6.5.1.4 Konsequenz

Es wird ein Kontext zur aktuellen Arbeitssituation hergestellt und die Hilfe direkt in das System integriert. Weitere Vorteile sind das Prinzip Benutzer helfen Benutzern, sowie ein soziales Feedback. Wichtig ist jedoch das Erreichen der kritischen Massen an Teilnehmern und Hilfebeiträgen.

6.5.2 Configuration Sharing, gemeinsamer Konfigurationsaustausch

6.5.2.1 Definition

Configuration Sharing umfasst technische Methoden, die es Benutzern erlauben, sich Konfigurationen als 'Anpassungsobjekte' gegenseitig zur Verfügung zu stellen.

6.5.2.2 Problembereich

Grundproblem ist der Benutzerwunsch, eine bestimmte Konfiguration, zum Beispiel von der Software eines Kollegen, für die eigene Instanz dieser Software zu verwenden (z.B. eine Toolbar in MS Word, eine Farbpalette in Corel Draw, oder ein 'Widget'-Miniprogramm zur Anzeige der Wettervorhersage auf dem Desktop).

6.5.2.3 Lösung

Die Lösung ist hier, Anpassungen in geeigneter Weise zu kapseln (vgl. Objektifizierung), und die gekapselte Anpassung übertragbar zu machen.

6.5.2.4 Konsequenz

Entscheidend für die Nutzbarkeit ist, welche Granularität der Anpassung übertragbar gehalten wird (also muss z.B. eine Toolbar Button für Button übertragen werden, als ganzes, oder gar nur gemeinsam mit der kompletten Laufinstanz der Software). Der Ansatz ist eine Grundvoraussetzung für die Delegationsunterstützung.

6.5.2.5 Ähnliche EUD-Methoden

Ähnliche EUD-Methoden sind Delegation Support und Objektifizierung.

6.5.3 Negotiation Support, Aushandlungsunterstützung

6.5.3.1 Definition

Mit Negotiation Support sind alle Funktionalitäten gemeint, die die Aushandlung von Konfigurationen und Anpassungen zwischen Benutzern unterstützen können.

6.5.3.2 Problembereich

Die Verhandlungsunterstützung wird immer dann nötig, wenn sich durch Konfigurationsmaßnahmen Konflikte ergeben, da die Konfiguration der Software die Interessen anderer Benutzer verletzt. Dies wird eher in kooperativen Anwendungen der Fall sein, z.B. bei den Zugriffsrechten auf gemeinsamen Arbeitsbereichen oder in der Konfiguration der Sichtbarkeit von Aktionen (Awareness-Services).

Pipek und Kahler (Pipek, Kahler 2006) beschreiben vier Klassen von Szenarien, in denen technisch bedingt unterschiedlich starke Bindungen zwischen Benutzern entstehen, die im Falle konfliktärer Interessen bezüglich technischer Konfigurationen Aushandlungen notwendig machen können und entsprechend unterschiedliche Unterstützungskonzepte benötigen.

6.5.3.3 Lösung

Negotiation Support sollte nicht nur eine direkte Unterstützung der Kommunikation umfassen, sondern auch auf den Diskussionsgegenstand, die Technik und ihre Konfiguration, eingehen können (z.B. mit geeigneten Visualisierungen wie Screenshots). Unter Umständen können auch Funktionen aus dem Bereich entscheidungsunterstützender Systeme sinnvoll sein.

6.5.3.4 Konsequenz

Aushandlungsprozesse kosten Zeit und können deshalb, wenn sie zu lange dauern oder zu häufig auftreten, den Nutzwert kooperativer Systeme herabsetzen.
Es gibt bisher nur wenige empirische Auswertungen solcher Verhandlungsprozesse (z.B. Pipek, Negotiating Infrastructure 2005)

6.5.3.5 Ähnliche EUD-Methoden

Ein ähnliches Prinzip verfolgt der User Community Support.

6.5.4 User Community Support, Unterstützung von Benutzergemeinschaften

Dieser Ansatz beschreibt eine ganze Klasse von EUD-Funktionen, denen gemeinsam ist, dass sie die Benutzer einer Technologie oder eines Werkzeugs als Interessensgemeinschaft verstehen und unterstützen.

6.5.4.1 Definition

Die 'User Community' eines Werkzeugs oder einer Technologie ist die Gesamtheit ihrer Benutzer. Diese werden nach dem Konzept der 'Communities of Practice' (Wenger, Communities of Practice 2000) als 'Community of Technology Practice' verstanden. Zunächst ist damit die Möglichkeit zur Kommunikation einzelner Benutzer untereinander und der Gesamtgruppe verbunden. Darüber hinaus gilt es aber auch, Benutzer im Sinne der Ideen 'legitimer peripherer Partizipation' (Wenger, Communities of Practice 2000) an der Praxis erfahrener Benutzer teilnehmen zu lassen. Dies kann neben direkter Kommunikation (vgl.

Negotiation Support, Demonstration Support) auch andere Nutzungsvisualisierungen umfassen (vgl. Observation Support), oder auch Delegationsbeziehungen unterstützen (vgl. Delegation Support).

Pipek (Pipek, Appropriation Support 2005) hat User Communities vor allem bezüglich der Unterstützung von Benutzern zur Erfindung, Realisierung und dem Transfer von Nutzungen (der Aneignungsunterstützung) untersucht.

6.5.4.2 Problembereich

Das Pattern basiert auf der Beobachtung, das Benutzer oft untereinander besser verstehen, wie Nutzungsprobleme zu beurteilen sind, und sich bei ähnlicher Nutzungspraxis auch besser vermitteln können, wie Probleme mit der Technikkonfiguration gelöst werden können.

6.5.4.3 Lösung

Das Pattern wird durch eine Reihe von Funktionalitäten zur Unterstützung von technikorientierter Kommunikation unter Benutzern implementiert, die in den oben angeführten Support-Pattern manifestiert sind.

6.5.4.4 Konsequenz

Ein großes Problem von User Communities ist die Frage der Beteiligung. Die Auseinandersetzung mit Technik, ihrer Nutzung und ihrer Konfiguration ist für viele Benutzer nicht primäre Arbeitsaufgabe. Oft wird nur bei dem Auftreten eines Nutzungsproblems in der eigenen Arbeit der Sprung auf die Metaebene der Technikdiskussion vollzogen, und dann auch zielorientiert auf die Lösung dieses Problems hingearbeitet. Deshalb ist es wichtig, Maßnahmen zu erarbeiten, die Benutzern unter geringstmöglichem Aufwand ermöglichen, von der Community zu profitieren.

Die Teilnahme oder Beobachtung einer User Community kann auch für die Designer der Software wertvolle Impulse geben. Es kann sich hier eine weitere wichtige Informationsquelle für das Requirements-Engineering ergeben.

6.5.4.5 Ähnliche EUD-Methoden

Ähnliche EUD-Methoden sind Negotiation Support, Demonstration Support, Observation Support und Delegation Support.

6.5.5 Configuration Browsing, Durchsuchen von Konfigurationssammlungen

6.5.5.1 Definition

Configuration Browsing beschreibt die Möglichkeit, in einem Repository alternative Softwarekonfigurationen zu recherchieren und gegebenenfalls für sich zu verwenden. Im Unterschied zum Configuration Sharing sind hier der Benutzer, der eine bestimmte Konfiguration erstellt hat, und der, der sie abruft, voneinander entkoppelt, unter Umständen werden sie nie direkt miteinander kommunizieren.

6.5.5.2 Problembereich

Eine derartige Repository-orientierte Lösung geht vor allen Dingen das Problem an, das oft eine Übersicht über vorhandene und sinnvolle Konfigurationen schwer zu bekommen ist.
Kahler (Kahler, More than WORDs 2001) stellte ein entsprechendes Konzept für Anpassungen von Microsoft Word vor.

6.5.5.3 Lösung

Ein Repository für Konfigurationen, unter Umständen noch geeignet strukturiert und kommentiert, kann die Übersicht über alternative Konfigurationen ermöglichen.

6.5.5.4 Konsequenz

Der fehlende Kontakt zum Ersteller einer Konfiguration macht eine Annotierung der im Repository befindlichen Konfigurationen ratsam. Unter Umständen kann man auch für Rückfragen einen Feedback-Kanal zur Verfügung stellen.
Bei der Architektur dieser Repositories kann man sowohl an zentralistische Client-Server-Architekturen denken als auch an Peer-to-Peer-Systeme. Generell kann man auch überlegen, Beziehungen zwischen einzelnen Konfigurationen zu visualisieren, z.B. wenn eine Konfiguration eine Spezialisierung einer anderen ist, oder Konfigurationen 'verwandt' sind, d.h. sich nur unwesentlich unterscheiden. Generell muss, wie bei allen anderen Repository-orientierten Ansätzen, das Problem der Wartung diskutiert werden.

6.5.5.5 Ähnliche EUD-Methoden

Dazu verwandt ist das Configuration Sharing.

6.5.6 Delegation Support, Delegationsunterstützung

6.5.6.1 Definition

Unter Delegation Support sind Funktionen zu verstehen, die es den Benutzern erlauben, Delegationsbeziehungen bei der Konfiguration von Software einzurichten und zu managen. Delegieren heißt in diesem Fall, dass die eigentliche technische Durchführung einer Konfigurationsänderung nicht vom Endbenutzer, sondern einer Vertrauensperson durchgeführt wird. Managen umfasst dabei die Verfolgung der Aktivitäten sowohl beim Delegierenden als auch beim Ausführenden und kann auch als Prozess abgebildet werden. Delegation Support kann Teil einer User Community Support-Strategie sein. Die Vertrauensperson/der Ausführende wird dabei überwiegend aus dem direkten Umfeld des Endbenutzers kommen, es kann sich aber auch um einen Vertreter der Entwickler des betroffenen Werkzeugs handeln.

6.5.6.2 Problembereich

Delegation Support geht das Problem an, dass sich auch trotz größter Anstrengungen, Werkzeuge so einfach anpassbar wie möglich zu machen, immer Benutzer finden werden, die sich nicht mit der technischen Konfiguration ihrer Arbeitswelt auseinandersetzen können oder wollen. Diesen wird so die Möglichkeit gegeben, ihre Konfigurationsvorstellungen von anderen Benutzern realisieren zu lassen. Die Möglichkeit zum Configuration Sharing ist dafür eine Voraussetzung.

6.5.6.3 Lösung

Delegation Support kann alles von der ersten Anfrage nach Unterstützung bis hin zum gemeinsamen Feintuning von Konfigurationen zwischen Expertenbenutzer und delegierendem Benutzer sein. Neben der Kommunikation und der Prozessabwicklung des Delegationsauftrages kann auch noch Funktionalität sinnvoll sein, durch die der Expertennutzer ein Bild vom technischen Kontext (z.B. verwendete Rechner, erreichbare Rechnernetze, parallel laufende Anwendungen, Betriebssystem und allgemeine Softwareumgebung des zu konfigurierenden Tools) der anzupassenden Anwendung bekommt, damit dieser nicht von einem unkundigen Benutzer selbst beschrieben werden muss. Dem Expertenbenutzer kann auch noch Funktionalität zum Managen der Anfragen (Priorisierung, Kommunikation, etc.) an die Hand gegeben werden.

6.5.6.4 Konsequenz

Da Benutzer möglicherweise Schwierigkeiten haben, ihre Wünsche klar zu artikulieren, und auch neue technische Optionen erst im Verlaufe der Nutzung entdecken, muss mit der Notwendigkeit eines wiederholten oder iterativen Vorgehens gerechnet werden.

6.5.6.5 Ähnliche EUD-Methoden

Configuration Sharing und User Community Support.

6.5.7 Demonstration Support, Demonstrationsunterstützung

6.5.7.1 Definition

Unter Demonstration Support versteht man Funktionen und Maßnahmen, die es Benutzern erlauben, untereinander Nutzungen einer Technologie zu demonstrieren. Dabei kann es sich um Vor-Ort-Demonstrationen handeln (hier ist es z.B. sinnvoll, auf eine sinnvolle Namensgebung von Funktionen und Interfaceobjekten zu achten, um die Kommunikation über die Technik zu erleichtern), oder um mediierte Demonstrationen (demonstrierender Benutzer und Zuschauer sind an verschiedenen Orten über das Internet miteinander verbunden), die entweder synchron (demonstrierender Benutzer und Zuschauer agieren gleichzeitig) oder asynchron (demonstrierender Benutzer und Zuschauer agieren zeitversetzt) ablaufen können.

6.5.7.2 Problembereich

Die Methode geht das Problem an, dass es sinnvoll ist, technikbezogene Erklärungen über Demonstrationen visuell zu unterstützen.

6.5.7.3 Lösung

Während für die Vor-Ort-Vorführung technisch wohl eher eine hinreichende Visualisierungsfläche das größte Problem darstellt, kann man bei mediierten Demonstrationen unterschiedliche Ansätze diskutieren. Z.B. können Demonstrationen auf der Basis von Application Sharing Lösungen funktionieren, oder durch das Werkzeug selbst unterstützt werden. Bei visueller Unterstützung und Kommunikationsunterstützung ist es sinnvoll, dies gut zu integrieren. Für den asynchronen Fall gibt es bereits Programme, mit denen man Softwarevorführungen erstellen, abspeichern und versenden kann.

6.5.7.4 Konsequenz

Mit einer derartigen Unterstützung gibt ein demonstrierender Benutzer unter Umständen auch privatheitsrelevante Informationen preis, es ist deshalb ratsam, dem Demonstrierenden alle Kontrollmöglichkeiten zu lassen.

6.5.7.5 Ähnliche EUD-Methoden

User Community Support und Observation Support verfolgen ähnliche Ansätze.

6.5.8 Observation Support, Beobachtungsunterstützung

6.5.8.1 Definition

Unter Observation Support versteht man Maßnahmen, die es Benutzern erlauben, andere Benutzer bei der Softwarenutzung zu beobachten, um daraus zu lernen. Es ist damit Teil einer User Community Support-Strategie. Dabei kann man grundsätzlich unterscheiden, ob eine bestimmte Nutzung eines Benutzers beobachtet wird, oder ob akkumulierte Nutzungen einer Gruppe von Benutzern (oder aller Benutzer) beobachtet werden können. Bei der Einzelbeobachtung besteht klar die Gefahr der Verletzung der Privatsphäre, weshalb

entsprechende Maßnahmen implementiert sein sollten, die einen beobachteten Benutzer auf die Beobachtung aufmerksam machen oder ihm die Kontrolle darüber lassen (Übergang zum Demonstration Support). Bei der Visualisierung von akkumulierter Nutzung besteht diese Gefahr nicht, aber es bleibt unter Umständen schwerer nachzuvollziehen, welche Nutzungen zusammenhängen. Das Pattern bildet eine der Voraussetzungen für den Recommendation Support.

6.5.8.2 Problembereich

Diese EUD-Methode implementiert eine wichtige Funktionalität um legitime periphere Partizipation für den User Community Support zu realisieren.

6.5.8.3 Lösung

Im Gegensatz zum Demonstration Support ist hier nur der mediierte Fall der Beobachtung relevant. Konzeptuell kann man sich z.B. über die notwendige Homogenität in der Benutzergruppe und ihre Größe Gedanken machen, und über die Art und Weise, vielleicht auch Kontextinformationen der Nutzung (Organisationale Rolle, Vorbildung, etc.) mit berücksichtigen zu können.

6.5.8.4 Konsequenz

Wichtig für die Akzeptanz dieser Systeme dürfte hier die Wahrung der Privatsphäre sein. In Deutschland könnten Funktionalitäten dieser Art arbeitsrechtlichen Beschränkungen unterliegen.

6.5.8.5 Ähnliche EUD-Methoden

User Community Support, Demonstration Support und Recommendation Support sind ähnliche EUD-Methoden.

6.5.9 Recommendation Support

6.5.9.1 Definition

Recommender-Systeme sind Programme, die den Nutzern so genannter Items empfehlen, an denen Nutzer Interesse haben könnten. Damit das System dem Nutzer neue Items empfehlen kann, müssen Informationen über Interaktionsverhalten bzw. Interessen in Nutzer-Profilen abgelegt sein.

6.5.9.2 Problembereich

In vielen Anwendungsdomänen stehen wir einer Informationsüberflutung gegenüber. Bei einer endlichen Menge von Angeboten sind wir meist nur an einer bestimmten Teilmenge entsprechend den persönlichen Neigungen interessiert. Wie finden wir aber, was uns interessieren könnte?

6.5.9.3 Lösung

Man zeigt dem Nutzer, was ihn interessieren könnte. Ist im Prinzip eine Vorauswahl aus der Gesamtmenge entsprechend einem oder mehreren Nutzerprofilen.

Das Ziel eines Recommender-Systems ist i.A. die Vorhersage (Prediction) eines Wertes (Wahrscheinlichkeitswert), mit dem Nutzer bestimmte Elemente (Items) mögen oder nicht. Je nach Anwendungsfall kann es sich bei Items um Internet-Seiten, Nachrichten, Produkte wie Bücher, CDs etc. oder Fernsehsendungen bzw. Videos handeln.

Recommender Techniken lassen sich allgemein in zwei Klassen unterscheiden:

- **Content-Based Recommendations** (Inhaltsbasierte Empfehlung) bzw. Case-Based Recommendations: Empfehlungen dieser Kategorie werden aufgrund von Nutzungsverhalten und Bewertungen der Nutzer in der Vergangenheit erstellt. Was den Nutzer damals interessierte, könnte auch zukünftig wieder für ihn von Interesse sein.
- **Collaborative Filtering** (Kollaborative Filterung) oder auch allgemein als Social Filtering bezeichnet: Nutzer mit ähnlichen Profilen interessieren sich mit einer gewissen Wahrscheinlichkeit auch für ähnliche Inhalte. So können einem Nutzer auch Empfehlungen anderer Nutzer mit ähnlichem Nutzerprofil hilfreich sein. Empfehlungen werden also, im Gegensatz zu der ersten Techniken, aufgrund einer Gruppe von Nutzern erstellt. Die Qualität der Empfehlungen steigt mit der wachsenden Zahl an Personen, die an einem derartigen System angeschlossen sind.

Jedes der vorgestellten Verfahren besitzt sowohl Vorteile als auch Nachteile. Systeme in denen beide Techniken zur Anwendung kommen werden Hybrid-Systemen genannt.

6.5.9.4 Konsequenz

Recommender-Systeme sind mittlerweile ziemlich gut erforscht und die entsprechenden Algorithmen kommen ziemlich gut an das tatsächliche Interesse des Nutzers heran. Diese Systeme bilden die Grundlage für Personalisierung und damit für individuell zugeschnittene Angebote.
Es sind aber nicht alle Items für Empfehlung geeignet. Ein Beispiel dafür wäre Medizin. Weil die Medizin bei vielen anderen Personen geholfen hat, muss es nicht unbedingt auch für mich die beste Medizin sein, da ich vielleicht Unempfindlichkeit, Allergie, etc. habe, was bei der Empfehlung nicht berücksichtigt wird. Bei Büchern, CDs, Videos, etc. ist dies dagegen weniger kritisch.

6.5.10 Objektifizierung

6.5.10.1 Definition

Diese EUD-Methode befasst sich mit einer Vergegenständlichung von Anpassungen, so dass diese als eine Einheit behandelt werden können. Durch die Objektifizierung ergibt sich die Möglichkeit zum Austausch, zur Archivierung, aber auch zum Diskurs über Anpassungen.

6.5.10.2 Einordnung

Objektifizierung ist dabei als eine unterstützende Technik anzusehen. Sie zielt nicht auf die eigentliche Anpassung ab, aber sie unterstützt das Verwalten und die gemeinsame Nutzen von Anpassungen. Sie kann beispielsweise eingesetzt werden um Undo-Funktion oder auch Explorations Umgebungen zu realisieren.

6.5.10.3 Problembereich

Systeme werden nicht nur angepasst, vielmehr werden Anpassungen mit Kollegen geteilt und darüber diskutiert.

6.5.10.4 Lösung

Die Grundlage jeglicher Objektifizierung besteht darin, dass Anpassungsdirektiven isoliert in einer Datei oder ähnlichem gespeichert werden. Dabei kann es sich um eine Sammlung von Parameterbelegungen oder auch um Skripte handeln, die Systemanpassungen vornehmen oder selbst Funktionserweiterungen des Systems darstellen.

Bezüglich der Lösungsansätze ist zu differieren, in welcher Form der Nutzer beim Export von Teilen der Anpassung (nur Teile der Systemanpassung, Erkennung von Abhängigkeiten des Anpassung, etc.) unterstützt wird und wie Objektifizierungsmöglichkeiten in die Applikation integriert sind.

6.5.10.5 Konsequenz

Objektifizierung vereinfacht die Wiederverwendung von Anpassungen und den Austausch und Diskurs über Anpassungen. Dabei besteht jedoch immer die Gefahr, dass – je nach gewähltem Format – Objektifizierungen unerwünschterweise auch Anpassungen des Systems beinhalten.
Ein weiteres Problem betrifft die Isolierung: Objektifizierungen beinhalten nicht das Gesamtsystem, sondern nur ein Delta, welches die Änderungen enthält, die jedoch möglicherweise nicht komplett sind und bei anderen Nutzern daher nicht ausgeführt werden können.

6.5.11 Community-Based Interface Translation

6.5.11.1 Definition

Bei der Gestaltung von Software findet meist ein Fokus auf eine kleine, klar abgegrenzte Gruppe von Nutzern statt, da dies augenscheinlich den größten Erfolg verspricht. Oft ist es aber erstrebenswert die Software nachträglich auch auf andere Gruppen auszudehnen. Dies erfordert eine Erweiterung und Anpassung sowohl des Interfaces wie auch der Funktionalität, bei der es sinnvoll und erstrebenswert ist auch den Endbenutzer in den Anpassungsprozess einzubinden.

Anpassungen können auf verschiedenen Ebenen durchgeführt werden: Auf der einen Seite die Erweiterung eines Programms durch erfahrene Programmierer, auf der anderen Seite durch meist unerfahrene Endbenutzer. Dazu wird eine Bandbreite von Einstellungen angeboten, anhand der ein Endbenutzer mit geringem Aufwand und geringen Kenntnissen die gewünschten Funktionen festlegen kann.

6.5.11.2 Problembereich

Ein Bereich ist das Bedürfnis von Benutzergruppen ein Interface in einer bestimmten Sprache zu Nutzen. Wenn ein Interface nicht in der Muttersprache des Nutzers existiert, erschwert sich die Nutzung des System stark, da nicht immer die Bedeutung aller Bezeichnungen auf dem Interface komplett korrekt erkannt werden.
Dieses Problem ergibt sich vor allem bei weltweit über das Internet genutzten Programmen. Hier ist es elementar wichtig, dass sich die für eine globale Nutzerschaft gebauten Produkte bezüglich der Sprache personalisieren lassen.

6.5.11.3 Lösung

Nach Jones et al. (Wikifying your Interface; 2006) beobachtet der Ansatz der Community-Based Interface Translation wie die Nutzer einen Zugang zu einer Sprache finden und wie Gruppenbasierte Methoden dabei helfen können diesen zu unterstützen. Als Grundvoraussetzung müssen die Interfaces der Programme flexibel genug sein Textbearbeitung zum Zweck der Übersetzung zu unterstützen.
Dies geht über einfaches Auswählen einer vordefinierten Sprache hinaus, und umfasst die Möglichkeit Sprachen hinzuzufügen und zu bearbeiten, sowie die Erstellung einer personalisierten Übersetzung des Interfaces.
Es lassen sich dabei drei Arten unterscheiden, eine Interface-Übersetzung zu erstellen: Automatische, Manuelle und Gruppenbasierte Übersetzung. Jede dieser Möglichkeiten bietet dabei spezifischen Vor- und Nachteile.

6.5.11.4 Konsequenz

Durch Adaptieren und Anpassen der Interfaces ist einer breiteren Nutzerschaft ein einfacherer Zugang zu dem System möglich. Zusätzlich wird durch den offenen und verteilten Ansatz der Aufwand verteilt und auch technisch weniger versierte Nutzer werden motiviert.
Schwierig ist im Moment noch die Verknüpfung von Bearbeitungssystemen zum Übersetzen (z.B. Wiki) und des Interfaces welches adaptiert werden soll.

6.5.12 Disclosure Principle, Nachvollziehbarkeit

6.5.12.1 Problembereich

Ein Problem bei EUD stellt der Übergang von der Nutzung zur Anpassung dar. Insbesondere durch GUI-Konzepte wurde die Trennung zwischen der Bedienungssprache und der Programmierungssprache vergrößert. Mit diesem Problem beschäftigt sich der Ansatz der Disclosure Principles.

6.5.12.2 Lösung

Bei der Nutzung werden in das Benutzerinterface zusätzliche Informationen eingefügt, die, besser als die GUI, die zugrunde liegenden Ablaufmechanismus sowie die Programmiermöglichkeit derselben offenbaren und so Möglichkeiten zu Anpassung offensichtlicher machen. Dies erleichtert dem Nutzer das Verständnis (erhöhte Nachvollziehbarkeit) und den Übergang von der Nutzung zur Anpassung.

6.5.12.3 Konsequenz

Durch die Nachvollziehbarkeit lernt der Nutzer während der Nutzung die Ablaufmechanismen kennen und bekommt dadurch einen leichteren Zugang zur Anpassbarkeit. Problematisch kann dabei sein, dass der Nutzer diese zusätzlichen Informationen auch dann aufgebürdet bekommt, wenn er keine Anpassungen vornimmt und vornehmen will.

6.5.12.4 Ähnliche EUD-Methoden

Ähnliche Methoden sind Gentle Slope of Complexity sowie Customization Gulf.

7 Potentiale und Erwartungen von/an Demonstratoren

Das Kapitel 6 gibt einen Überblick über die im Projekt identifizierten EUD-Patterns. Ein Pattern steht dabei allgemein für eine Lösungsvorschrift für wiederkehrende Probleme. Es ist daher nicht verwunderlich, dass sich in der Literatur und in der Praxis verschiedene Realisierungen finden lassen, die ein entsprechendes EUD-Pattern umsetzen bzw. anwenden. Da solche Ansätze ein klares Bild von konkreten Anwendungsfällen und deren Lösungspotential aufzeigen, erfolgte eine gezielte Sammlung von relevanten Demonstratoren zu den identifizierten Patterns. Diese Demonstratorensammlung vermittelt einen Überblick darüber, in welchen Domänen heute EUD-Patterns bereits zum Einsatz kommen.

Die Patternsammlung als fundamentale Basis und die Demonstratorenübersicht als Beispielüberblick, bilden die Grundlage für die im EUDISMES-Projekt anzuwendenden Lösungsstrategien. Geeignete EUD-Maßnahmen werden bei der Umsetzung von Aneignungs- und Unterstützungsmaßnahmen in den beteiligten Unternehmen eine Schlüsselrolle einnehmen. Daher ist das Wissen über eine hohe Zahl von EUD-Methoden (Gesamtheit), deren Lösungspotential (Generalität und Konzeptualisierung) sowie deren konkrete Realisierung in der Praxis (Umsetzung im Kontext) essentiell. Die Demonstratoren vermitteln in diesem Bezug die Umsetzung von EUD-Maßnahmen in einem konkreten Anwendungskontext. Auch wenn die identifizierten Kontexte der Demonstratoren sich nicht direkt in Kontexte im projektspezifischen Anwendungsfall von kleinen und mittelständischen Unternehmen (KMU) übertragen lassen, so wird doch deutlich, wo Stärken, Schwächen und Potentiale eines Patterns im Problemlösungsprozess liegen.

8 Übersicht Demonstratoren

8.1 Architekturkonzept

8.1.1 Web Services

Ein vielfach angeführtes Beispiel für Web Services ist ein Reisebuchungssystem. Zu dessen Realisierung, könnte ein Unternehmen verschiedene Dienste wie Flugreservierung, Hotelreservierung, Mietwagenbuchung etc. in einer eigenen Anwendung kombinieren. Dadurch entstünde ein neuer Web Service, der als eigenständiges System angeboten werden könnte.

Es gibt zum Beispiel seit 2002 von Google einen Web Service, der durch seine Funktionalität die gleichen Möglichkeiten der Suchmaschine anbietet, wie die Benutzerschnittstelle auf der Webseite selbst. Programme können nun durch das Ansprechen der Schnittstelle direkt nach Informationen im Internet suchen, erhalten über die Schnittstelle die Ergebnisdaten und können diese für ihre eigenen Aufgaben verwenden. Das Parsen der Google-Webseite ist dazu keine auch nur annähernd gleichwertige Alternative.

Ein weiteres Beispiel ist die Interaktion von Fluggesellschaften und Reisebüros. Die Fluggesellschaften stellen Möglichkeiten zum Nachschlagen bzw. Buchen von Flügen über einen Web Service bereit. Die Reisebüros bieten auf ihrer Webpräsenz Flüge verschiedener Fluggesellschaften an, von denen die Reisebüros in Echtzeit über UDDI erfahren. Der Kunde kann auf der Webpräsenz des Reisebüros nun zentral Preise und Termine verschiedener Flüge vergleichen und direkt buchen.

8.1.2 Makroprogrammierung

Ein Beispiel für Makroprogrammierung, ist die Aufzeichnung von Textbearbeitung und Arbeitsschritten in Microsoft Word. Diese lassen sich einfach aufzeichnen und werden als Visual Basic-Code gespeichert. Bei Bedarf lässt sich dieser dann mit Programmierkenntnissen weiter bearbeiten und verfeinern. Optional lassen sich die Makros dann über Tastenkommandos oder die Symbolleisten abrufen.

8.2 Programmiermethode

8.2.1 Natural Programming – Natürliche Programmierung

Ein Beispiel für Natural Programming ist Asset-Oriented-Modelling (AOM), bei dem KLEEN Talk eingesetzt wird. AOM ist eine Erweiterung für das Entity-Relation-Modelling. Die Haupteigenschaft liegt in der Trennung von Entitäten und Beziehungen. Durch Aufnahme natürlich sprachlicher Elemente und natürlichem Satzbau, ist es dem englischsprachigen Anwender möglich, Datenbank-Schemata zu definieren.

8.2.2 Microsoft Workflow Foundation

Die Microsoft Workflow Foundation ist ein Beispiel für Natural Programming. Sie bietet eine Infrastruktur zur einfacheren Entwicklung von Workflow-Anwendungen, sowohl in geschäftlicher als auch technischer Hinsicht, aber auch für dokument- und webbasierte Workflows. Sie umfasst ein Programmier-Modell, eine Ausführungsmaschine und Werkzeuge, um Geschäftsprozessanwendungen in Windows zu realisieren. Es besteht die Möglichkeit, Geschäftsprozesse flexibel anzupassen. Das Modellieren, Entwickeln und Debuggen kann mit dem Designer für Visual Studio über eine intuitive diagrammatische Schnittstelle geschehen (Modellierung mittels Fluss- und Zustandsdiagrammen). Beispielsweise zeichnet der Anwender Geschäftsprozesse mit Hilfe vorgegebener Aktivitäten,

die er dem Prozess entsprechend verbindet; dabei sind auch konditionale Verzweigungen einfach realisierbar.

Somit richtet sich das Werkzeug vor allem an fachlich versierte, aber technisch unerfahrene Endbenutzer. Der große Schritt gegenüber einer „from scratch“-Programmierung besteht außerdem darin, dass bereits eine große Menge domänenspezifisches Wissen zur Verfügung steht und vom Anwender visuell abgerufen werden kann. Zudem können stets „ad hoc“-Aktivitäten hinzugefügt werden.

Konsequenz
Workflow-Management-Systeme ermöglichen es dem Endbenutzer, in der für ihn gewohnten "Sprache" zu sprechen und erlauben es ihm dadurch, Anpassungen an Systeme leicht vornehmen zu können. Zudem ermöglicht es ein einfaches Hinzufügen von ad hoc-Akivitäten. Allerdings ist unter Umständen manuelles Kodieren immer noch erforderlich.

8.2.3 Visuelle Programmierung

Beispiele für visuelle Programmierung sind:

- **LabView** ist eine graphische Programmiersprache von National Instruments. Das Akronym steht für "Laboratory Virtual Instrument Engineering Workbench". Haupt-Anwendungsgebiete von LabVIEW sind die Mess- und Automatisierungstechnik. Die Programmierung folgt dem Datenfluss-Modell, wodurch sich die Sprache zur Datenerfassung und -verarbeitung anbietet. LabView-Programme werden als Virtuelle Instrumente oder einfach VIs bezeichnet. Sie bestehen aus zwei Komponenten: das Frontpanel enthält die Benutzerschnittstelle, das Blockdiagramm den graphischen Programmcode. Dieser wird nicht von einem Interpreter abgearbeitet, sondern kompiliert. Dadurch ist die Performanz vergleichbar mit der anderer Hochsprachen. Derzeit liegt LabView in der Version 8.20 vor.

- **COVISE** ist ein flexibles Visualisierungssystem, mit dem die Visualisierung wissenschaftlicher Daten auf einfache Weise, graphisch programmiert werden kann. COVISE bietet eine umfangreiche Bibliothek von Programmteilen an, den sogenannten Modulen, die zu einem Datenflussnetz interaktiv verbunden werden können. Über eine Programmierschnittstelle (API) können neue Module in das Visualisierungssystem integriert und für die graphische Programmierung bereitgestellt werden.

- Der **IBM Data Explorer** ist ein Programm-Paket zur Visualisierung wissenschaftlicher Daten. Es hält fertige Programme/Module zur Transformation von Daten, Downsizing, Arithmetik, 3-D-Darstellung von Daten, z.B. Berechnung von Isoflächen, Volume-Rendering, Stromlinien, Emission von Testteilchen, X/Y-Plots, 3-D-Achsen, Achsenbeschriftung, Colorbars, Bildbeschriftung, Transformation von dargestellten Objekten, Manipulation von Lichtquellen, Kameraeinstellungen, Layoutgestaltung, Animation und Debugging bereit.

- **AVS Express** ist ein Programm-Paket zur Visualisierung von Daten. Es beinhaltet fortgeschrittene Visualisierungen wie z.B. Vektorfeldstromlinien, ein generalisiertes "Glyph"-System, Module zum Erstellen von Isolinien/Isoflächen durch Verfolgen von Startpunkten, etc. Bei der Datenbearbeitung bietet das Programm unter anderem 2D/3D Interpolationen höherer Ordnung und Netzoptimierung. Unterstützt werden irreguläre Geometrien, das Transformieren von (regulären und irregulären) Gittern und Daten in Form von z.B. Polarkoordinaten, Zylinderkoordinaten und sphärischen

Koordinaten. Das Programm ist außerdem mit Modul-Bibliotheken und Makros zur 3D-Volumenbildverarbeitung ausgestattet.

- **Vis5D** ist ein Softwaresystem zur Visualisierung von Daten aus numerischen Wettermodellen oder ähnlichen Quellen. Das Vis5D-System enthält das vis5d Visualisierungsprogramm, verschiedene Programme zum Management und zur Analyse von 5-dimensionalen Datengittern sowie Anweisungen und Beispiele zur Konvertierung von Daten in das Vis5D-Format. Ferner sind Beispieldaten des LAMPS-Modells und von Bob Schlesinger's thunderstorm-model, Topographien und Landkarten enthalten. Wegen Zeitmangels haben die Programmierer von Vis5D das Programm an ein Open-Source-Projekt übergeben. Dieses führt die Entwicklung unter dem Namen Vis5D+ fort.

8.2.3.1 Business Intelligence Anwendungen mit SAP NetWeaver Visual Composer

SAP NetWeaver Visual Composer ist ein Beispiel für Visuelle Programmierung. Endbenutzer gerechte Business Intelligence Anwendungen erlauben es ihm, ohne Eingriffe in das System, detailliert auf seine Bedürfnisse angepasste Berichte zu verfassen oder Unternehmensdaten nutzerangepasst anzufragen und aufzubereiten.

In SAP NetWeaver ist ein Visual Composer integriert, der es ermöglicht, modellbasierte Anwendungen flexibel und ohne manuelle Programmierung zu entwickeln. Während sich der Business-Explorer auf Business-Analysten fokussiert, die ausschließlich in einem Business Intelligence (BI) Kontext arbeiten, ist der Visual Composer ein Business-Analysten-Tool, mit einem weiteren Skopus. Man kann neben BI-Inhalten auch andere (SAP und Nicht-SAP) Datenquellen einbeziehen, sowie eine Integration von SAP ERP Daten durchführen.

Zunächst können mit dem SAP NetWeaver Visual Composer, über eine einfache und intuitive Notation in einem Storyboard, Modelle aufgebaut werden. Die Modelle beschreiben die Datensicht, die Benutzeroberflächensicht oder eine Kombination aus beidem. Hierbei ist eine effektive Unterstützung durch vorgegebene Muster gewährleistet. Es ist möglich, die Komposition per intuitiven Drag & Drop zu erstellen. Über den Zwischenschritt eines Modell-Repositorys, in dem das Modell in einer internen Modellierungssprache abgelegt wird und mit Hilfe von Plugins kann automatisch ausführbarer Code generiert werden.

Die Entwicklungsumgebung ermöglicht es dem Endbenutzer, die SAP BI-Software ohne besondere Entwicklungskenntnisse detailliert zu konfigurieren. Es werden hierzu verschiedene Visualisierungsmöglichkeiten, wie Tabellen, Diagramme oder Formularfelder bereitgestellt, die sich über Ankreuzfelder oder Menüs auswählen lassen. Das Ergebnis ist eine angepasste Software oder eine Portalanwendung, die auf die Anforderungen des Endbenutzers ausgerichtete Analysen bereitstellt.

Konsequenz

Business Intelligence Anwendungen nehmen dem Endbenutzer das aufwändige Programmieren von Berichten oder Anfragen nach wichtigen Unternehmensdaten ab, indem sie visuelles Modellieren in dieser Domäne unterstützen. Dabei ist der Abstraktionsgrad (angebotene Muster), trotz graphischer Unterstützung bei bestehenden Anwendungen, oft noch nicht hoch genug.

8.2.3.2 SAP Quick Viewer

SAP Quick Viewer ist ein weiteres Beispiel für Visuelle Programmierung. Mit SAP Query und dem Quick Viewer können Berichte definiert werden. Dabei ist es nicht erforderlich zu programmieren.

Um einen Report zu definieren, werden nur einzelne Textteile wie z.B. der Titel eingegeben und Felder und Optionen markiert und geordnet (Quick Views), die den Aufbau des Reports bestimmen sollen. Die Listen können dann per Drag & Drop editiert werden. Der Quick Viewer eignet sich für Anfänger oder für den gelegentlichen Gebrauch.
Die Reports, die mit dem Quick Viewer erstellt worden sind, können an externe Programme (z.B. Standard-Office-Programme) weitergeleitet werden.

Konsequenz
Der Quick Viewer ermöglicht die visuelle Programmierung von Berichten, hat dadurch aber eine beschränkte Ausdrucksfähigkeit.

8.2.3.3 LEGO Mindstorms

Unter anderem ein Beispiel für visuelle und haptische Programmierung ist Lego Mindstorms. Dies ist der Name einer Produktserie der LEGO Company, die einen programmierbaren Legostein (RCX genannt), sowie Elektromotoren, Sensoren und LEGO Technik-Teile (Zahnräder, Achsen, Lochbalken, Pneumatik Teile usw.) enthält, um Roboter und andere autonome und interaktive Systeme zu konstruieren und zu programmieren.

Für die Entwicklung von Embedded Systems, werden hohe Programmierkenntnisse vorausgesetzt. Dem gegenüber stehen hohe Entwicklungskosten. Die Hauptproblematik besteht in der Vermittlung von Expertise in der Entwicklung von Embedded Systems, beginnend im Kindesalter.

Obwohl es ein technisches Spielzeug ist, kann es auch (wie von LEGO und MIT ursprünglich auch geplant) als Lehrmittel eingesetzt werden; es dient als gutes Beispiel für ein Embedded System, mit computergesteuerten elektromechanischen Teilen. Beinahe alle Arten von Embedded Systems, vom Aufzug bis hin zu Industrierobotern, können mit Mindstorms nachgebaut werden.

Der programmierbare Legostein RCX besitzt einen Renesas H8/300 Microcontroller als CPU. Er wird programmiert, indem ein in einer der diversen Programmiersprachen geschriebenes Programm, vom PC zur CPU des RCX mit Hilfe einer IR Schnittstelle übertragen wird. Nachdem sich das Programm auf der CPU befindet, kann der mit einem RCX gebauten Mindstormsroboter völlig autonom handeln und auf äußere und innere Ereignisse reagieren, entsprechend den Programmieranweisungen. Des Weiteren können zwei oder mehr RCX miteinander über die IR Schnittstelle kommunizieren, was Wettbewerbe und Kooperationen ermöglicht. Zusätzlich besitzt der RCX noch drei Motorausgänge sowie drei Sensoreingänge.

Konsequenz
Es handelt sich dabei um ein didaktisches Lehrmittel, bei dem mit geringer Programmiererfahrung und geringen Kosten, schnell Artefakte entwickelt werden können.

Beispiel der Programmierung
Das Handyboard und der RCX stellen bildlich gesprochen die „Gehirne“ der Roboter dar. Dabei ist das Handyboard ein kompakter Haufen Computerchips, konstruiert, um Befehle entgegenzunehmen. Der RCX ist eine Erfindung von LEGO, die es erlaubt mit Hilfe von verschiedenen C ähnlichen Programmiersprachen Programme zu schreiben. Dabei müssen die Programme keine hohe Komplexität aufweisen, sie können auch kurz und einfach gehalten sein. Wenn man einen Roboter baut, sollte man Servomotoren benutzen, die gut mit dem RCX zu kontrollieren sind. Ein simples Programm für den RCX könnte folgendermaßen aussehen:

```
void main ()
{
motor(1,100);
sleep(2.0);
ao();
}
```

Jedes Programm dieser Programmiersprache sollte mit „void main()“ beginnen. Motor (1,100) bedeutet, dass der Motor 1 sich mit 100 % Geschwindigkeit dreht. Eine Anweisung wird mit einem Semikolon beendet. Folgt eine sleep–Anweisung, bedeutet dies lediglich, dass das Programm den vorherigen Befehl für die Zeit, die in den Klammern steht, ausführt und erst danach fortfährt. In diesem Beispiel dreht sich der Motor 1 mit 100 % für 2 Sekunden und wird dann mit dem Befehl „ao()“, für Alles aus, angehalten. Das Programm beginnt und endet jeweils mit den geschweiften Klammern.

8.2.3.4 AgentSheets

AgentSheets ist ein Tool zur visuellen Programmierung, das Computer-Laien und Endbenutzer jeden Alters den Aufbau eigener Computersimulationen und -anwendungen ermöglicht. Dieses agentenbasierte Simulations- und Entwicklungswerkzeug, fasst Agenten, Spreadsheets und Java-Portierung zusammen.

AgentSheets vornehmlichste Aufgabe ist es dabei, als Gedankenverstärker zu funktionieren, um das eigene Verständnis über die Problematik direkt in ein Programm zu überführen.
Das eingebaute Java-Tool Ristretto, soll dabei eine einfache Publikation mit Hilfe des Internet ermöglichen.

Aufbau und Ablauf der Anwendungen finden in einer visuellen Arbeitsumgebung statt, in der Endbenutzer ihre interaktiven Simulationen und Visualisierungen mittels einer Drag & Drop-Funktion erstellen können. Ein Agent ist in AgentSheets, ein vom Benutzer programmierbares Objekt. Agenten können auf verschiedene Ereignisse wie Mausklicks oder Tastatureingaben reagieren, sich auf einem sogenannten Worksheet bewegen, und diverse Funktionen ausführen: Aussehen verändern, E-Mails versenden, rechnen etc. Dabei arbeiten sie nicht alleine, sondern interagieren auf dem oben erwähnten rasterartigen Worksheet und erzeugen so eine Simulation des Objekts.

Ursprünglich war AgentSheets als High End Produkt zur Erstellung interaktiver Simulationen auf Großrechnern gedacht, doch vollzog sich ein Wandel hin zur Entwicklungsumgebung für eine breitere Maße an Benutzern. Der wesentliche Schritt dabei war etwas, das die Entwickler von AgentSheets „Tactile-Programming“ nennen.

Tactile bedeutet zu Deutsch fühlbar, sinnlich. Diese Vokabel soll andeuten, dass hier das visuelle Programmierkonzept schon erweitert wird. Denn im Gegensatz zu der üblicherweise rein technischen Sicht der Problematik, soll hier das Augenmerk auf den Wünschen der Entwickler liegen, nämlich den Sachverhalt zu visualisieren, zu verstehen und ihn zu diskutieren.

Um dies zu Ermöglichen, ist das Kommunizieren mit dem Programm wesentlich. Diese Eigenschaft soll durch die Programmiersprache von AgentSheets, Visual Agent Talk, gewährleistet werden. Visual Agent Talk ist eine regelbasierte Programmiersprache, die Bedingungen, Regeln und Aktionen unterstützt. Letztgenannte sind vollständige Objekte, die im Laufe der Programmentwicklung erkundet werden können. Das soll heißen, es ist zu jedem Entwicklungszeitpunkt möglich, eine Regel oder Bedingung zu testen, ohne vorher

einen Aufwendigen Übersetzungsvorgang in Kauf nehmen zu müssen. So kann immer wieder überprüft werden, ob das Verhalten der Agenten dem gewünschten entspricht.
Weitergehend wird bei der Selektion einer beliebigen Regel, Bedingung oder Aktion, eine Animation gestartet, die zum Verständnis der Funktionsweise des Objektes beitragen soll.

Beispiel
Elektric World (von Alexander Repenning) ist ein einfacher Zyklussimulator für elektrischen und magnetischen Strom. Es ist möglich Quellen, Drähte, Glühbirnen, manuelle Schalter, elektromagnetische Schalter und Spulen zu einem kompletten Kreislauf zu verknüpfen und den Stromfluss zu simulieren.

Konsequenz
AgentSheets ermöglicht dem Endbenutzer eine einfache Realisierung von Computeranwendungen ohne Programmieraufwand mittels intuitiver Drag & Drop-Funktion.

8.2.4 Skriptsprachen

8.2.4.1 Spreadsheet Programming, Spreadsheetprogrammierung in Excel

Spreadsheetprogrammierung ist ein Demonstrator für die EUD-Methode Skriptsprachen und außerdem ein Beispiel für eine „task-specific end user programming language“.

Endbenutzer haben innerhalb ihres Aufgabengebietes bestimmte Aufgaben zu erfüllen. Dabei erwarten sie aufgabenspezifische Funktionalität auf höhere Ebene, so dass sie sich weder groß einarbeiten, noch "low-level"-Programmierfunktionen anwenden müssen.

Bei der Spreadsheet-Programmierung werden Beziehungen zwischen Zell-Werten aufgebaut (dabei sind die Zellen die Variablen und die Funktionen die Beziehungen zwischen den Variablen). Dabei werden für die Funktionen arithmetische, statistische und logische Operationen angeboten (teilweise auch noch weitere). Die Spreadsheets erlauben den Nutzern bereits nützliche Arbeit mit einem geringen Einarbeitungsaufwand zu erstellen und dann in immer fortgeschrittenere Konzepte einzusteigen.
Durch eine deutlich flachere Lernkurve als bei normalen Programmiersprachen ist die Spreadsheetprogrammierung einfach zu erlernen und zu nutzen. Spreadsheets erlauben den Nutzern komplexe Anwendungen, mit einer großen Anzahl von Beziehungen zwischen den Entitäten innerhalb einer großen Problemstellung, zu erstellen. Dazu sind keine Kenntnisse höherer Programmiersprachen nötig (Variablen deklarieren, Speicher zuweisen, etc.). Dies führt allerdings auch zu deutlich weniger Flexibilität und Allgemeingültigkeit (da sie nur eine kleine Anzahl an Operationen erlaubt).

8.3 Konzeptioneller Ansatz

8.3.1 Ambient Prototyping

Die Firma Anoto bietet ein Konzept aus digitalen Stiften (z.B. Logitech IOPen) und einem mit einem feinen Punktraster überzogenen Papier an. Mit der Kamera im Stift werden nicht nur die Schreibbewegungen aufgezeichnet, sondern auch die genaue Position eines Stiftes auf dem Papier festgehalten. Das Punktraster kann dabei jede Stelle auf jedem Blatt des Anoto-Papiers eindeutig identifizierbar machen. In einem unveröffentlichten Prototyp einer schwedischen Universität wurde dieses Konzept dazu benutzt, Fenster von Anwendungen auf dem Papier nicht nur zu malen, sondern sie nach Zuordnung der gemalten Fensterelemente auch auf dem Papier zu bedienen (z.B. durch einen 'Doppeltip' auf den gemalten OK-Button). Hier wird also in der Realwelt die Anwendung auf dem Papier programmiert und genutzt.

8.3.2 Augmented Reality Tailoring

Zur Veranschaulichung des Augment Reality Tailoring Ansatzes, sei an dieser Stelle auf ein Projekt am Lehrstuhl Wulf verwiesen. Ein bei einem Industrie-Projekt entwickelter Ansatz zielt daraufhin ab, dass einem Fitnessgeräte-Nutzer, hilfreiche Informationen zur durchgeführten Übung übermittelt werden. Die auf ein mobiles Endgerät übermittelten Informationen helfen, eine Übung richtig und gelenkschonend durchzuführen. Die Informations-Ausgabe erfolgt dabei als Visualisierung auf einem Display eines mobilen Endgerätes (Augmented Reality). Dabei fließen Parameter die den persönlichen Status betreffen, mit in die Ausgabe ein. Diese sehr an persönliche Bedürfnisse angepasste Anwendung, ermöglicht ein hohes Maß an Individualisierung (Tailoring).

8.3.3 Model-Based Development

Ein gutes Beispiel für das Prinzip des Model-Based Developments ist das, vom Kölner Systemhaus Paragon entwickelte, Software-Entwicklungsframework namens PARAKLET, welches im Rahmen iterativer Software-Entwicklungsprozesse ein möglichst effizientes Prototyping ermöglichen soll. Integraler Bestandteil der PARAKLET-Entwicklungsumgebung ist dabei insbesondere ein so genanntes Mapping-Werkzeug, welches eine frühzeitige Verknüpfung (= Mapping) von prototypischen Benutzungsoberflächen (GUI) mit Objekt- und Datenmodellen ermöglicht. PARAKLET stellt dazu eine Reihe von Werkzeugen und Entwicklungs-Bibliotheken bereit, die es selbst wenig erfahrenen Java-Programmierern ermöglichen sollen, schon nach kurzer Schulungs- und Einarbeitungszeit, fehlerfreie und lauffähige Prototypen zu entwickeln.
Die lauffähigen Prototypen sind bereits mit einer konsistenten Datenbasis verknüpft und ermöglichen somit schon in frühen Prototyping-Stadien eine testweise Anwendung des entstandenen prototypischen Systems.

Anders als bei den, aus klassischen iterativen Prototyping-Verfahren bekannten, Mock-Ups und Beispiel-Oberflächen, handelt es sich bei den von Paragon bereitgestellten Artefakten nicht um sogenannte „Throw-Away“-Prototypen, sondern um bereits lauffähige erste Anwendungen. Statt von Prototypen wird daher auch eher von „lokalen Systemen“ gesprochen, die als „stand-alone“-Lösungen bereits lauffähig sind. Sobald diese prototypischen „lokalen Systeme“ eine hinreichend befriedigende Qualität erreicht haben, brauchen sie zur Fertigstellung der Anwendung, nur noch in eine bestehende Systemarchitektur einer Anwenderorganisation integriert werden.

8.3.4 Programming by Example

Beispiele für Programming by Example sind **Trainable Information Assistants**: Dabei handelt es sich um automatisierte Informationssammlungen im Netz, mit deren Daten über Flug, Hotel, Wetter und Karten, Nutzer ihre Reise einfach selber organisieren können.

Ein anderes Beispiel ist das Tool **Turquoise**, mit dem Nicht-Programmierer dynamische Internet-Seiten erstellen können oder das **Internet Scapbook**, bei dem die Browsing-Gewohnheiten des Nutzers aufgezeichnet werden. So muss der Nutzer nicht mehr die einzelnen Seiten ansteuern, sondern es werden die Informationen dieser Seiten auf einer Seite gesammelt und dort ständig aktualisiert. Ein anderes Tool ist **Creo**, das die Aktionsfolgen im Netz (z.B. Kauf im online-Shop) aufzeichnet, damit Nutzer bei erneuten Bestellungen schneller ihr Ziel (z.B. Abschluss einer Bestellung) erreichen. (Lieberman, H.; Faaborg, A 2006)

8.4 Gestaltungsprinzip

8.4.1 ActionWorks von ActionTechnologies

ActionWorks ist ein Beispiel für Domain-Oriented Design Environments. ActionWorks ist ein Bündel von „Person to Person"-Prozess-Werkzeugen. Es wird versucht die traditionelle kollaborative Art der ad hoc-Zusammenarbeit zwischen Menschen mit der transaktionalen Art der Geschäftsprozesse zu verbinden. Geschäftsprozesse können einerseits fest mit einem Process Builder modelliert werden, sind auf der anderen Seite offen für Anpassungen und für ad hoc-Prozesse. Die Modellierung im Process Builder erfolgt ausschließlich grafisch und über Wizard-ähnliche Dialoge. Ad hoc ablaufende Person-to-Person-Kollaborationen können unterstützt und in Echtzeit analysiert werden. Das hierbei stattfindende End-User-Development geschieht für den Endbenutzer also fast unbemerkt.

Dem Ansatz des Domain-Oriented Design Environments folgend, können ad hoc Aktivitäten in feste Abläufe eingebunden werden, sind also für den Endbenutzer leicht anpassbar. Zudem werden visuelle und dialogorientierte Modellierungen unterstützt. Es ist allerdings immer noch nötig, ein neues Tool zu erlernen.

8.4.2 Domain-Oriented Environments

8.4.2.1 Geschäftsprozess-Management mit ARIS for SAP Netweaver

Dies ist ein Beispiel für den Ansatz des Domain-Oriented Environments. Das ARIS Toolset der IDS Scheer AG unterstützt im Rahmen des Geschäftsprozessmanagements Beschreibungen und Analysen von Unternehmensabläufen.
Im Zusammenspiel von ARIS for SAP NetWeaver und SAP NetWeaver wird die dadurch entwickelte Geschäftsprozesssicht in SAP Anwendungen umgesetzt.

Der typische Arbeitsablauf beinhaltet drei Schritte

- Die Geschäftsprozessmodellierung mit „ARIS for SAP NetWeaver": Auf dieser abstrakten Ebene werden aus rein betriebswirtschaftlicher Sicht die Prozesse eines Unternehmens modelliert. Für den Modellierer bedarf es keines technischen Wissens oder der Durchführung von Programmierarbeit.
- Die Modellierung der Prozesskonfiguration: Als Ausgangspunkt stehen Referenzmodelle zur Verfügung. Anwender ordnen die Referenzmodelle im SAP Solution Manager mit den Prozessen und Transaktionen der SAP-Systeme in die Prozessarchitektur ein und passen sie an die speziellen unternehmenstypischen Bedürfnisse an. Die Verwendung von SAP Referenzinhalten für die fachliche Prozessbeschreibung und die Konfiguration sowie die Reduzierung von Komplexität durch Standardmethoden, machen dem Anwender diese Aufgabe leichter und befreien ihn so von Anpassungen durch programmierende Eingriffe.
- Die Verknüpfung der modellierten Prozessflüsse mit den Anwendungen: Ausgehend von der Geschäftsprozessanalyse werden systemübergreifende Prozesse mit den Applikationsprozessen der SAP- und Nicht-SAP-Anwendungen zu einem in SAP XI ausführbaren Prozessmodell verbunden. Dieser Schritt muss zur Zeit noch manuell vollzogen werden, so dass die direkten Möglichkeiten eines Endbenutzers noch beschränkt bleiben.

„ARIS for SAP NetWeaver" ist von der Konzeption her ein typisches Beispiel für ein Werkzeug, mit dem Endbenutzern leicht ihre Systeme an unternehmensspezifische Anforderungen anpassen können. Dem Endbenutzer reicht alleine sein betriebswirtschaftliches Wissen aus, technische Kenntnisse sind nicht erforderlich.

Geschäftsprozessmanagementsysteme wie ARIS, lassen den Endbenutzern in der für ihn gewohnten "Sprache" sprechen und erlauben ihm so Anpassungen an Systeme leicht vornehmen zu können. Allerdings läßt sich dabei der Automatisierungsgrad des Prozesses, vom Erstellen eines Geschäftsprozessmodells bzw. Workflows hin zur Systemimplementierung, nur in einem geringen Maß nahtlos durchführen.

8.4.3 Direct Activation

Ein Beispiel für Direct Activation ist die Funktion der „F1-Taste“, die erwartungsgemäß die Hilfe aufruft. Ein Weiteres Beispiel ist das Kontextmenü zu einer Funktion, das in einigen Anwendungen durchgängig über die rechte Maustaste aufgerufen werden kann. Der dort vorhandene Menüpunkt „Eigenschaften“, ist ein Beispiel für eine mehrstufige Aktivierung.

Ein anderes Beispiel für Direct Activation, ist das Dashboard des Mac OS. Hier ist das Bezugsmittel die räumliche Nähe. Als Interaktionspunkt dient eine Schaltfläche auf dem Dashboard.

8.4.3.1 Virtual Post-Its

Virtual Post-Its sind ein weiteres Beispiel für Direct Activation. Häufig liefern heutige Hilfesysteme nicht die gewünschte Antwort auf offene Fragen oder erst nach langer Suche. Dabei wird der Kontext des Nutzers bei der Suche, meist nicht berücksichtigt.
Mittels virtueller Post-Its wird ein kontextbasiertes Hilfesystem aufgebaut, welches allein durch die Interaktion der einzelnen Nutzer einer bestimmten Anwendung entsteht. Die Nutzer können Fragen in Form von virtuellen Post-Its, kontextbezogen an einzelne GUI-Objekte, wie z.B. Schaltflächen oder Tabellen, einer Anwendung anbringen. Diese sollen dann von anderen Nutzern, bei denen die virtuellen Post-Its ebenfalls an den Objekten erscheinen, beantwortet werden. Grundlage dieser Interaktion ist ein Wiki-System, in dem der Content der Post-Its verwaltet wird. Um die einzelnen GUI-Objekte identifizieren zu können, wird ein eindeutiger Schlüssel benötigt. Dieser setzt sich aus mehreren Attributen des Objektes zusammen und ist die Basis für die Verknüpfung zum Wiki. Ein virtuelles Post-It besteht dabei aus mehreren Hyperlinks, die die Fragen oder Hilfethemen repräsentieren. Diese Links öffnen bei einem Mausklick einen neuen Bereich in der Anwendung. Hier werden dann die Informationen oder Antworten speziell zu diesem GUI-Objekt aus dem Wiki-System geladen und angezeigt. Um nach einem Aufruf der Post-Its über eine Schaltfläche in der Toolbar, die Übersichtlichkeit auf der Anwendungsoberfläche zu bewahren, ist ein Schieberegler implementiert, der den Grad der Detailliertheit und somit die Anzahl der angezeigten Post-Its festlegt.

Die Grundidee ist die Hilfe im Kontext des Objektes zu präsentieren oder einige Informationen über ein bestimmtes Objekt bereitzustellen. Das Feature dabei ist, nicht nur ein Objekt zu nutzen um die Hilfeinformationen zu erfassen, sondern auch den Ort des Objektes zu nutzen um Hilfeinformationen zu präsentieren.

Konsequenz
Der Nutzer kann direkte Hilfe in Bezug auf seinen Kontext erhalten oder erfragen.

8.4.3.2 Konfabulator/ Yahoo Widget Engine

Der Konfabulator ist ebenfalls ein Beispiel für Direct Activation. Die kostenlose Yahoo Widget Engine - früher unter dem Namen Konfabulator bekannt - ist eine Sammlung von Widgets genannten Hilfsprogrammen. Widgets sind Java-Script Anwendungen, die eine gezielte aber unaufdringliche Darstellung von Systeminformationen oder Informationen aus dem Internet realisieren.

Widgets zeichnen sich vor allem durch ihre leichte Anpassbarkeit aus. Jedes auf dem Desktop angezeigte Widget, kann jederzeit wieder ausgeblendet werden und stellt dann seinen Dienst ein. Über das "Konfabulator"-Symbol in der Systray, kann das betreffende Widget wieder eingeschaltet werden. Außerdem kann jedes Widget an einer beliebigen Stelle auf dem Desktop, durch ziehen mit gedrückter Maustaste, verschoben werden. Sollten zu viele andere Fenster die Widgets überdecken, kann man mit "F8" in den "Konsposé"-Modus wechseln. Dadurch werden alle Fenster von Windows-Programmen ausgeblendet und nur die Widgets angezeigt. Die meisten Widgets lassen sich zusätzlich nach eigenem Wunsch verändern und anpassen. Durch die einfache Java-Script-Programmierung und dem Vorteil, dass jedes Widget problemlos auf anderen Desktops eingesetzt werden kann, gibt es mittlerweile eine Vielzahl unterschiedlicher Widgets.

Beispiele
Das Programm beinhaltet über 20 verschiedene Widgets, unter Anderen: Analoguhr, Kalender, Wetter, iTunes-Steuerung, Akkulaufzeit und Aktienkurse. Über 1.000 weitere Widgets aller Art können kostenlos unter widgets.yahoo.com oder www.widgetgallery.com heruntergeladen werden.

In der Rubrik "System Utilities" findet man zahlreiche Widgets, die beispielsweise den Füllgrad der Festplatte, die Auslastung des Systems oder die Qualität der WLAN-Verbindung anzeigen. Mit "URR" kann ein RSS-Reader direkt auf dem Windows-Desktop integriert werden. Speziell an deutsche Anwender, die immer auf dem Laufenden sein möchten, richtet sich das "German News Widgets". Täglich kommen neue Widgets hinzu oder alte werden aktualisiert.

Konsequenz
Die Widgets zeichnen sich durch einfache Programmierung in Java-Script aus. Sie bieten zudem eine leichte Austauschbarkeit und Transportierbarkeit sowie eine einfache Integration auf anderen Plattformen.

8.5 EUD Services

8.5.1 Community Help in Context

Das Konzept „Community help in Context“ verbindet den Ansatz einer kontextsensitiven Hilfe mit einem Community Konzept. Die Architektur des ChiC ist in drei Module aufgeteilt, AIM (Application Integration Module), CBHS (Community Based Help System), CAM (Contex-Aware Adaptation Module) und kann durch das AIM Modul in beliebige Anwendungen integriert werden. Die einzelnen Komponenten besitzen folgende Funktion:

- AIM – Application Integration Module
  Das Application Integration Module integriert das Hilfesystem, direkt in die vom Benutzer verwendete Applikation. Der Benutzer interagiert dabei über das AIM Modul mit dem Hilfesystem und erhält Hilfeinformationen im Kontext seiner Arbeit, auf dessen Anforderung bereitgestellt. Drei Eigenschaften werden dabei durch das AIM – Modul realisiert:
- CBHS – Community Based Help System
  Das CBHS–Modul stellt die Communityfunktionalitäten bereit, d.h. den Punkt, an dem Benutzer zusammentreffen, um ein gemeinsames Ziel zu verfolgen. Dabei liegt hier die Flexibilität in der Auswahl von Communitysystemen. Das Modul gewährleistet die Verwendung traditioneller Communitysysteme, jedoch bestückt mit speziellen Funktionalitäten, die für die Bereitstellung von kontextsensitiven Hilfen nötig sind.

- CAM – Context-Aware Adaptation Module
  Das CAM verbindet nun das Application Integration Modul (AIM) und das Community Based Help System (CBHS). Es fungiert somit als Vermittler zwischen den beiden Modulen und fokussiert dabei auf den genauen Eintrittspunkt in das CBHS-Modul. Die Herausforderung des CAM, liegt dabei in der Realisierung von Kontexterfassungs-Algorithmen. Abhängig vom aktuellen Arbeitskontext des Benutzers und dem modellierten Benutzermodell, muss der Einstiegspunkt in das CBHS an den Benutzer bzw. an die Nutzungssituation angepasst werden. Zusätzlich sollte nicht nur ein bestimmtes CBHS als Hilferessource angebunden werden können; das CAM sollte auch in der Lage sein, andere Hilferessourcen über das Internet mit einzubinden. Somit könnte das CAM auch als Provider unterschiedlicher Hilfequellen fungieren.

8.5.2 Configuration Sharing

Beispiele für Configuration Sharing sind das Buttons-System (MacLean, A., K. Carter, et al. 1990), mit dessen Hilfe kleine Buttons in einem X-Windows-System nicht nur erstellt, sondern auch per E-Mail an Kollegen gesendet werden können sowie die Übertragung von Toolbars und anderen Konfigurationselementen von MS Word in einem Groupwaresystem (Kahler 2001). Ein anderes Beispiel wäre die Konfiguration von Handys durch den Mobilfunkbetreiber mit Hilfe von SMS.

8.5.3 Demonstration Support

Demonstration Support kann dazu benutzt werden, ein Repository von Nutzungen eines Werkzeugs anzulegen, welches dann z.B. auch bei der Beschreibung organisationaler Vorgänge eine wertvolle Ergänzung sein kann.

Ein Beispiel für die Unterstützung von Demonstration Support findet sich in dem Programm NetMeeting. NetMeeting ist ein Open Source Konferenz-Tool von Microsoft zum Führen von Audio-, Video- und Datenkonferenzen. Die Application Sharing-Funktion unterstützt die Gruppenarbeit mit beliebigen Windows-Anwendungen. Das entsprechende Anwendungsprogramm muss daher nur auf einem der verbundenen Computer installiert sein. NetMeeting bietet auch ein Whiteboard. Alle Funktionen sind auf Industriestandards basiert, so dass Anwender anderer kompatibler Programme auch teilnehmen können.

8.5.4 Observation Support

Technisch sind unterschiedliche Realisierungen von Observation Support denkbar, von der Übertragung von Bildschirminhalten bis hin zum Tracking von Mausklicks in einer Anwendung, die einem Beobachter woanders in irgendeiner Form visualisiert werden. Ein Beispiel für die Visualisierung von akkumulierten Nutzungen hat Linton (2003) am Beispiel der Visualisierung der Auswahlhäufigkeit von Menüpunkten in Microsoft Word beschrieben.

8.5.5 Recommendation Support

Der Online-Buchhandel Amazon ist dafür ein prominentes Beispiel. Hier werden dem Kunden zu einem ausgewählten Produkt Empfehlungen präsentiert, die ihm die Navigation im umfangreichen Produktsortiment erleichtern sollen. Dabei verwendet Amazon mehrere Empfehlungsverfahren bzw. -arten, z.B. Lieblingslisten anderer Kunden oder Produktpaare, die häufig gemeinsam gekauft wurden.

Die METRO-Gruppe geht noch einen Schritt weiter und prüft gegenwärtig den Realtime-Recommendation-Ansatz für die physische Einkaufswelt in ihrem Future-Store. Über sog. Personal Shopping Assistants (PSA) - am Einkaufswagen installierte Displays – werden die jeweiligen Artikel vom Kunden selbst erfasst. Unmittelbar dabei werden ihm, auf seinen

bisherigen Einkauf perfekt abgestimmte, weitere Artikelempfehlungen in Echtzeit unterbreitet.

8.5.6 Objektifizierung

Beispiele für Objektifizierung sind der Austausch von Anpassungen mit Dritten (z.B. Teilen von .cshrc Files bei Unix Systemen), die Archivierung und Sicherung von Systemzuständen (z.B. Sicherungspunkte bei Windows XP und die Individualisierung bei Systemwechsel (z.B. das Arbeiten mit dem Rechner des Kollegen)

8.5.7 Disclosure Principle

Das von DiGiano entworfene und untersuchte Beispiel ist ein Graphikprogramm, mit zwei parallelen und synchronisierten Eingabeinterfaces. Im oberen Teil des Programms ist eine GUI-Schnittstelle, mittels der man die graphischen Objekte mittels Direct Manipulation bearbeiten kann. Im unteren Teil ist eine Kommandozeilen-Shell mit deren Hilfe die graphischen Objekte durch Kommandos oder daraus zusammengesetzte Skripts.
Die Besonderheit der Anwendung ist nun, dass Eingabe über das eine Interface auch im anderen Interface dargestellt wird und visa versa. Damit erreicht man, dass man die Vorteile von „Direct Manipulation“ hat, ohne dass die Mächtigkeit von einer Kommandozeilen Steuerung hinter dem Interface verloren geht.

9 Einsatzpotentiale für EUD-Pattern

Nachdem in den vorigen Kapiteln die konzeptionellen und technischen Grundlagen für den Einsatz von EUD-Aneignungs- und Unterstützungsmaßnahmen gelegt ist, werden im Folgenden relevante Ansatzpunkte in den identifizierten Prozessszenarien beschrieben. Der Meilenstein „Whitepaper: Endbenutzerorientierte Anpassbarkeit für ERP-Systeme in KMU“ beschreibt detailliert die einzelnen Szenarien. Dieses Kapitel ordnet Unterstützungsmaßnahmen auf „High Level“- Ebene (siehe Kapitel 5 „EUD-Services“) den entsprechenden Szenarien zu. Aufgrund der Vielzahl der identifizierten Ansatzpotentiale werden im weiteren Projektverlauf nur die innovativsten Ansätze weiterverfolgt. Dazu sind noch Workshops mit den beteiligten Projektpartnern vorgesehen. Die Ausführung dieses Kapitels werden im weiteren Projektverlauf fortgeschrieben und angepasst.

Einer der wichtigsten Aspekte bei den Ansatzpotentialen innerhalb der Prozessszenarien liegt in der Unterstützung von Nutzergemeinschaften („Community-Support“). Die im betrieblichen Umfeld vorhandene Expertise sollte in vielfacher Weise besser nutzbar sein. Existente Probleme oder Schwachpunkte werden von Einzelnen gelöst, ohne dass von diesen Lösungsbeschreibungen auch Andere profitieren. Gute Ideen und Verbesserungsansätze der Mitarbeiter gehen vielfach unter. Die Flexibilität eines Unternehmens sollte nicht nur von der Expertise und Einflussnahme von „Experten“ abhängen. Dieses Potential sollte einer großen Nutzerbasis zukommen, damit jeder Einzelne im Fall einer wie auch immer gearteten Veränderung von innen oder außen schnell und dynamisch handeln kann.

9.1 Problemanalyse

In Abbildung 4 sind die im „Whitepaper: Endbenutzerorientierte Anpassbarkeit für ERP-Systeme in KMU“ besonders unterstützungsrelevanten Prozessschritte in dem Prozessszenario „Problemanalyse“ rot markiert. Tabelle 1 stellt geeignete EUD-Services für dieses Szenario den konkreten Unterstützungsmaßnahmen gegenüber. Dieses Unterstützungspotential ist Grundlage für mögliche prototypische Realisierungen.
Dabei geht es um die Möglichkeit für Endbenutzer, Probleme mit der existierenden Infrastruktur zu artikulieren und so einer Bearbeitung zugänglich zu machen. Entscheidend ist dabei, dass EUD-orientierte Maßnahmen es ermöglichen, hier während der Nutzung, d.h. im Kontext des Problemauftretens und so unmittelbar wie möglich, ein Problem zu erfassen. Dabei ist noch nichts über die Behebung des Problems gesagt, welche ebenfalls (die notwendigen Infrastrukturen und Interfaces vorausgesetzt) durch die Endbenutzererfolgen könnte, oder aber durch Softwarehersteller bzw. Drittbetreuern der Softwareinfrastruktur (z.B. Unternehmensberatungen). Die zusätzliche Qualität von EUD-Maßnahmen lässt sich in solchen Szenarien dadurch unterstreichen, dass auch das Feedback über den Vollzug von Änderungen im Endbenutzerkontext rückgemeldet werden kann.

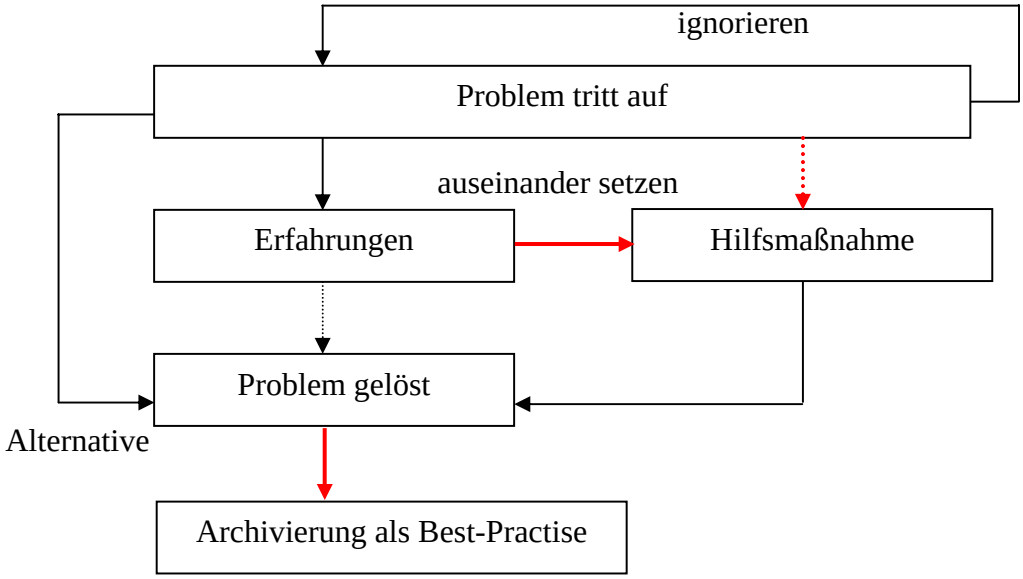

Abbildung 4: EUD-Potential zur Unterstützung der Problemanalyse

EUD-Service	EUD-Ansatzpotential
Community Help in Context	Aufruf Kontextabhängiger Supportfunktionen
Configuration Sharing	Abruf von „funktionierenden“ Konfigurationen
User Community Support, Unterstützung von Benutzergemeinschaften	Bereitstellung einer geeigneten Infrastruktur für Community-Support
Demonstration Support	Bereitstellung einer Hilfe-Basis in der Problemlösungsschritte beschrieben sind – z.B. auch als Video, Audio, Grafik etc.
Recommendation Support	Nach Hilfeanfrage durch Nutzer können andere Nutzer Problemlösungsvorschläge unterbreiten

Tabelle 1: Zuordnung EUD-Services EUD-Ansatzpotential

9.2 Änderung der Infrastruktur aus Anwendersicht

In Abbildung 5 sind die im „Whitepaper: Endbenutzerorientierte Anpassbarkeit für ERP-Systeme in KMU“ besonders unterstützungsrelevanten Prozessschritte in den Prozessszenario „Infrastrukturänderung“ rot markiert. Tabelle 2 stellt geeignete EUD-Services für dieses Szenario den konkreten Unterstützungsmaßnahmen gegenüber.

Dabei geht es hier um Szenarien, in denen Endbenutzer sich einer extern motivierten Infrastrukturänderung gegenübergestellt sehen. EUD-Maßnahmen helfen hier, während der Nutzung eine Feinabstimmung zwischen neuen Infrastrukturmerkmalen und tatsächlichen bzw. möglichen Nutzungen im Aufgabenkontext herzustellen. Dabei ist jedoch zu beachten, dass es nicht nur um Startbetreuung und Schulungen gehen kann, sondern vielmehr eine kontinuierliche Verbesserung der Nutzungspraxis erforderlich ist. Deshalb sind auch hier EUD-Maßnahmen sinnvoll, die einen entsprechend nachhaltigen Dialog über tatsächliche und alternative Nutzungen bzw. über vorhandene und alternative Funktionalitäten im Laufen halten.

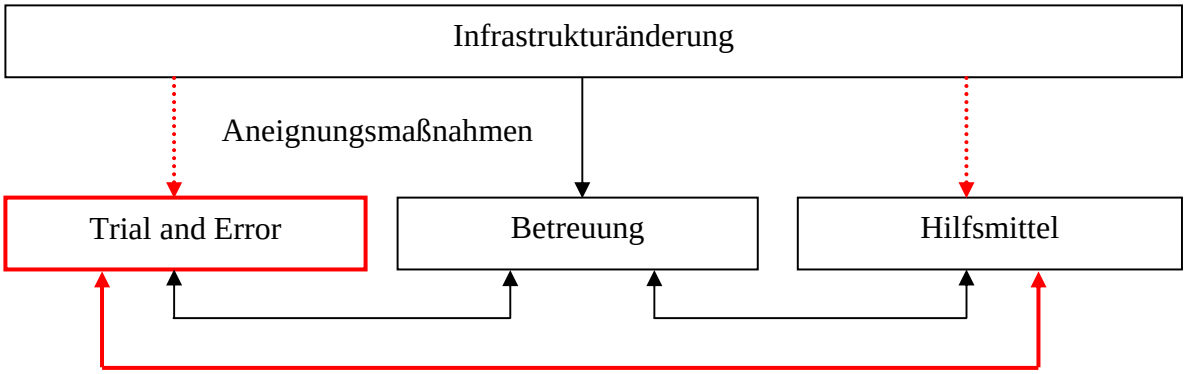

Abbildung 5: EUD-Potentiale im Kontext von Infrastrukturänderungen

EUD-Service	EUD-Ansatzpotential
Community Help in Context	Aneignungsunterstützung für eine neue Infrastruktur basierend auf kontextabhängigem Support
Configuration Sharing	Austausch von „Best Practice“ Konfigurationen – z.B. Übertragung des Customizing auf Abteilungen o.ä.
User Community Support, Unterstützung von Benutzergemeinschaften	Bereitstellung einer geeigneten Infrastruktur für Community-Support
Configuration Browsing, Durchsuchen von Konfigurationssammlungen	Bevor Maßnahmen zum Configuration Sharing durchführbar sind, können alternative Konfigurationen von verschiedenen Nutzern angezeigt werden
Delegation Support, Delegationsunterstützung	nach Infrastrukturänderungen können bestimmte Personen – z.B. Key User – zur Erstellung von Aneignungshilfen aufgefordert werden
Demonstration Support	Bereitstellung einer Basis von „Best Practice“ Beispielen in der Problemlösungsschritte beschrieben sind – z.B. auch als Video, Audiodatei etc.
Observation Support	bei dem Einsatz von Fernwartungstool wie Remotesteuerung können „Experten“ das Vorgehen des Nutzers beobachten und Hilfestellungen geben

Tabelle 2: Zuordnung EUD-Services EUD-Ansatzpotential

9.3 Änderung der Nutzungspraxis

In Abbildung 6 sind die im „Whitepaper: Endbenutzerorientierte Anpassbarkeit für ERP-Systeme in KMU“ besonders unterstützungsrelevanten Prozessschritte in den Prozessszenario „Änderung der Nutzungspraxis“ rot markiert. Tabelle 3 stellt geeignete EUD-Services für dieses Szenario den konkreten Unterstützungsmaßnahmen gegenüber. Dabei geht es um Szenarien, in denen Änderungen problemorientiert aus der Nutzungspraxis initiiert werden. Man kann diese Szenarien als ‚Breakdown’-Szenarien bezeichnen, wobei es sich dabei nicht um einen wirklichen technischen Zusammenbruch handeln muss, sondern lediglich eine (möglicherweise subjektive) Inkongruenz zwischen den erwarteten und den möglichen Nutzungsmöglichkeiten einer Softwareinfrastruktur festgestellt wird. Faktisch kann dem eine falsche Erwartungshaltung, ein echtes technisches Problem, eine u.U. nicht vollständig

abwärtskompatible Infrastrukturänderung oder eine nicht vollständig übersehbare Änderung des Aufgabenkontextes zugrunde liegen.

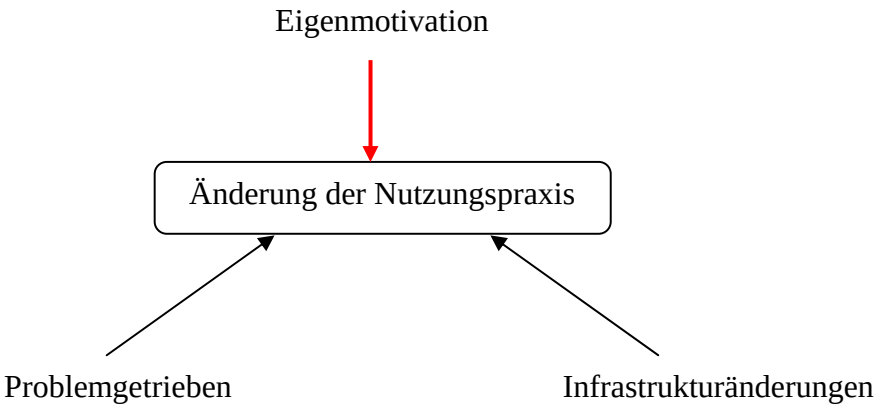

Abbildung 6: EUD-Potentiale zur Unterstützung der Änderung der Nutzungspraxis

EUD-Service	**EUD-Ansatzpotential**
Community Help in Context	Eingabe von Problembeschreibungen und selbst erstellten Lösungen in Abhängigkeit des Kontextes -> Eigenmotivierte Lösung steht auch anderen Nutzern in diesem Kontext zur Verfügung
Configuration Sharing	Freigabe von selbst erstellten Konfigurationen mit einer zugehörigen Beschreibung
Negotiation Support, Aushandlungsunterstützung	Diskussion von „Best Practice“ erst in einer Gruppe von „Experten“ (z.B. Key User) bevor Freigabe für Andere erfolgt
User Community Support, Unterstützung von Benutzergemeinschaften	Bereitstellung einer geeigneten Infrastruktur für Community-Support
Demonstration Support	Bereitstellung von Visualisierungen von eigenmotivierten Lösungen – z.B. kurzes Tutorial
Recommendation Support	Möglichkeit die eigenmotivierten Lösungen Anderen zu empfehlen mit Möglichkeit einer Feedbackfunktion zur Bewertung der Lösung durch Andere

Tabelle 3: Zuordnung EUD-Services EUD-Ansatzpotential

9.4 Nutzungsinnovationen

In Abbildung 7 sind die im „Whitepaper: Endbenutzerorientierte Anpassbarkeit für ERP-Systeme in KMU“ besonders unterstützungsrelevanten Prozessschritte in den Prozessszenario „Nutzungsinnovation“ rot markiert. Tabelle 4 stellt geeignete EUD-Services für dieses Szenario den konkreten Unterstützungsmaßnahmen gegenüber. Diese Szenarien adressieren eher Hindernisse bei der Entstehung und Verbreitung von ‚Best Practice’ in Unternehmen. Dabei sind EUD-Methoden deshalb eine wertvolle Ergänzung von durch die Management-Ebene initiierten Praxisvermittlungen, weil sie neben der reinen Wissensvermittlung auch der Etablierung einer Unternehmenskultur zuträglich sind, die Endbenutzer als Artikulatoren von Problemen und Gestalter von Infrastrukturen ernst nimmt. Benutzer nehmen dann auch die

strategische Bedeutung ‚guter’ Infrastruktur für die eigene Arbeit stärker wahr, und können deren Verbesserung einerseits als ständige Nebenaufgabe in den Arbeitsalltag einbauen, und können zum anderen auch eher ihre Leistungen auf diesem Gebiet Kollegen und Vorgesetzten darstellen.

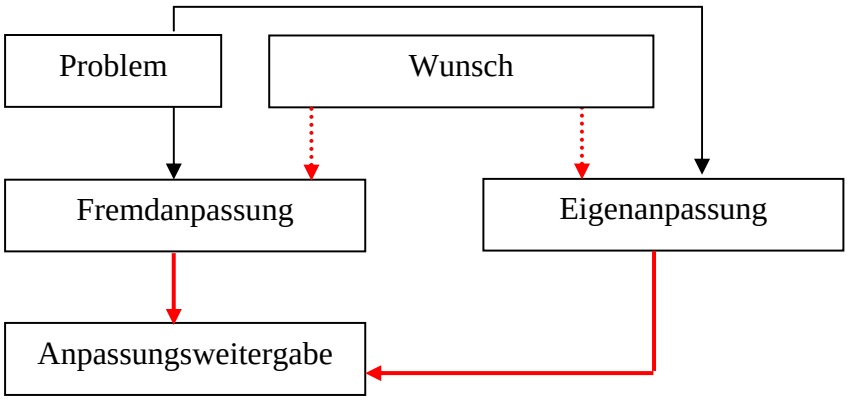

Abbildung 7: EUD-Potentiale zur Unterstützung von Nutzungsinnovationen

EUD-Service	**EUD-Ansatzpotential**
Community Help in Context	Verweis auf bestehende Anpassungslösungen bei Hilfeanforderungen in Abhängigkeit vom aktuellen Kontext
Configuration Sharing	Austausch von eigenangepassten Lösungen – Vorlagen, Views, Berechnungen, Auswertungen etc.
Negotiation Support, Aushandlungsunterstützung	Bereitstellung geeigneter Maßnahmen zur Unterstützung der Diskussion über selbst angepasste Lösungen
User Community Support, Unterstützung von Benutzergemeinschaften	Bereitstellung einer geeigneten Infrastruktur für Community-Support
Configuration Browsing, Durchsuchen von Konfigurationssammlungen	Weitergabe selbst angepasster Lösungen bzw. Verbesserungs- und Unterstützungs-Maßnahmen
Demonstration Support	Vor Weitergabe von Eigenanpassungen an Andere sollte Möglichkeit existieren, deren Lösungspotential zu demonstrieren
Recommendation Support	Empfehlung von Eigenanpassungen an Andere; Bewertung dieser Empfehlungen; Einfluss der Bewertungen in den Empfehlungsprozess
Objektifizierung	Unterstützung der Verwaltung und Nutzung unterschiedlicher Anpassungsmaßnahmen; beispielsweise Bereitstellung einer Undo-Funktion um bestehende Konfiguration nach einem Wechsel wiederherzustellen
Disclosure Principle	Neben offensichtlichen Anpassungen des Interfaces sollen die einzelnen Änderungsschritte auch für Andere nachvollziehbar sein; der Anpassende könnte dazu eine kurze Anleitung erstellen; auch automatische Makroaufzeichnungen der Änderungsschritte sind denkbar

Tabelle 4: Zuordnung EUD-Services EUD-Ansatzpotential

10 Zu einer Roadmap hochanpassbarer Systeme

Im vorliegenden Dokument wurden vor allen Dingen die Voraussetzungen und Möglichkeiten der Entwicklung hochanpassbarer Systeme gesammelt und beschrieben. In diesem letzten Kapitel wird noch einmal zusammengefasst, was allgemein für die Entwicklung von Roadmaps zur Realisierung der beschriebenen Maßnahmen gesagt werden kann.
Im Groben kann man die Entwicklung hochanpassbarer Systeme in drei Schritte teilen: der Entscheidung für geeignete Flexibilisierungstechnologien, die Entscheidung für technische und organisatorische Maßnahmen zur Verwaltung dieser Flexibilität und der Etablierung einer kontinuierlichen endbenutzergetriebenen Praxis der Weiterentwicklung von Infrastrukturen im Arbeitskontext. Wir diskutieren hier noch einmal einige Herausforderungen und Möglichkeiten.

10.1 Flexibilisierungstechnologien

In den letzten Jahrzehnten hat das Software Engineering immer wieder neue Technologien zur Flexibilisierung von Softwareinfrastrukturen hervorgebracht. Von Softwaremodulen über Softwareobjekte und Softwarekomponenten sind wir aktuell bei Softwareservices angelangt. Dabei ging es tatsächlich in jeder neuen Generation technischer Konzepte darum, Software immer besser in kleine, verständliche Einheiten zerlegen zu können. Im Kontext des End-User Developments darf man jedoch nicht aus den Augen verlieren, dass das Ziel dieser Aktivitäten die Wiederverwendung von Software war, und die Adressaten dieser Konzepte waren im Normalfall ausgebildete Programmierer. Das hat zum Einen Folgen für die Konzepte selber, die stark auf informatischen Traditionen aufbauen und einen fundamental formalen Charakter beibehalten haben. Zum Anderen reflektieren die darin adressierten Probleme die Dynamiken von Szenarien des End-User Developments nur begrenzt. Zum einen kann man dabei die mangelnde Fähigkeit, Systembestandteile während der Nutzung weiterzuentwickeln bzw. auszutauschen als Defizit herausstreichen. Auch wenn es Speziallösungen aus dem Bereich hochverfügbarer Systeme gibt, sind aktuelle Flexibilisierungstechnologien vor ihrem immer genau darauf zu testen, inwieweit sie ‚Hot-Swap'-fähig sind. Für häufig verwendete Standardtechnologien wie Enterprise Java Beans oder Web Services trifft dies nur sehr bedingt zu.
Zum Anderen bilden existierende Flexibilisierungstechnologien auf der Protokollebene nicht alle aus Endbenutzersicht relevanten Zusammenhänge ab. So ist z.B. die Konsistenz von Visualisierungsstilen unterschiedlicher Komponenten oder Services nicht auf Protokollebene zusicherbar. Zurzeit müssen derartige Abhängigkeiten somit in gesonderten Strukturen verwaltet werden (z.B. in Style Guides). Eine Verankerung in direkt in den Komponenten und Services ist selbstverständlich möglich, allerdings führt die ohne einen entsprechenden Standard zu Lösungen, die lediglich innerhalb eines bestimmten Komponenten- oder Serviceuniversums Gültigkeit haben.

10.2 Organisatorisches und technisches Flexibilitätsmanagement

End-User Development verspricht auf der einen Seite ein schnelleres und kosteneffizienteres Management unternehmenswichtiger Softwareinfrastrukturen durch Entwurf, Anpassung und Weitergabe lokaler Lösungen durch Endbenutzer, es stellt aber gleichzeitig hohe Anforderungen an die Managementschnittstelle von Flexibilisierungstechnologien. Im Gegensatz zum Szenario der Softwarewiederverwendung muss diese im End-User Development einer großen Bandbreite von Arbeitskontexten und Wissenshintergründen seitens der Endbenutzer gerecht werden. Dem kann auf mehrfache Weise entsprochen werden.

Auf einer rein technischen Ebene sind beispielsweise Ideen aus dem Bereich Visuelle Programmierung und Domänen-orientierte Designumgebungen relevant. Für eine Roadmap

ist zu beurteilen, welche Maßnahmen für eine wie große Benutzerbandbreite geeignet sind, ob es sinnvoller ist, gut visualisierte abstrakte Einheiten zu bilden, oder sich eher der Metaphern einer Benutzerdomäne zu bedienen. Gleichzeitig muss hinterfragt werden, was an technischer Komplexität verborgen werden soll, und was gerade nicht, da es zum Verständnis der zu konfigurierenden Technologien unabdingbar ist (vgl. Pattern Disclosure-Prinzip). Entsprechende visuelle Konzepte können auch auf bestimmte Funktionalitätsdomänen beschränkt realisiert werden, um die Konsistenz der verwendeten Metaphern nicht zu sehr in Frage zu stellen. Trotzdem verbleibt auch dort das Problem, dass es unklar ist, ob eine Zerlegung in Funktionsdomänen von Seiten der Designer von Flexibilisierungstechnologien auch dem Bild der Zerlegung in Funktionalitäten seitens der Endanwender entspricht. Hierbei kann sich auch die in der Informatik traditionelle Aufteilung von Interfaceebene und Anwendungslogik als allgemeiner zusätzlicher Konfliktgrund bemerkbar machen.

Ebenfalls noch auf der eher technischen Seite muss für eine Roadmap die Frage nach geeigneten Usability-Untersuchungen und –Maßnahmen gestellt werden. Wie kann angesichts der zu erwartenden Heterogenität der Benutzer wie auch der eben angedeuteten Heterogenität der Schnittstellen auch bei anspruchsvollen Aufgaben eine Benutzbarkeit im Anwendungskontext sichergestellt werden?

Eher im organisatorischen Bereich liegt die Frage der Anpassung von Infrastrukturentwicklungsprozessen an die Möglichkeiten, die Flexibilisierungstechnologien bieten. Letztendlich müssen nicht nur Technologien flexibel managebar gemacht werden, sondern auch die Prozesse, die diese unterstützen. Die ist zum einen von den etablierten formalen und informalen Organisationsstrukturen her möglicherweise Problematisch, zum anderen ist mit einer solchen Strategie der Prozessflexibilisierung auch ein erhöhter Kommunikationsaufwand verbunden, der die Nutzenbilanz von End-User-Development-Maßnahmen belastet. Im einzelnen umfasst dies auch - neben Aspekten wie dezentraler Rechtevergabe für Modifikationen an Infrastrukturen - die Frage nach dem Kompetenzen und Zuständigkeiten im Rahmen möglicher Auftragsvergabe an (organisationsexterne) Dritte.
Im Bereich kooperativer EUD-Maßnahmen ist jedoch die Ermöglichung organisationsinterner Delegationsmuster der zentrale Punkt. In allen beobachteten KMU gab es zum Teil ausgeprägte Strukturen der Arbeitsteilung, wenn es um die Verteilung technischen Wissens über bzw. organisatorischer Verantwortung für Teile der Softwareinfrastruktur ging. Die aktive Unterstützung derartiger Delegationsprozesse, die letztendlich auch individuelles wie organisationales Lernen ermöglichen, ist bei der Etablierung hochanpassbarer Systeme einer der wichtigsten Punkte.

Eine Roadmap für hochanpassbare Infrastrukturen muss auch diese Verknüpfungen zwischen der technischen und der organisatorischen Ebene berücksichtigen.

10.3 Etablierung einer reflektiven Nutzungspraxis

Die Etablierung einer Nutzungspraxis, die auch wirklich Nutzen aus der zunehmenden Flexibilisierung von Softwareinfrastrukturen ziehen kann, geschieht jedoch keineswegs ausschließlich auf der Basis technischer und prozessorientierten Maßnahmen. Auch wenn Endbenutzern so mehr Mitgestaltungsmöglichkeiten zur Verfügung stehen, ist es letztendlich eine Frage der Unternehmenskultur, wie nachhaltig von diesen Möglichkeiten Gebrauch gemacht werden kann. Die Etablierung einer Nutzungskultur, in der die Nutzung von Technologien von einem kontinuierlichen Verbesserungsprozess begleitet wird, ist dem Problem der Etablierung von Wissenstransferprozessen in Unternehmen ähnlich. Erfahrungen in diesem Bereich haben gezeigt, dass naive Ansätze, wie z.B. Anreizsysteme, die ausschließlich auf monetäre Äquivalente für ‚richtiges' Verhalten setzen, zu kurz greifen. Es ist vielmehr notwendig, das gesamte soziale System und seine Dynamik als Plattform für eine

Etablierung neuer Nutzungstraditionen zu nutzen. Hierbei können Ansätze aus dem ‚Community Engineering’ einen wertvollen Beitrag leisten.

Im Rahmen der Etablierung einer Roadmap ist es zunächst wichtiger, Hindernisse für einen Austausch über Infrastrukturnutzungen und –veränderungen zu identifizieren. Hierbei können die Starrheit von Abteilungsgrenzen und das notwendige Vorhandensein einer kritischen Anzahl von Benutzern für Community-Konzepte thematisiert werden. Da vor allen Dingen letzteres für kleine und mittelständische Unternehmen ein größeres Problem darstellen kann, muss man sich Überlegen, in diesem Bereich auch Partner außerhalb der Organisation zu suchen. Bei kleinen Betrieben, z.B. im Handwerk, kann es sinnvoll sein, mit entsprechenden Branchen- oder Standesorganisationen zusammenzuarbeiten, um interorganisationale Netzwerke zu Fragen flexibler Infrastrukturentwicklung zu gründen, die es den Mitgliedsunternehmen ermöglichen, die Potentiale, die Flexibilisierungstechnologien und End-User-Development-Maßnahmen bieten, auch auszunutzen.

11 Anhang

11.1 Beitragende und Projekteinbettung

Universität Siegen - Lehrstuhl Wirtschaftsinformatik und Neue Medien
(http://www.wulf.uni-siegen.de; http://www.pipek.uni-siegen.de)
Prof. Dr. Volker Wulf *(volker.wulf(at)uni-siegen.de)*
Juniorprof. Dr. Volkmar Pipek *(volkmar.pipek(at)uni-siegen.de)*
Dipl. Psych. Markus Rohde *(markus.rohde(at)uni-siegen.de)*
Dipl. Inf. Jan Heß *(jan.hess(at)uni-siegen.de)*
Dipl. Inf. Gunnar Stevens *(gunnar.stevens(at)uni-siegen.de)*
Dipl.-Wirt. Inf. Christian Dörner *(christian.doerner(at)uni-siegen.de)*
Björn Borggräfe *(björn.borggräfe(at)uni-siegen.de)*
Mirko Heinbuch *(mirko.heinbuch(at)uni-siegen.de)*
Markus Hofmann *(markus.hofmann(at)uni-siegen.de)*
Moritz Weber *(moritz.weber(at)uni-siegen.de)*

SAP AG *(http://www.sap.de)*
Dr. Roger Kilian-Kehr *(roger.kilian-kehr(at)sap.com)*
Dr. Stefan Scheidl *(stefan.scheidl(at)sap.com)*
Dipl.-Wirtsch.-Inform. Michael Spahn *(michael.spahn(at)sap.com)*
Dipl.-Ing. Todor Stoitsev *(todor.stoitsev(at)sap.com)*

BUHL DATA *(http://www.buhl-data.de)*
Dr.-Ing. Thorsten Vogel *(vogel(at)buhl-data.com)*

Sternjakob GmbH & Co. KG *(http://www.sternjakob.de)*
Herr Günter Beres *(guenter.beres(at)sternjakob.de)*

Strähle+Hess GmbH & Co. KG *(http://www.straehle-hess.de)*
Herr Jörg Nagel *(j-nagel(at)hightex.de)*

Natursteinwerk Schiffer GmbH *(http://www.natursteinschiffer.de)*
Herr Klaus Overbeck *(naturstein.schiffer(at)t-online.de)*

Dachdecker-Meisterbetrieb Heinrich Vißer *(http://www.visser-bedachungen.de)*
Herr Heinrich Vißer *(info(at)visser-bedachungen.de)*

Die Roadmap „Entwicklung hochanpassbarer Systeme“ besteht im Kern aus einer Sammlung von EUD-Methoden und Demonstratoren, die zum Zwecke der Generalisierung in Patternform dargestellt sind. Dieser Katalog von konzeptionellen und technischen Maßnahmen bildet die Grundlage für mögliche EUD-Unterstützungsmaßnahmen. Auf die im „Whitepaper: Endbenutzerorientierte Anpassbarkeit für ERP-Systeme in KMU“ identifizierten allgemeinen Prozessszenarien erfolgt ein Bezug dahingehend, dass EUD-Ansatzpotentiale zur Prozessunterstützung dargelegt sind. Diese Ansatzpotentiale dienen als Grundlage für folgende prototypische Realisierungen.

12 Literatur

Ackerman, M.; Pipek, Volkmar; Wolf, Volker (Hrsg.) (2003). “Sharing Expertise: Beyond Knowledge Management”. MIT-Press, Cambridge, MA.

Alexander, C. (1977). „A Pattern Language. Towns, Buildings, Construction“, Oxford University Press.

Bentley, R. and P. Dourish (1995). „Medium versus Mechanism: Supporting Collaboration through Customisation”. Proceedings of ECSCW 95, Kluwer.

Blackwell, A. F. (2006). „Psychological issues in end-user programming. End User Development“. Springer: 9-30.

Burnett, M., C. Cook, et al. (2004). "End-User Software Engineering". Communications of the ACM 47(9): 53-58.

DiGiano, C. (1996). „Self-Disclosing Design Tools: An Incremental Approach Toward End-User Programming”. University of Colorado.

Dourish, P., J. Lamping, et al. (1999). „Building Bridges: Customisation and Mutual Intelligibility in Shared Category Management”. Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (Group99).

Draxler, S. and G. Stevens (2006). "Getting Out Of A Tailorability Dilemma". INFORMATIK 2006 - Informatik für Menschen : GI-Edition-Lecture Notes in Informatics (LNI) P-93(1): 576 - 580.

Fischer, G. (1996). „Domain-oriented design environments“. Proceedings of the 18th international conference on Software engineering. 517-520

Fischer, G., E. Giaccardi, et al. (2004). "Meta-design: a manifesto for end-user development". Communications of the ACM 47(9): 33 - 37.

Fischer, G. and J. Ostwald (2002). „Seeding, Evolutionary Growth, and Reseeding“. Proc. of PDC 02.

Fischer, G (1994). „Domain-Oriented Design Environments, Automated Software Engineering”. Kluwer Academic Publishers, Boston, MA, 177-203.

Gamma, E., R. Helm, et al. (1995). „Design Patterns. Elements of Reusable Object-Oriented Software“. Addison Wesley.

Gantt, M. and B. A. Nardi (1992). “Gardeners and Gurus: Patterns of Cooperation among CAD Users”. Proceedings of CHI ‘92.

James Jerome Gibson (1979). „The ecological approach to visual perception“. Houghton Mifflin, Boston

Green, T. and M. Perte (1996). "Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework". Journal of Visual Languages & Computing 7: 131–174.

Green, T. R. G. (1989). „Cognitive dimensions of notations. People and Computers. A. Sutcliffe and L. Macaulay“. Cambridge University Press: 443-460.

Grudin, J. (1994). "CSCW: History and Focus". IEEE Computer 27(5): 19-26.

Heineman, G.T., Council, W.T. (2001): „Component-Based Software Engineering: Putting the Pieces Together”. Addison-Wesley Publishing Company, Reading, Massachusetts.

Henderson, A. and M. Kyng (1991). „There's no place like home: Continuing Design in Use. Design at work: Cooperative Design of Computer Systems“. Lawrence Erlbaum Ass.: 219-240.

Jones, M.C., Rathi, D., & Twidale, M.B. (2006). “Wikifying your Interface: Facilitating Community-Based Interface Translation”. In Proceedings of the 6th ACM conference on Designing Interactive, DIS 2006. 321 - 330

Kahler, H. (2001). “Supporting Collaborative Tailoring. Department of Communication, Journalism and Computer Science”. Roskilde, Roskilde University.

Kahler, H. (2001). "More than WORDs - Collaborative Tailoring of a word processor". Journal of Universal Computer Science (7:9), pp 826-847.

Letondal, C. (2006). “Participatory Programming: Developing Programmable Bioinformatics Tools for End-Users”. End User Development, Springer.

Lieberman, H., Ed. (2001). “Your Wish is My Command: Programming by Example”. Morgan Kaufmann.

Lieberman, H., Faaborg, A. (2006): “A goal-oriented web browser”. CHI 2006: 751-760

Lieberman, H., F. Paternò, et al., Eds. (2006). “End User Development”. Springer.

Mackay, W. E. (1990): “Users and customizable Software: A Co-Adaptive Phenomenon”. Doctoral Dissertation, The Massachusetts Institute of Technology.

Mackay, W. E. (1991). “Triggers and Barriers to Customizing Software”. Proceedings of CHI ´91.

MacLean, A., K. Carter, et al. (1990). “User-Tailorable Systems: Pressing the Issues with Buttons”. Proceedings of CHI 90.

Malone, T. W., K.-Y. Lai, et al. (1994). “Experiments with Oval: A Radically Tailorable Tool for Cooperative Work”. Proceedings of CSCW ´92.

Miller, L. A. (1974). "Programming by Non-Programmers". International Journal of Man-Machine Studies 6(2): 237-260.

Mørch, A. (1997). “Three Levels of End-user Tailoring: Customization, Integration, and Extension. Computers and Design in Context”. MIT Press: 51-76.

Myers, B., S. E. Hudson, et al. (2003). “Past, Present, and Future of User Interface Software Tools. Human-Computer Interaction in the New Millennium”. J. Carroll Addison-Wesley.

Myers, B. (1998) “Natural Programming: Project Overview and Proposal”. Technical Report, Carnegie Mellon University School of Computer Science, CMU-CS-98-101 and CMU- HCII-98-100. Pittsburgh.

Oberquelle, H. (1994). “Situationsbedingte und benutzerorientierte Anpaßbarkeit von Groupware. Menschengerechte Groupware - Software-ergonomische Gestaltung und partizipative Umsetzung”. 31-50.

Page, S. (1996); Johnsgard, T.; Albert, U.; Allen, C.: „User Customization of a Word Processor“. in Proceedings of CHI’96, April 13.-18 1996, pp.340-346

Pane, J. F. and B. A. Myers (1996). “Usability Issues in the Design of Novice Programming Systems”. CMU.

Pane, J. F. and B. A. Myers (2006). “More Natural Programming Languages and Environments”. End User Development, Springer: 31-50.

Pane, J. F., C. A. Ratanamahatana, et al. (2001). "Studying the language and structure in non-programmers' solutions to programming problems". International Journal of Human Computer Studies 54(2): 237-264.

Paternò, F., M. Klann, et al. (2003). “Research Agenda and Roadmap for EUD”.

Pipek, V. (2005): "From Tailoring to Appropriation Support: Negotiating Groupware Usage". in: Faculty of Science, Department of Information Processing Science (ACTA UNIVERSITATIS OULUENSIS A 430), University of Oulu, Oulu, Finland, p. 246.

Pipek, V. (2005):"Negotiating Infrastructure: Supporting the Appropriation of Collaborative Software". International Reports on Socio-Informatics (IRSI), Vol. 2 Iss. 1, IISI, Bonn, p. 44 p.

Pipek, V., Kahler H. (2005). "Tailoring together: A systematization and two cases". International Reports on Socio-Informatics 1(2): 1-48.

Pipek, V., and Kahler, H. (2006): "Supporting Collaborative Tailoring". in: End-User Development, H. Lieberman, F. Paterno and V. Wulf (eds.), Springer, Berlin, D.

Rothermel, K. J., C. R. Cook, et al. (2000). “WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation”. Proceedings of the 22nd International Conference on Software Engineering.

Stevens, G. and S. Draxler (2006). “Partizipation im Nutzungskontext”. Konferenzband der Mensch & Computer.

Stiemerling, O. (2000). “Component-Based Tailorability”. Computer Science. Bonn, Universität Bonn.

Teege, G. (2000). "Users as Composers: Parts and Features as a Basis for Tailorability in CSCW Systems". JCSCW 9: 101-122.

Tyre, M.; Orlikowski, W.J. (1994): „Windows of Opportunity: Temporal Patterns of Technological Adaption in Organizations“. in Organization Science, Vol. 5, No.1, 1994, pp. 98-118

Twidale, M., Chanier, T., Pengelly, M., and Self, J. (1992). “Conceptual Modeling” in “Error Analysis in Computer-Assisted Language learning Systems”, chapter “Intelligent Tutoring Systems for Foreign Language Learning“. Berlin.

Wenger, E. (1998): “Communities of Practice - Learning, Meaning and Identity”. Cambridge University Press.

Wulf, V. (1999). "Evolving Cooperation when Introducing Groupware-A Self-Organization Perspective". Cybernetics and Human Knowing 6(2): 55-75.

Wulf, V. (1999). “On Search for Tailoring Functions: Empirical Findings and Implications for Design”. in Proceedings of OZCHI’99, November 28-30, Wagga Wagga, Australia

Wulf, V. and B. Golombek (2001). "Direct Activation: A Concept to Encourage Tailoring Activities". Behaviour & Information Technology 20(4): 249 – 263.

Wulf, V. and M. Rohde (1995). “Towards an Integrated Organization and Technology Development”. Proceedings of the Symposium on Designing Interactive Systems.