



Beiträge zur Forschungsdisziplin Mensch-Computer-Interaktion

Harald Reiterer

Zusammenfassung

Der Beitrag gibt einen Überblick über ausgewählte Forschungsarbeiten des Autors im Bereich der Forschungsdisziplin Mensch-Computer-Interaktion. Diese Arbeiten umfassen Methoden und Gestaltungsprinzipien zur ergonomischen Gestaltung von Software, software-ergonomische Evaluationsverfahren, Entwicklungswerkzeuge für ergonomische graphische Benutzeroberflächen, Wissensmanagementsysteme für Usability Engineers und Visuelle Suchsysteme und sind im Laufe der letzten 15 Jahre entstanden.

1 Einleitung

Der Autor ergreift mit diesem Beitrag zur Festschrift aus Anlass des sechzigsten Geburtstages von Jürgen Krause die günstig erscheinende Gelegenheit, eigene Beiträge zur Forschungsdisziplin Mensch-Computer-Interaktion (MCI) Revue passieren zu lassen. Die Forschungsdisziplin Mensch-Computer-Interaktion war auch dem Jubilar immer ein wichtiges Anliegen, wie seine zahlreichen wissenschaftlichen Arbeiten in diesem Forschungsfeld zeigen (Krause 1991, 1993, 1999). Er hat durch sein Wirken und seine wissenschaftlichen Arbeiten einen wichtigen Beitrag zur Etablierung dieses Faches in der deutschen Informatikszene geleistet! Dafür ist ihm mein Dank gewiss.

Meine Forschungsarbeiten werden im Folgenden ausgewählten Entwicklungen der Forschungsdisziplin MCI zugeordnet und in chronologischer Reihenfolge beschrieben. Damit ergibt sich gleichzeitig ein – zugegeben sehr lückenhafter – historischer Abriss von wesentlichen Entwicklungen im Bereich der Gestaltung der MCI in den letzten 15 Jahren. Im ersten Abschnitt dieses Beitrages wird auf die große Bedeutung der Entwicklung der *graphischen Benutzeroberflächen* für das umfassende Vordringen des Computers in alle Lebensbereiche eingegangen. Als direkte Folge daraus ergab sich, dass sich die Gestaltungsaufgaben des Informatikers im Bereich der Softwareentwicklung maßgeblich veränderten. Einige dieser Veränderungen werden in den folgenden Abschnitten vorgestellt. Zuerst wird gezeigt, dass aus der *Softwaregestaltung* eine wesentlich umfassendere *Arbeitsgestaltung* geworden ist. Fragen, welchen Beitrag die MCI zur *Qualitätssicherung von Softwarepro-*



dukten leisten konnte, bestimmte vor allem die Forschung in den 80er und frühen 90er Jahren. Diese produktorientierte Betrachtungsweise wurde dann in den 90er Jahren durch eine verstärkt prozessorientierte Betrachtungsweise erweitert, in der man untersuchte, wie der Softwareentwicklungsprozess werkzeugmäßig unterstützt und methodisch ergänzt werden muss, damit die resultierenden Softwareprodukte auch software-ergonomischen Qualitätsstandards entsprechen. Damit wurde der Disziplin Software Engineering die Disziplin *Usability Engineering* zur Seite gestellt. Im letzten Abschnitt dieses Beitrages werden einige aktuelle Entwicklungen im Bereich MCI aufgegriffen und vor allem Ansätze zur *Visualisierung großer Datenbestände* vorgestellt. Ein Forschungsgebiet, das Professor Krause durch sein Mitwirken am Aufbau des Studienganges Computervisualistik an der Universität Koblenz ebenfalls maßgeblich in Deutschland etabliert hat.

2 Der Siegeszug der graphischen Benutzeroberfläche

Die ersten Schritte des Autors in das spannende Forschungsfeld MCI erfolgten im Jahre 1988 am Institut für Statistik und Informatik der Universität Wien unter der Leitung von Prof. Dr. A Min Tjoa. Im Rahmen eines Projektes wurde ein Handbuch zur menschengerechten Gestaltung von EDV-Systemen im Bürobereich entwickelt (Tjoa, Kolm & Koch 1990). Ziel dieses Vorhabens war es, ein Vorgehensmodell sowie Methoden und Gestaltungsrichtlinien zur ergonomischen Gestaltung von Software im Bürobereich zu erarbeiten, die vor allem Softwareentwickler als Handreichung dienen sollten. Als Vorbild dieses Vorhabens dienten Handbücher zur hardware-ergonomischen Gestaltung von Bildschirm- und Büroarbeitsplätzen, die in den 80er Jahren sehr populär waren.

Die graphische Benutzeroberfläche, auch GUI für Graphical User Interface genannt, war aufgrund der Verfügbarkeit von grafikfähigen Bildschirmen sowie Dank des Erfolges des Apple Macintosh gerade dabei, die bisherigen Formen der MCI zu revolutionieren. Bis dahin beherrschten alphanumerische Benutzeroberflächen – meist mit einfacher Menüführung bzw. mit kommandosprachlicher Dialogführung – die Szene, die auf so genannten unintelligenten Terminals (sprich Endgeräten, die über keine Möglichkeit der Graphikdarstellung verfügten) betrieben wurden. Die GUIs in Kombination mit Zeigeinstrumenten (z.B. der Maus) eröffneten völlig neuen Benutzergruppen den Zugang zum Computer. Die umfassende Verbreitung von so genannten Endbenutzerwerkzeugen (z.B. Textverarbeitungs-, Graphik- oder Tabellenkalkulationsprogrammen) war erst vor dem Hintergrund des großen Erfolges der GUIs bei den Benutzern möglich. Damit konnten eine Vielzahl von Arbeitsplätzen mit Computern ausgestattet werden, die bisher von den Möglichkeiten des

Computers nicht unmittelbar profitieren konnten. Die wesentlichen Vorteile der graphischen Benutzeroberfläche basierten auf einer Reihe von Faktoren. Die mentale Belastung des Benutzers wurde verringert, da er nicht durch verschachtelte Menüstrukturen navigieren bzw. keine kryptischen Kommandosprachen erlernen musste. Alle für die Interaktion mit dem Computer erforderlichen Informationen werden mittels Fenster in Kombination mit graphischen Menüs (Hauptmenübalken mit Pull-down Menüs) und Icons angezeigt. Damit reduzierte sich die mentale Belastung des Benutzers auf das (Wieder-)Erkennen der Bedeutung von Menüeinträgen bzw. Icons. Durch die Möglichkeit der direkten Manipulation von Objekten am Bildschirm mit einem Zeigeelement wie es beispielsweise die Maus darstellte, konnten grundlegende Kommandos auf intuitiv erfassbare, direkt-manipulative Art und Weise ausgeführt werden (z.B. das Verschieben eines Files von einem Folder in einen anderen Folder).

Das massive Vordringen des Computers – wie gezeigt unter anderem auch Dank des Erfolges neuer Möglichkeiten der Mensch-Computer-Interaktion – bewirkte auch eine Reihe von Konsequenzen, mit denen der typische Informatiker bis dahin nicht konfrontiert worden war und die neue Lösungswege erforderten. Dies soll anhand der folgenden Sachverhalte exemplarisch verdeutlicht werden:

- Softwaregestaltung ist auch Arbeitsgestaltung.
- Die Softwarequalitätssicherung erforderte neue Methoden zur softwareergonomischen Qualitätssicherung.
- Die werkzeuqmäßige Unterstützung des Interaktionsdesigns bei der Softwareentwicklung und die Ergänzung des Software Engineering um Methoden des Usability Engineering.

3 Softwaregestaltung ist auch Arbeitsgestaltung

Das umfassende Vordringen des Computers in viele Anwendungsbereiche eines Unternehmens oder einer Behörde bewirkte, dass mit dem Einsatz des Computers auch die bisherigen Aufgabenverteilungen und oft auch die bisherige Organisationsgestaltung neu überdacht werden mussten, um die Potentiale dieses neuen Werkzeuges voll ausschöpfen zu können. Dies war bei den bis in die frühen 80er Jahre dominierenden zentralen Rechenzentren in den Unternehmen nicht der Fall, da die Auswirkungen des Einsatzes von Computern für die meisten Arbeitsplätze nur indirekte Folgen hatten (meist in Form von dicken, schwer verständlichen Computerausdrucken, die in regelmäßigen Abständen auf den Schreibtisch landeten). Jetzt konnten aber aufgrund den einfach zu bedienenden, auf leistungsfähigen Datenbanken basierenden Anwendungsprogrammen ganze Geschäftsprozesse in Software abgebildet und un-

mittelbar am Arbeitsplatz bereitgestellt werden. Dies war in der Regel keine simple 1:1 Abbildung der bisherigen Geschäftsprozesse, sondern der Computer eröffnete ganz neue Möglichkeiten der Aufgaben- und Organisationsgestaltung. Typische Schlagworte dieser Zeit waren die „Rundumsachbearbeitung“, „One-Face-to-the-Customer“ oder „Business Process Re-Engineering“. Dadurch entstanden Arbeitsplätze mit völlig neuen Aufgabenzuschnitten und der jeweilige Stelleninhaber bekam leistungsfähige Programmpakete auf seinem Computer zur Verfügung gestellt, die ihm die Erledigung, in sich geschlossener Geschäftsvorfälle, ermöglichten. Vielen Unkenrufen zum Trotz, eröffnete der Einsatz des Computers damit eine bisher nicht da gewesene, anspruchsvollere und damit menschengerechtere Aufgabengestaltung an vielen Arbeitsplätzen im Büro- und Verwaltungsbereich. Diese neuen Aufgaben zeichneten sich beispielsweise durch Eigenschaften wie Ganzheitlichkeit, Persönlichkeits- und Lernförderlichkeit aus. Dies bedeutete aber auch für den Softwareentwickler, dass er im Zuge der Entwicklung und Einführung von Anwendungsprogrammen in Gestaltungsbereiche vorstieß, die bisher nicht die seinen waren. Gerade die MCI hat diese Problematik früh erkannt und als Forschungsdisziplin versucht, Methoden, Werkzeuge und auch Gestaltungsprinzipien zu entwickeln, die den Informatiker in die Lage versetzten auf diese neuen Möglichkeiten der Softwaregestaltung adäquat zu reagieren. Dies führte dazu, dass die klassischen Methoden des Software Engineering durch Methoden und Techniken des so genannten Usability Engineering ergänzt wurden, beispielsweise um Methoden der Aufgabenanalyse oder des Work Re-Engineering. Bei diesen Methoden steht nicht die adäquate Gestaltung der Daten oder der Funktionen im primären Blickfeld, sondern hier wird versucht, unter Berücksichtigung der Organisations- und Aufgabenerfordernisse sowie der Charakteristika der zukünftigen Benutzer, gebrauchstaugliche Software zu entwickeln unter *gleichzeitiger* Ausschöpfung der Gestaltungspotentiale, hinsichtlich einer ergonomischen Aufgaben- und Organisationsgestaltung. Gebrauchstaugliche und aufgabenangemessene Software zeichnet sich dadurch aus, dass sie sowohl effektiv (sprich für den jeweiligen Aufgabenbereich die erforderlichen Funktionen bietet) als auch effizient (sprich unter Minimierung der notwendigen Dialogschritte) genutzt werden kann. Zusätzlich sollte der Benutzer auch ein subjektives Gefühl der Zufriedenheit (Joy of Use) bei der Benutzung des Arbeitsmittels Computer empfinden.

Die sich aus oben dargestellten Sachverhalten ergebenden Konsequenzen für die Softwaregestaltung wurden Anfang der 90er Jahre vom Autor in einer Monographie mit dem Titel „Software-Ergonomie“ veröffentlicht (Koch, Reiterer & Tjoa 1991). Es war zum damaligen Zeitpunkt eines der wenigen deutschsprachigen Bücher, das sich dieser Thematik angenommen hatte. Das dort vorgestellte Vorgehensmodell (z.B. iterativ, prototyping-orientiert und mit Einbeziehung der Benutzer in die verschiedenen Phasen der Soft-

wareentwicklung), die empfohlenen Methoden (z.B. zur Aufgaben- und Organisationsgestaltung) und die grundlegenden software-ergonomischen Gestaltungsprinzipien haben auch nach über 10 Jahren noch ihre Gültigkeit!

4 Software-ergonomische Qualitätssicherung

Methoden der Qualitätssicherung, basierend auf unterschiedlichen Metriken, haben eine lange Tradition im Bereich der Software-Entwicklung (z.B. Inspektion, Review, Walkthrough). Da aber durch den Erfolg der graphischen Benutzeroberflächen die Programmierung des Teiles eines Anwendungsprogramms, der für die Interaktion zwischen dem Benutzer und der Anwendung zuständig ist, immer wichtiger wurde, zeigte sich, dass die klassischen Qualitätssicherungsverfahren sowohl vom Standpunkt der Metriken als auch hinsichtlich der verfügbaren Testmethoden Defizite aufwiesen. Dies vor dem Hintergrund der bis dahin bestehenden geringen Bedeutung der MCI für den Erfolg eines Programms auch nicht weiter verwunderlich, zumal die Benutzer in der Regel Anwendungsspezialisten waren. So standen bis dahin vor allem Fragen der Performance, der Portabilität, der Korrektheit eines Programms im Vordergrund. Das Kriterium der Benutzbarkeit bzw. der Gebrauchstauglichkeit (Usability) gewann jetzt aber maßgeblich an Bedeutung. Der Autor beteiligte sich daher 1988 an einem Forschungsprojekt der damaligen Forschungsgruppe Mensch-Maschine-Kommunikation der GMD in St. Augustin / Bonn¹, das unter der Leitung von Prof. Dr. Reinhard Oppermann stand und das die Entwicklung eines Testverfahrens zur Bewertung der software-ergonomischen Qualität von Software zum Ziel hatte. Das Verfahren trug den Namen EVADIS (Evaluation von Dialogsystemen) und war eines der ersten software-ergonomischen Testverfahren. Es wurde 1992 in einer überarbeiteten Version in Buchform veröffentlicht (Mayhew 1999). Anhand von ergonomischen Dialogprinzipien (z.B. Erwartungskonformität, Selbstbeschreibungsfähigkeit, Fehlerrobustheit) und von Prüfitems konnte eine Beurteilung der software-ergonomischen Qualität eines Programms erfolgen. Es handelt sich dabei um ein heuristisches Evaluationsverfahren, das von einem Experten (z.B. Usability Engineer) durchgeführt wird. Das Verfahren wurde vor allem sehr erfolgreich in der Lehre für Studierende der Informatik bzw. Wirtschaftsinformatik eingesetzt, da es die Studierenden in praktischer Art und Weise mit den wesentlichen ergonomischen Dialogprinzipien und den dabei zu beachtenden Gestaltungsregeln vertraut machte. Dies führte auch dazu, dass das Evaluationsverfahren EVADIS von der GI im Rahmen des von ihr vorgelegten Software-Ergonomie Curriculums als Evaluationsverfahren für die Lehre empfohlen wurde.

¹ Heute Fraunhofer-Institut für Angewandte Informationstechnik (FIT).

5 Die werkzeugmäßige Unterstützung des Interaktionsdesigns

Die intensive Beschäftigung des Autors mit Fragestellung der software-ergonomischen Qualitätssicherung hatte gezeigt, wie wichtig die Unterstützung der Anwendungsentwickler im eigentlichen Entwicklungsprozess mit software-ergonomischem Gestaltungswissen ist. Nicht erst wenn der Entwicklungsprozess abgeschlossen ist, sollte die Unterstützung der Entwickler einsetzen (z.B. durch Evaluationsverfahren), sondern bereits im Entwicklungsprozess selbst. In einem nachfolgenden Forschungsprojekt IDA (Interface Design Assistance) entwickelte der Autor als Gastforscher in der damaligen Forschungsgruppe Mensch-Maschine Interaktion der GMD in St. Augustin / Bonn² ein User Interface Management System (UIMS), das über eine Reihe von Funktionen verfügte, die den Programmierer bei der Realisierung von ergonomischen GUIs unterstützen (Reiterer 1996). Abbildung 1 zeigt das als Basis genutzte UIMS namens Dialog Manager der ISA GmbH (Stuttgart) und die zusätzlich mittels einer „IDA-Toolbar“ angebotenen Funktionen.

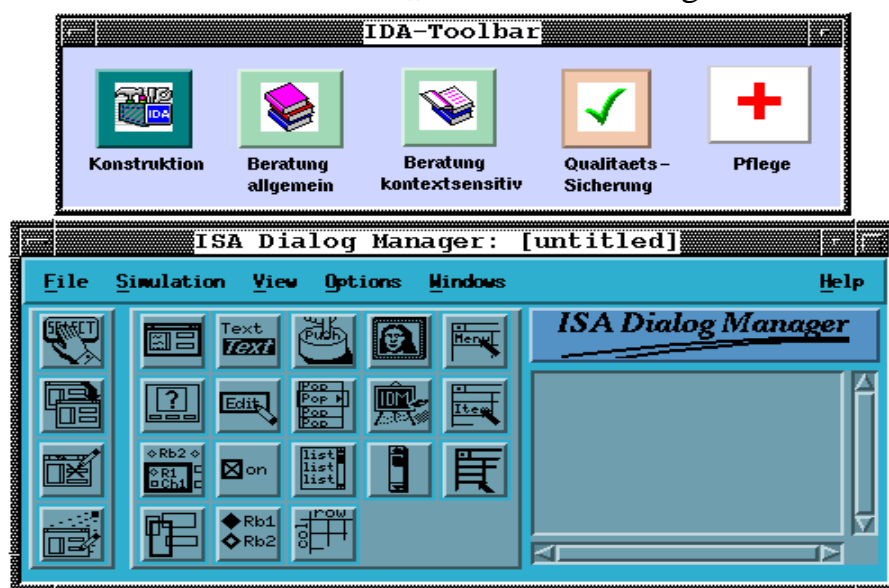


Abbildung 1: IDA – Interface Design Assistance (Stand 1995)

Dazu gehörte eine objekt-orientierte Bibliothek von wiederverwendbaren Dialogbausteinen (Konstruktion genannt), ein hypermediales Beratungssystem und eine wissensbasierte Qualitätssicherung. Die Bibliothek beinhaltete eine Reihe von anwendungsspezifischen Dialogbausteinen, die nicht nur das statische Look eines GUIs umfassten (z.B. ein Dialogfenster zum Anlegen eines neuen Kunden), sondern auch das dynamische Feel (z.B. das Öffnen weiterer Dialog- oder Mitteilungsfenster, die mit dem ursprünglichen Dialogfenster in einem semantischen Zusammenhang stehen). Die Dialogbausteine wurden

² Heute Fraunhofer- Institut für Angewandte Informationstechnik (FIT).

nach software-ergonomischen Prinzipien für bestimmte Anwendungsdomänen entwickelt und ermöglichten dem Anwendungsentwickler auf wiederverwendbare Dialogkomponenten zurückzugreifen. Das hypermediale Beratungssystem war in der Lage, Anfragen des Entwicklers zu beantworten (z.B. Wann setze ich sinnvollerweise ein Notebook ein?) und lieferte in anschaulicher Form Erklärungen anhand von interaktiven Beispielen (mit Text, Ton, Animation und Interaktion). Siehe dazu Abbildung 2.

Die wissensbasierte Qualitätssicherung stellte eine Art software-ergonomischen Debugger dar. Er konnte software-ergonomische Regelverstöße erkennen, sowie automatisch korrigieren und lieferte die notwendigen Erklärungen unter Einsatz des hypermedialen Beratungssystems. Siehe dazu Abbildung 3. Die Qualitätssicherung basierte auf einem Expertensystem, welches ergonomische Gestaltungsregeln in seiner Regelbasis beinhaltete und mittels seiner Inferenzmaschine diese Regeln auf das jeweilige GUI anwenden konnte.

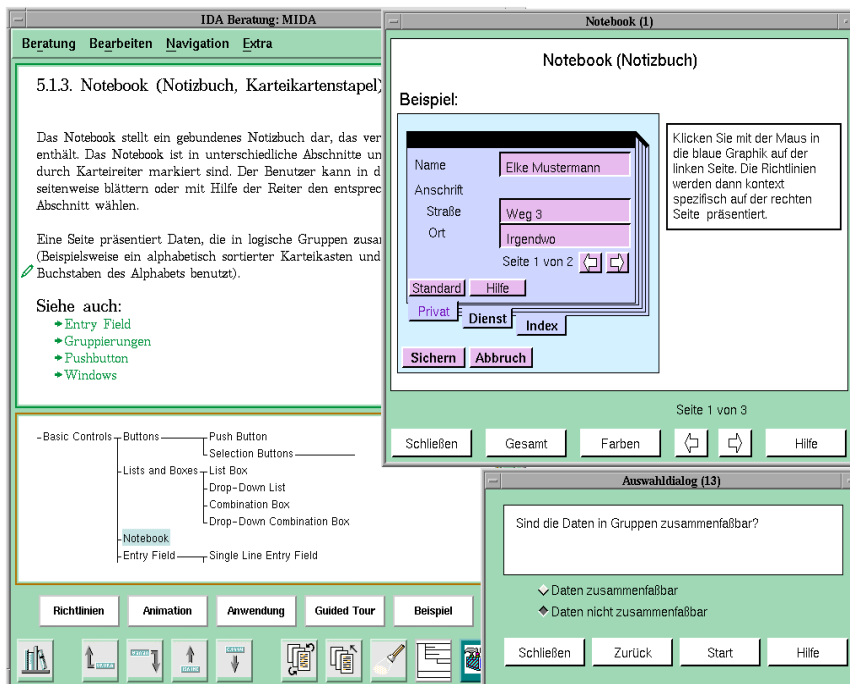


Abbildung 2: IDA – hypermediales Beratungssystem (Stand 1995)

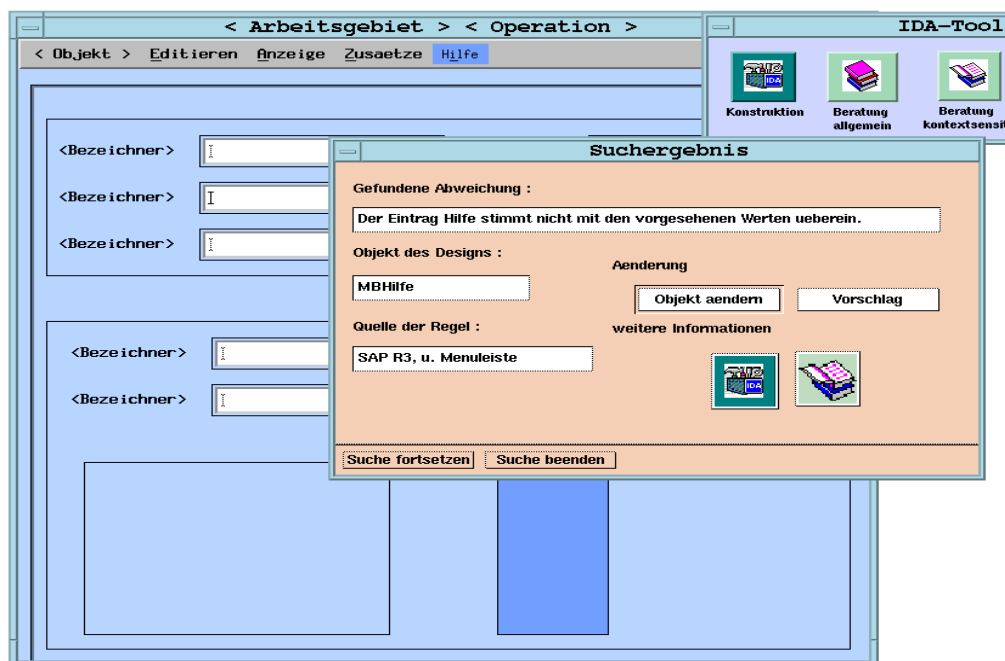


Abbildung 3: IDA – Qualitätssicherung (Stand 1995)

Mitte der 90er Jahre war IDA eines der wenigen Forschungssysteme im deutschsprachigen Raum, das auf nationalen und internationalen Konferenzen gezeigt werden konnte (Bachmann et al. 1995, Reiterer 1994a, 1994b, 1995a, 1995b).

Der Siegeszug der Programmiersprache Java, die zunehmende Verbesserung der Ausbildung der Anwendungsentwickler im Bereich der Entwicklung von GUIs sowie der Trend zu einheitlichen Programmierumgebungen (keine Trennung mehr in UIMS zur Entwicklung des GUIs und in Programmierwerkzeug zur Entwicklung des Anwendungscodes) haben dazu geführt, dass UIMS heute nur mehr eine geringe Bedeutung haben. Wiederverwendbare Bibliotheken von Dialogbausteinen und online verfügbare Style Guides (z.B. Java Look and Feel Design Guidelines³) sind heute nicht mehr wegzudenken.

Neben der werkzeuggestützten Unterstützung des Anwendungsentwicklers bei der Programmierung des GUIs wurden in den folgenden Jahren im Bereich MCI wesentlich umfassendere, methodische Unterstützungsmaßnahmen entwickelt. Diese beschränkten sich nicht mehr primär auf die Qualitätssicherung durch die Entwicklung ausgereifter Evaluationsverfahren oder von Werkzeugen zur Unterstützung des Implementierungsprozesses, nein, der gesamte Softwareentwicklungsprozess wurde zum Gegenstand der Forschung. Unter dem Schlagwort Usability Engineering wurden eigene Vorgehensmodelle

³ Sun Microsystems Inc. (2001). Java Look and Feel Design Guidelines. Second Edition. <http://java.sun.com/products/jlff/ed2/book/> [Zugriff September 2004].

entwickelt, die sich durch Eigenschaften wie iterativ, prototyping-orientiert und evaluationszentriert auszeichnen. Ergänzend wurden eine Vielzahl von Methoden zur Erfassung der Anforderungen (vergleichbar dem Requirement Engineering) entwickelt, die für die erfolgreiche Umsetzung der MCI maßgeblich sind. Als Stichworte seien hier genannt: Essential Use Cases, Hierarchische Aufgabenanalyse, Aufgabenszenarien, Fragebögen zur Erfassung von Benutzereigenschaften, Contextual Design, Participatory Design⁴. Auch spezielle Methoden zur Unterstützung der eigentlichen Entwurfsarbeiten wurden entwickelt. Auch hier mögen die folgenden Stichworte zur Verdeutlichung genügen: Designszenarien, Prototyping mittels Mock Ups oder Storyboards, HCI Patterns, Gestaltungsprinzipien und Normen.⁵

Aktuelle Arbeiten des Autors in diesem Forschungsbereich versuchen die Softwareentwickler während des gesamten Entwicklungsprozesses mit dem notwendigen Methodenwissen (im Sinne von base practicies) aber auch Erfahrungswissen aus vorangegangenen Projekten (so genannten best practicies) zu unterstützen. Dazu kooperierte der Autor mit der Forschungsabteilung der Daimler Chrysler AG in Ulm im Rahmen eines Projektes namens PROUSE (Process centred Usability Engineering Environment), das unter der Leitung von Michael Offergeld und Richard Oed durchgeführt wurde (Metzker & Reiterer 2002a, 2002b). PROUSE stellt eine Art Wissensmanagementsystem für Usability Engineers dar, in dem neben dem notwendigen Methodenwissen auch so genannte Erfahrungspakte zur Verfügung gestellt werden. Dazu wird Erfahrungswissen semi-formal beschrieben und in einem Repository verwaltet. Durch geeignete Mechanismen, wie beispielsweise Filterwerkzeuge, die unter Berücksichtigung der jeweiligen Rahmenbedingungen eines Projektes nur bestimmte Methoden bzw. Erfahrungspakte vorschlagen oder einem ausgefeilten statistischen Bewertungsverfahren, das auf den Beurteilungen der Benutzer der Wissensinhalte basiert, werden dem Benutzer des Systems maßgeschneiderte Wissensbausteine zur Verfügung gestellt. Diese beinhalten auch eine Reihe von Artefakten (z.B. Fragebögen, HCI Patterns, Templates, Codebeispiele), die vom Usability Engineer unmittelbar bei seiner Arbeit genutzt werden können. Abbildung 4 zeigt ein derartiges Erfahrungspaket, wie es sich dem Benutzer von PROUSE präsentiert.

⁴ Einen aktuellen und umfassenden Überblick über die Methoden des Usability Engineering gibt das sehr lesenswerte Buch zu diesem Thema von D. Mayhew (Mayhew 1999).

⁵ Einen guten Überblick zum aktuellen Stand des Wissens im Forschungsbereich HCI findet man in Preece, Rogers & Sharpe 2002.

