Proseminar: *Human Computer Interaction*

Prof. Jan Borchers

Lehrstuhl für Medieninformatik X

Sommersemester 2005

Design for People at Work

Nina Hilke

Ann Malzkorn

Inhalt

1. Einleitung

1.1 Einleitung

Es gibt viele Menschen, die den Computer als Nachteil erleben, weil er in Lebensbereiche eindringt und sie gezwungen sind, sich anzupassen. Für andere Menschen eröffnet die Welt der Computer viele neue und spannende Möglichkeiten. Diese unterschiedlichen Auffassungen können vor allem in der Arbeitswelt zu Problemen führen. Genau diese Probleme untersucht Sarah Kuhn zusammen mit ihren KollegInnen bei der industriellen Anwendung von computergestützten Systemen. Sie arbeiten eng mit den ArbeiterInnen und Angestellten zusammen, die täglich mit computergesteuerten Maschinen arbeiten müssen. In ihrem Text zeigt sie einen Weg, wie man Softwaresysteme entwickeln kann und dabei komplexe soziale und politische Faktoren mit einbezieht. Denn diese Faktoren spielen eine große Rolle bei der Planung und Entwicklung, und ganz besonders für Erfolg oder Mißerfolg eines Systems.

Beim gestalten[1] von Arbeitsplätzen muss der/die DesignerIn sich mit den verschiedenen Parteien (ArbeiterInnen bzw. Angestellte, ManagerInnen) auseinandersetzen, die für diesen Prozess eine Rolle spielen, und deren Interessen vielfach sehr unterschiedlich sind.
Sarah Kuhn befürwortet das Human-centered Design (vgl. Kap.6) und stellt die Methode des Participatory Designs vor.
Im Anschluss daran wird es noch einen Überblick über das gesamte Buch geben und eine Reflexion darüber, welche Erkenntnisse gewonnen wurden. Danach werden zudem die Entwicklungen bis heute vorgestellt.

1.2 Sarah Kuhn

1987 erhielt Sarah Kuhn ihren Doktortitel. 1997 arbeitete sie als Junior-Professorin im Fachbereich *Policy and Planning* an der University of Massachusetts in Lowell. Mittlerweile ist sie außerordentliche Professorin (associate professor) im Fachbereich *Regional Economics and Social Development*. Dieser Fachbereich ist interdisziplinär angelegt und beschäftigt sich mit politischen Wissenschaften, Geschichte, Volkswirtschaft, aber auch Psychologie, Soziologie und Stadtplanung. Die meisten Kurse werden von Teams

1 Im Sinne der von Mitchell Kapor geforderten Berufsbezeichnung des Softwaredesigners fanden wir es schwierig, ein dem englischen verb „to design“ entsprechendes deutsches Wort zu finden. Daher verwenden wir vorzugsweise gestalten, entwickeln oder entwefern.

unterrichtet, um so den Studenten verschiedene Blickwinkel auf das gleiche Problem zu geben.

Sarah Kuhn hat in Zusammenarbeit mit Mitchell Kapor und William Mitchell am MIT studiobasierte Kurse in Software Design entwickelt. Zurzeit lehrt sie *Software Design in Context*.

Ihre Forschungen konzentrieren sich auf das Benutzen und das Design von Informationstechnologie. Sie beschäftigt sich mit Arbeitsprozessen und der Bedeutung bzw. den Auswirkungen von Computertechnologie auf den Arbeitsplatz. Diese Interessen führten dazu, dass sie sich auch für das Design von Informationstechnologie interessiert und vor allem dafür wie Experten ausgebildet werden.

Im Jahr 2000/01 war sie Mitglied beim Radcliffe Institute an der Universtät von Harvard und beschäftigte sich mit dem Projekt Design as Work/Work as Design. Dieses Projekt untersuchte, ob sich der Designprozess mit Problemen des Geschlechts, der Gemeinschaft und der wirtschaftlichen Entwicklung auseinandersetzt. Darüber hinaus hat sie zahlreiche Preise und Auszeichnungen für ihre Arbeiten erhalten. So zum Beispiel 1997 den Teaching Award von der University of Massachusettes. Zusätzlich war sie Mitglied im Vorsitz bei der Participatory Design Conference 1998.

Zur Zeit beschäftigt sie sich hauptsächlich mit einer Studie bezüglich Frauen und Männern im Umgang Software- und Internetarbeitsplätzen. So hat sie 2003 das Buch *What makes Dick und Jane run? Examining the retention of women and men in the software and internet industry* veröffentlicht. Ebenso interessiert sie sich für den Einsatz von *Design Studios* und Projekt-Unterricht in den Computer- und Ingenieurwissenschaften.

2. Design for People at Work

Für viele Menschen bedeuteten Technologien am Arbeitsplatz einen hohen Gewinn an Effizienz und Selbstbestimmung. Dies mag für Selbständige, die sich die Software mit der sie arbeiten möchten aussuchen können, sicherlich gelten, allerdings machen diese nur einen kleinen Teil der arbeitenden Bevölkerung aus. Der Rest hat in solchen Dingen keine Entscheidungskompetenz, sondern muss auf das zurückgreifen, was oft von höheren Instanzen spezifiziert wird.

Wo gut gestaltete Arbeitsplätze und Systeme die Arbeit enorm vereinfachen, die Produktivität und Zufriedenheit der Angestellten erhöhen, kann es im Gegenzug eine ernüchternde Erfahrung sein, mit Systemen arbeiten zu müssen, die nicht an ausschlaggebende Arbeitsprozesse angepasst sind.

Die Effekte die ein computergesteuertes System auf die Arbeit haben kann, hängen ganz stark davon ab welches geistige Konzept die SoftwaredesignerInnen von der Arbeit haben.

Entscheidend für den Erfolg eines Systems ist für Sarah Kuhn vor allem die Annahme über die jeweiligen Arbeit, die in das System eingebettet ist. Je mehr dieses Konzept mit der Realität der auszuführenden Arbeit übereinstimmt, desto erfolgreicher ist es. Andererseits kann ein System, dass den Arbeitenden viele Probleme bereitet ein nicht zu unterschätzender Faktor für Unproduktivität, Unzufriedenheit und eine erhöhte Anzahl von gesundheitlichen Problemen sein.
Dabei geht Sarah Kuhn nicht nur vom klassischen Computer im Büro, sondern vor allem auch von automatisierten Arbeitsschritten wie sie in Fabriken und Werkshallen vorkommen, aus.

2.1 Konzepte von Arbeit

Wenn der Kontext der Arbeit nicht richtig verstanden wird, spiegelt sich dies im späteren System wieder und hat negative Folgen für die ArbeiterInnen und für die Produktivität. Das Softwaresystem repräsentiert die Arbeit und das, was in ihr relevant erscheint. So ist im Beispiel der Telefongesellschaft die Zeit, die bei der Reparatur verbracht wird weitaus wichtiger als z.B. das Ausbilden neuer Arbeiter. Die Daten, die vom System gesammelt werden, haben einen vermeintlich objektiven Charakter und somit großes Gewicht für die Manager. Auch wenn die Daten die Realität nur bedingt wiedergeben können.
Bei der Betrachtung von Arbeit kann man den *organizational* bzw. *explicit view* dem *activty-orientated* bzw *tacit view* gegenüberstellen. Der *explicit view* beschreibt Arbeit als eine Reihe von Handlungen, die genau definiert werden können, z.B. in Handbüchern. Der *tacit view* hingegen sieht Arbeit als eine komplexe Mischung aus Aktivität, Kommunikation und Koordination, welche ständig von den ArbeiterInnen und ManagerInnen beeinflusst wird. Man kann sagen, dass Arbeit immer aus Teilen beider Ansichten besteht. Dabei hängt die Effizienz von Arbeit nicht von wohldefinierten Arbeitsschritten ab, sondern vielmehr von der Kompetenz der ArbeiterInnen schwierige und komplexe Probleme zu lösen. Oft wird bei Verbesserungsmaßnahmen (die ja nur zu oft vom Management initiiert und durchgeführt werden) nur der *explicit view* betrachtet, was es den ArbeiterInnen erheblich erschwert, ihre Arbeit vernünftig zu machen. Sie entwickeln dann Hilfsmechanismen um sich die Arbeit zu erleichtern, wie es etwa beim Big Bank Problem der Fall war, wo eine Authentifikationskarte von verantwortlichen Personen an die Angestellten weitergegeben wurde.
Dieses Problem der Ineffizienz entsteht aber schon beim Design Prozess. Also muss auch hier etwas verändert werden.

2.2 Demokratischer Ansatz zur Elimination der Kosten von schlechtem Design

Systeme, die ohne Beachtung der gegebenen Arbeitssituation entwickelt werden, verursachen sehr hohe Kosten; nicht nur weil die Produktivität sinkt, sondern auch weil erhebliche Nachteile für die ArbeiterInnen entstehen. Diese Probleme entstehen einerseits durch Ersetzung und Intensivierung, aber auch durch Überwachung und Reduzierung der Anforderungen. Die Automatisierung führt dazu, dass weniger Menschen gebraucht werden um die gleiche Arbeit zu erledigen. Das führt natürlich zu großer Unsicherheit bezüglich der Arbeitsplatzsituation. Ende der 90er stieg in den USA die Zahl der Halbtagsbeschäftigten und gleichzeitig sanken der durchschnittliche Verdienst und die Zuschüsse zur Krankenversicherung. Der große Konkurrenzdruck führt dazu, dass die Menschen immer länger und härter arbeiten müssen. Überwachung durch Elektronische Systeme führt zu erhöhtem Streß für die Angestellten. Außerdem benötigen die Angestellten immer weniger Kompetenzen, da ein großer Teil ihrer Arbeit von Computern erledigt wird. Dadurch verschwinden manche Berufe vollständig. Hinzu kommt, dass die Angestellten die Kontrolle über ihre Arbeit verlieren und sie weniger selbständig werden. Darüber hinaus hat das stressige, routinierte Arbeiten auch gravierende Folgen für die Gesundheit.

Probleme für den/die DesignerIn können auftreten, wenn es zur Firmenphilosophie gehört, dass die Arbeitsbedingungen für die Angestellten schlecht sind: der/die DesignerIn gerät in die Schußline zwischen den beiden Parteien.

In Europa ist es weit verbreitet, dass die Arbeiter ein Mitbestimmungsrecht haben – etwa durch Gewerkschaften und Betriebsräte. Zudem gibt es Gesetze und Richtlinien, die die Qualität des Arbeitsplatzes bestimmen. Das hilft den DesignerInnen, weil es Rahmenbedingungen schafft. In den USA ist man jedoch noch nicht so fortschrittlich. Dennoch gibt es zahlreiche Bestrebungen, die Situation der ArbeiterInnen und Angestellten zu verbessern. Es wird zum Einen die sogenannte *Worker´s Bill of Right* gefordert, welche angelehnt an die Bürgerrechte der Verfassung der Vereinigten Staaten gewährleisten soll, dass Angestellte auch während der Arbeitszeit ihre Grundrechte behalten. Genauso setzen sich viele Gewerkschaften für die Rechte der Arbeiter ein, auch wenn es in den USA noch einige weniger Gewerkschaften als in Europa gibt. Zusätzlich wird gefordert, dass jede/r DesignerIn einen *Code of Ethics* unterschreiben muss, welcher dann Richtlinien für deren Arbeit vorgibt. Denn gerade die DesignerInnen haben eine sehr große Verantwortung gegenüber den Menschen, da computerbasierte Systeme einen immensen Einfluss auf das Leben und Arbeiten der Menschen haben – insbesondere wenn sie, wie oben ausgeführt, nicht frei gewählt werden können.

2.3 Schlechte Beispiele

Um einen differenzierten Blickwinkel für die Probleme zu bekommen, die nach der Installation eines Systems in der Arbeitswelt auftreten können, legt uns Sarah Kuhn einige Beispiele von Systemen dar mit denen es gravierende Konflikte gab.

2.3.1 Fall 1: Das Trouble-Ticket-System

Das Trouble-Ticket-System (TTS) wurde in den späten 80ern für eine Telefongesellschaft entwickelt. Das großrechnerbasierte System wurde für die Ablaufkoordination, Arbeitspläne und das Aufzeichnen von Problemen und Fortschritten benutzt. Das System arbeitete wie folgt: Bei einer Fehlermeldung wurde ein Job-Ticket erstellt und an die zuständige Telefongesellschaft gesendet. Dort nahm sich ein/e ArbeiterIn des Tickets an und erledigte den Job; war dieser getan oder konnte nicht mehr getan werden, wurde das Ticket zurück an die zentrale TTS gesendet.
Bevor das TTS zum Einsatz kam, gab es schon Job-Tickets. Der Einsatz war allerdings ein wenig anders: die Arbeit wurde geteilt, die ArbeiterInnen berieten sich gegenseitig und lösten die Probleme nicht alleine. So konnten die verschiedenen Arbeitsbereiche aufgesplittet werden und spezielle Kompetenzen von anderen TesterInnen genutzt werden. Allerdings war der Kontakt zu MittersterInnen eine der Hauptmotivationen, das TTS zu entwickeln – einem Übermaß an Konversation und Kontakten sollte vorgebeugt werden um sicherzustellen, dass die TesterInnen ihre Zeit auch wirklich mit arbeiten bzw. reparieren verbrachten. Gespräche wurden als ineffizient und nicht sachdienlich erachtet. So arbeitet der/die TesterIn im TTS alleine; wenn der Job nicht vollständig erledigt werden konnte, wurde das schriftlich festgehalten und das Ticket zum TTS zurückgeschickt, damit sich ein/e nächste/r TesterIn darum kümmern konnte. Mit der Einführung des TTS wurde nun aber auch ein großer Vorteil der Kommunikation unter den TesterInnen unterbunden, so z.B. die Vernetzung und die Möglichkeit, von den Kompetenzen der KollegInnen zu profitieren. Ein weiterer kritischer Punkt war die Überwachung von Kontakten mit KollegInnen – also Beratungen untereinander oder die Einarbeitung neuer MitarbeiterInnen – die aus Sicht des Management zu einem Verlust an Arbeitseffizienz führte und tatsächlich auch sanktioniert wurden. Resümee ist also, dass das System vorwiegend Nachteile mit sich brachte: statt einer effizienten Arbeitsweise wurde das Team demotiviert, die KollegInnenschaft aufgebrochen und die Zeit für das Bearbeiten eines Jobs ausgeweitet, anstatt sie zum gemeinschaftlichen Lösen zu verwenden.

2.3.2 Fall 2: Big Bank

Das zweite von Sarah Kuhn vorgestellte System wurde in einer Bank angewendet. Dies

war die computergestützte Ausführung von Bankgeschäften (also Überweisungen, Umbuchungen, Kontoeinrichtungen etc.) und musste den Bankstandards an Genauigkeit, Sicherheit und Kundenservice genügen. Wie dieses Beispiel noch zeigen wird, können allerdings nicht alle Standards gleichzeitig ein maximales Level erreichen. Das System realisierte Funktionen und Regeln, die normalerweise unter der Obhut erfahrener AbteilungsleiterInnen durchgeführt wurden. So z.B. die Einbindung der Sicherheitsschranke, dass "under a specified set of exceptional situations (defined by management at the bank through the setting of software parameter settings), the teller's terminal will freeze with a message about the account on the screen.", das Terminal des/der entsprechenden BankkassiererIn blockiert wurde und erst wieder benutzt werden konnte, wenn ein/e MitarbeiterIn mit Befugnis (meistens also ein/e AbteilungsleiterIn) diesen entsperrte. Auch wenn dieses System den hohen Sicherheitsanforderungen einer Bank gerecht werden sollte, stellte es sich doch als unpraktikabel für den Bankalltag heraus, da oben geschilderte Situation in etwa alle 5 bis 10 Minuten auftrat. Folge dessen war vor allem, dass Entsperrkarten an MitarbeiterInnen die diese eigentlich nicht besaßen weitergegeben wurden um so den Betrieb aufrecht zu erhalten. Dies untergrub sämtliche Befugnisvorkehrungen, da die Entsperrkarten nun frei verfügbar waren. Trotzdem kam es durchaus vor, dass ein/e MitarbeiterIn eine/n AbteilungsleiterIn rief, da es ihrer Meinung nach ein kritisches Problem gab (z.B. unbekannte Kunden o.ä.). Sarah Kuhn zitiert Salzman und Rosenthal, die bezüglich dieses Systems zu folgender Festellung kommen: Obwohl das realisierte System der DesignerInnen den BankmitarbeiterInnen durch die Sicherheitsvorkehrungen des Management das Vertrauen für Routineentscheidungen absprach, wurde diesen durch die Weitergabe der Karten viel Verantwortung zugetragen.

2.3.3 Fall 3: Das HELP-System

Das dritte Beispiel welches als gescheitert vorgestellt wird, ist ein System, das für einen größeren US-amerikanischen Flugzeughersteller implementiert und HELP (Help Employees Locate People) genannt wurde. Es kam in einer großen Fertigungshalle zum Einsatz und hatte zum Einen die Funktion, MechanikerInnen nach Hilfe ausrufen zu lassen, wenn sie Unterstützung benötigten (daher der Name). Ob sie nun ein Ersatzteil, eine Beratung mit KollegInnen oder eine Pause brauchten, durch einen Knopfdruck konnte der/die MechanikerIn seine Position und sein Bedürfnis angeben. Dadurch, dass dieser Teil des Systems den Arbeitsbetrieb reibungslos in Gang hielt, wurde er sowohl von den MechanikerInnen als auch vom Management als positiv eingestuft. Was der Name des

Systems nun allerdings nicht verriet, war die Tatsache, dass alle Bewegungen im System aufgezeichnet wurden. So konnte nicht nur der Maschinenstand in Statistiken aufgenommen werden, sondern auch die Arbeitszeit der MechanikerInnen genau beobachtet und überwacht werden – eben dadurch, dass indiziert werden musste, wozu Hilfe benötigt wurde. Es gab tägliche Berichte an die AbteilungsleiterInnen und das obere Management wurde wöchentlich bzw. monatlich informiert.
Die Ziele dieses Systems sind offensichtlich: Mehr Kontrolle über die Zeiteinteilungen der MechanikerInnen um damit die Produktivität zu erhöhen. Allerdings, und das ist neben datenschutzrechtlichen Gründen der wichtigste Punkt, verleiteten die Statistiken zu Fehlschlüssen und Einseitigkeit. Ein Beispiel hierzu ist die Förderrate der Produktionshalle. Um die Produktion nicht unnötig aufzuhalten, wurde von den SystemdesignerInnen implementiert, dass jeder Abfall der Förderrate auf weniger als 80% des Normalzustandes gemeldet wurde. Die Schwierigkeit hierbei liegt in der Definition des Normalzustandes. Hatten die DesignerInnen wirklich einen Begriff von der Förderrate, deren Normalzustand, und konnten sie einschätzen was ein Abfall von 20% bedeutete? Dies konnte das System eben nicht melden, aber es ist nicht ungewöhnlich, dass Förderraten falsch gesetzt werden. Es obliegt den Fähigkeiten des/der MechanikerIn, diese nach den aktuellen Bedingungen zu verlangsamen oder zu erhöhen. So ist das Wissen über die Gründe des aktuellen Förderratenzustands ausschlaggebend zur Interpretation der Statistiken des Überwachungssystems.

3. Arbeitsbezogene Ansätze für Software-Design

Für gute Software, die der Arbeitsumgebung genügen, Eigenschaften und Funktionen besitzen und diese umsetzen noch bevor der/die KundIn weiss dass er sie erfordert, benennt Sarah Kuhn zwei grundlegende Merkmale. Zunächst ist es die Tatsache, dass der Arbeitsplatz eine weitaus komplexere Situation ist und so eine größere Anforderung an Software schafft, als dies zu Hause der Fall wäre. Und zum zweiten gibt es dort den Unterschied zwischen KundIn und BenutzerIn. Unter der Annahmen, dass KundIn und BenuterIn nicht gleich sind, besteht die Schwierigkeit also darin, die richtige Person zufrieden zu stellen - gibt es doch berechtigte Gründe, dass das Management die Software gut heißt. Welcher Seite soll nun der Vorrang gegeben werden? Vor allem beim testen des Systems spielt diese Frage eine zentrale Rolle. Vorgestellt werden sollen nun zwei Konzepte, die sich auf den Endnutzer und den Arbeitskontext konzentrieren.

3.1 Human-centered design

Wie schon im Vortrag *Action-Centered Design* behandelt, kennzeichnet das H*uman-Centered-Design* eine starke Einbindung der NutzerInnen in den Designprozess. Im Gegensatz zum T*echnology-Centered-Approach,* der klare Linien zieht und sich fast gar nicht mit dem organisatorischen Umfeld des Systems, geschweige denn seinen sozialen Implikaturen beschäftigt, stellt das *human-centered-Design s*ystemergonomische Zusammenhänge klar in den Vordergrund. Hierbei geht es nicht nur um eine Einbeziehung der BenutzerInnen in den Designprozess, sondern vor allem auch um die Möglichkeit, als NutzerIn die Software im (Arbeits-)alltag flexibel gebrauchen zu können und sie sich nach eigenen Bedürfnissen einzurichten. Daraus ergibt sich der Anspruch an die Systemdesigner eine Software zu schaffen, die sich an Gegebenheiten und Erfahrungslevels der NutzerInnen anpassen lässt – anstatt anders herum.

3.2 Participatory Design

Wie auch das H*uman-Centered-Design* ist das Participatory Design ausgerichtet auf die Partizipation der EndnutzerInnen. Diese sollen möglichst früh in den Designprozess eingebunden werden. Die Sichtweise der Einbeziehung der Endnuzter hat politische Wurzeln; Angestellte und ArbeiterInnen sollten Einfluss erhalten auf die Gestaltung von Software für den Arbeitsplatz. Im Norwegen der 1970er Jahre begann eine Kooperation zwischen Computerexperten und der Gewerkschaft der Stahl- und MetallarbeiterInnen. Ziel dieser Vereinigung – Kristen Nygaard engagierte sich federführend zusammen mit Gewerkschaftsfunktionären – war es, ein nationales Mitbestimmungsabkommen (*codetermination agreement*) zu schaffen, um den Gewerkschaften so ein Mitspracherecht für die Installation neuer Technologien am Arbeitsplatz einzuräumen.
Zwei Projekte sollen für diese Ausarbeitung exemplarisch vorgestellt werden, die genau die oben benannten Prinzipien verwirklichen wollten und nach effektiven Wegen zur Zusammenarbeit von SystemdesignerInnen und Gewerkschaften respektive Arbeiterorganisationen strebten.

3.2.1 DEMOS

Das Projekt DEMOS brachte in der zweiten Hälfte der 1970er Jahre ForscherInnen aus der

Informatik, Soziologie, Ökonomie und den Ingenieurwissenschaften in einem interdisziplinären Team zusammen. Arbeitsfelder des von der schwedischen *Trade Workers Union* geförderten Projektes waren ein Lokwerkstatt, eine Tageszeitung, eine Metallarbeitsfabrik und ein Kaufhaus. Für die Einbeziehung der DEMOS TeilnehmerInnen gab es verschiedene Gründe. In der Lokwerkstatt waren Gewerkschaftsmitglieder

3.2.2 UTOPIA

Das UTOPIA Projekt setzte sich zum Ziel, ein arbeitsorientiertes Design für computerbasierte Werkzeuge qualifizierter Angestellter zu entwickeln und zu implementieren. Das Projektteam aus dem Zusammenschluss von schwedischen und dänischen ForscherInnen und der Nordic Graphic Workers´ Union sollte eine Prozess- und Methodenkontrolle von geschulten Angestellten für das Seitenlayout und die Bildbearbeitung herstellen. Im Laufe des Projektes sahen sie den Nutzen und die gesteigerte Nachfrage für ein plattformunabhängiges Layoutprogramm, was es in dieser Form damals noch nicht gab. In der Entwicklungsphase des Projektes wurden diverse Benutzerzentrierte Ansätze verwirklicht: Angestellte und DesignerInnen arbeiteten nicht nur eng zusammen, sie lernten auch gegenseitig voneinander und es wurden schon früh Prototypen entwickelt, die den NutzerInnen eine Vorstellung vom System gaben.

3.2.3 Heute

Sarah Kuhn erläutert vier Prinzipien nach Greenbaum und Kyng (*Design at Work,* 1991, S.

4):

1. Die Notwendigkeit, dass DesignerInnen Arbeitabläufe ernst nehmen und als

komplexe Situation wahrnemen.

2. Der Designer arbeitet mit Menschen, nicht mit abstraktem normgerechten

Humankapital.

3. Arbeitsabläufe können nur in ihrem Kontext verstanden und bewertet werden.

4. Arbeit ist im Wesentlichen kooperative, kommunikative und soziale Tätigkeit.

Diese ganz generellen Aussagen gelten auch heute noch. Was hat sich aber verändert? Im

Folgenden soll ein kleiner Überblick geschaffen werden, der einzelne Konzepte vorstellt, aber keineswegs erschöpfend ist.

Wie diverse AutorInnen (unter anderem Grudin und Pruitt (2002)) hervorheben, ist es heutzutage bei Software“massenware“ schwer, die BenutzerInnen klar zu kennzeichnen. Im

Gegensatz zu Projekten aus den 1970er Jahren, die Software für eine einzige Firma evaluierten, müssen heute aus Tausenden oder gar Millionen von BenutzerInnen repräsentative TeilnehmerInnen identifiziert bzw. zusammengestellt werden. Grudin und Pruitt (2002) reflektieren unter anderem das UTOPIA Projekt und fassen zusammen: für neue NutzerInnen kann ein neues Produkt durchaus sinnvoll sein, jedoch können Angestellte, die sich bereits mit alten und vorgegebenen Systemen auseinandersetzen mussten und sich nun darin zu Recht finden Schwierigkeiten mit einer Umstellung haben. Um ein neues System so NutzerInnenfreundlich wie möglich zu gestalten muss ein/e DesignerIn umfangreiche Kenntnisse über das Arbeitsumfeld des/der NutzerIn erlangen. Für Beyer und Holzblatt (1999) ist dazu das Modell *Meister und Lehrling* ein passendes Konzept. Wie ein Lehrling solle der/die DesignerIn von den Angestellten Kenntnisse über ihre Arbeit erlangen. Dabei kommen Gespräche und Begehungen der Arbeitsstätte zum Einsatz. Angelehnt an die erste der Prinzipien von Greenbaum und Kyng formulieren Beyer und Holzblatt ihr Konzept so: „It recognizes that the customers are the experts in their work and the designers are not".

4. Rückblick auf das Buch Bringing Design to Software

Die wissenschaftliche Anthologie von Terry Winograd wurde vor allem für Software Designer geschrieben. Sie zeigt verschiedene Blickwinkel und soll somit dazu führen, dass das Design von Software verbessert werden kann. Es geht vor allem auch darum ob und wie man Methoden aus anderen Bereichen des Designs auf Software Design anwenden kann. Daher wird nun eine Zusammenfassung der vorhergehenden Vorträge dargelegt, um den Text von Sarah Kuhn in den Kontext der Anthologie zu stellen.

4.1 Was war das Ziel des Buches?/ Drei Fragen

Das Buch hatte zum Ziel die folgenden drei wichtigen Fragen zu beantworten.

1. *Was ist Qualität in Software Design? Und wie kann sie vermittelt werden?*
   Hier ging es darum herauszufinden welche konventionellen Methoden erfolgsversprechend sind und ob sie für Software Design geeignet sind. Ferner sollte geklärt werden welche Art von Ausbildung angebracht ist.
2. *Wo steht Software Design im Hinblick auf andere Disziplinen?*

Diese Fragestellung beschäftigte sich damit was man aus anderen Bereichen (z.B. Architektur) lernen kann und wie Software Design mit den traditionellen Aspekten von Software zusammenpasst.

3. *Was ist Design?*
   Diese Frage warf einen Blick darauf ob es eine einheitliche Theorie gibt und ob Design zu Wissenschaft und Ingenieurwesen passt. Hinzu kommt die Frage wie menschliche Belange sinnvoll eingegliedert werden können.

Das Buch hat ein großes Spektrum von Antworten und Möglichkeiten aufgezeigt. Auf diese soll im folgenden Abschnitt eingegangen werden, indem die vorhergehenden Vorträge kurz zusammengefasst werden.

4.2 Zusammenfassung der einzelnen Kapitel:

Kapitel 1: A Software Design Manifesto (Profile: Software Design and Architecture)
Der erste Vortrag des Seminars behandelte das Manifest von Mitchell Kapor. 1990 wandte er sich an die Öffentlichkeit mit dem Aufruf, die neue Berufsbezeichnung des Software-Designers einzuführen.
Um zu verstehen warum es für ihn wichtig war, diese neue Berufsbezeichnung einzuführen, ist es grundlegend, zu begreifen was Software Design ist. The Association for Software Design (ASD) beantwortete diese Frage: “Software design sits at the crossroads of all computer disciplines: hardware and software engineering, programming, human factors research, ergonomics.” (vgl. Kap. 1).
Desweitern stellte sich die Frage: Was ist Design? Dabei zeigte sich, dass Design vor allem der Benutzerfreundlichkeit dient und somit Kommunikation erzwingt. Um die Unterschiede zwischen Software Design und der herkömmlichen Programmierung zu verstehen wurde die Architektur-Software-Analogie vorgestellt. Dabei ist der Software Designer mit einem Architekten zu vergleichen, der den Auftrag bekommt nach den Vorstellungen des Kunden ein Haus zu bauen. Einem Programmierer käme bei dieser Betrachtung etwa die Rolle des Bauingenieurs zu, der das Haus baut.
Inzwischen wurde dem Aufruf von Mitchell Kapor nachgekommen und es gibt die offizielle Berufsbezeichnung des Software Design Engineer. An der RWTH Aachen wird der Studiengang Software Systems Engineering angeboten. Am Georgia Institute of Technologie kann man Human-Computer-Interaction studieren.

Das **Kapitel 2**: *Design of the conceptual Model (Profile: the Alto and the Star)* hat sich mit dem Begriff des Conceptual Model befasst. Das Conceptual Model ist ein Konzept zur

Gestaltung von benutzerfreundlichen Anwendungen. Es soll dem Benutzer ermöglichen die Software intuitiv zu benutzen und schnell zu erlernen.

Beim Conceptual Model wird davon ausgegangen, dass der Benutzer ein mentales Modell von der Software bzw. dem System hat. Ein häufiger Fehler, der bei der Entwicklung neuer Software gemacht wird ist, dass dieses mentale Model nicht ausreichend beachtet wird. Deshalb ist eine gründliche Aufgabenanalyse sehr wichtig.
Von PARC (Palo Alto Research Center) wurde der erste PC mit einer graphischen Benutzeroberfläche entwickelt. Das war der Xerox Alto. Bei seinem Nachfolger, dem Xerox Star wurde das Conceptual Model durch verschiedene Elemente realisiert. Das sind Symbole bzw. Metaphern, Property Sheets, direct Manipulation, WYSIWYG und die Übereinstimmung von Befehlen in verschiedenen Anwendungen.
Am Beispiel der Klimaanlage im Auto haben wir gesehen wie ein gelungenes Conceptual Model aussieht, wohingegen dieses Konzept beim Telefon schlecht umgesetzt wurde.
Obwohl der Xerox Star nicht erfolgreich war, schaffte er eine gute Grundlage für folgende Systeme, wie wir sie auch heute noch kennen. Das Conceptual Model spielt auch heute noch eine wichtige Rolle im Bereich des Software-Designs.

Der folgende Vortrag beschäftigte sich mit **Kapitel 4**: *Design Languages (Profile: Macintosh Human Interface Guidelines)*. Designsprachen dienen dem Designer dazu ihren Produkten eine Bedeutung zu geben, die für den Benutzer schon anhand des äußeren Erscheinungsbildes erkennbar sind.
Designsprachen bestehen aus eine Sammlung von Grundelementen, Kompositionsregeln und einer Sammlung von Anwendungsbereichen. Sie bieten dem Benutzer durch Interpretation die Möglichkeit des „lerning by doing“. Außerdem können sich Designer durch Assimilation bedeutsame Erfindungen zu nutze machen. So sind zum Beispiel Fenster, Icons, Menüs und Zeiger Standard für Computerinterfaces.
Als Beispiel wurden die Macintosh Human Interface Guidelines betrachtet. Diese basieren auf dem WIMP (Windows, Icons, Menus, Pointer) Interface. Sie geben genaue Vorschriften für alle Interfaceelemente.
Die einzelnen Prinzipien des Mac und des Anti- Mac, als Alternative zum Mac wurden etwas genauer betrachtet. Die Idee des Anti- Mac stammt von Don Gentner und Jakob Nielsen und hat den Ansatz, das klassische WIPM Interface weiter oder gar neu zu entwickeln.

In **Kapitel 5:** *The Consumer Spectrum (Profile: Mosaic and the World Wide Web)* wurde speziell auf den Endverbraucher eingegangen.

Um gute Software zu designen ist es wichtig, Annahmen darüber zu treffen, auf was der Endverbraucher Wert legt, also was genau er haben will. Dabei teilt man den Endverbraucher in erfahrene und unerfahrene NutzerInnen ein. Der unerfahrene Benutzer möchte ein Produkt das einfach zu bedienen ist, wohingegen der erfahrene Nutzer mehr Wert auf Effektivität und Effizienz legt. Um besser zu Verstehen was der Endverbraucher will, bedient man sich des Konzeptes der *Schwelle des Verweigerns*.
Im Allgemeinen sieht die Elektronikindustrie Benutzerfreundlichkeit als sehr wichtiges Kriterium für den Erfolg eines Produktes an. Allesdings hat sich herausgestellt, dass auch die Kosten ein wichtiges Kriterium sind. Wenn sich Kosten bzw. Aufwand und Nutzen eines Produkts im Gleichgewicht befinden spricht man von der *Schwelle des Verweigerns*. Ein Produkt ist dann ideal, wenn es sich an die Fähigkeiten des Benutzers anpasst, und dem Nutzer immer mehr Funktionalitäten anbieten kann.
Am Beispiel des tragbaren Media-Players iPod wurde gezeigt, dass eine ausführliche Betrachtung des Endverbrauchers ein Produkt erfolgreich machen kann. Paul Saffo sagte dazu, dass der iPod sich deshalb auf dem Markt durchgesetzt hat, weil er kein Produkt sondern ein System ist.

In **Kapitel 6**: *Action-centered Design (Profile: Business Process Mapping)* wurde das Konzept des *Action-centered Design* vorgestellt. Bei diesem Konzept steht der Benutzer, dessen Fähigkeiten und seine Ansprüche an eine Software im Mittelpunkt. Aus diesem Grund ist eine ständige Kommunikation mit den späteren Nutzer, der oftmals jedoch nicht Auftraggeber ist, sehr wichtig.
Software nach alten Konzepten war nicht auf den Nutzer und dessen Arbeitsumfeld zugeschnitten. Das Action-centered Design beschäftigt sich nun genau damit, das Arbeitsumfeld zu erkennen, zu abstrahieren und in ein unterstützendes Softwaresystem einzubinden.
Ende 1980 entstand der neue Ansatz des Human-centered Design, der das veraltete Software-Engineering ablöste. Das Action-centered Design stellt eine Unterkategorie des Human-centered Design dar. Eine der wichtigsten Annahmen ist, dass das Ergebnis der Entwicklung ein zufriedener Kunde ist.
Der DIA- Cycle (Design- Implement- Analyze) beschriebt dabei das grundsätzliche Vorgehen. Das Produkt wird designt und implementiert und dann dem Kunden zum Testen gegeben. Diese Schritte werden so lange wiederholt bis der Kunde zufrieden ist. So wird gewährleistet, dass sich das Produkt den Ansprüchen des Kunden anpasst. Die Kundenzufriedenheit ist das wichtigste Ziel, und es darf nicht vergessen werden, dass diese Zufriedenheit nicht von Dauer ist.
Das Beispiel der Finanzsoftware Quicken hat gezeigt, dass sich dieses Konzept bewährt hat.

Kapitel 8: *The Designer´s Stance (Profile: IDEO)* beschäftigte sich näher mit dem Designer selbst. Zunächst wurde einmal der Blick auf die Unterschiede zwischen Engineering und Design gelenkt. Beim Engineering löst man Probleme und wendet dabei mathematisch, logische Methoden an. Beim Design hingegen ist viel mehr Kreativität erforderlich. Es wurde der Begriff *Creative Leap* vorgestellt. Damit ist gemeint, dass Designer Probleme aus verschiedenen Blickwinkeln betrachten müssen um innovative Idee entwickeln zu können.
Ganz wichtig für ein erfolgreiches Design ist Teamarbeit. Denn nur die vielen unterschiedlichen Meinungen machen es möglich ein neuartiges Produkt zu entwickeln.
Später wurde das erfolgreiche Designunternehmen IDEO vorgestellt. Die Methodologie bei IDEO besteht aus den fünf Schritten Verstehen, Beobachten, Veranschaulichen, Auswerten/ Verfeinern und Implementieren. Zu Anfang geht es darum, dass der Designer sich mit dem Produkt auseinandersetzt und entsprechende Informationen einholt. Danach ist es wichtig sich auf den Benutzer und dessen Aufgaben zu konzentrieren. Anschließend wird die Situation veranschaulicht in welcher der Kunde das Produkt benutzt. Danach folgen Test und eventuelle Verbesserungen. Natürlich gibt es viele verschiedene Versionen dieser Methode, die mit einander vermischt werden können.

In **Kapitel 10**: Cultures of Prototyping (Profile: Hypercard, Director and Visual Basic) wurde gezeigt, warum Prototypen für die Entwicklung von Software so wichtig sind.
Es wurde darauf eingegangen, dass Kommunikationsprobleme große Folgen haben können. Zum einen kann der Kunde seine Vorstellungen nicht richtig artikulieren, was dazu führt dass der Entwickler keine Fachkenntnisse erlangt und das führt wiederum dazu, dass ein Produkt entsteht, dass nicht den Vorstellungen und Wünschen des Kunden entspricht.
Um diese Probleme zu vermeiden benutzt man die Methode des Prototyping. Prototyping bezeichnet dabei die Herstellung eines Prototypen. Ein Prototyp ist die praktische Umsetzung der Spezifikationen des Kunden. Es gibt zwei Vorgehensweisen beim Prototyping. Bei der herkömmlichen Vorgehensweise werden nur wenige Prototypen hergestellt. Diese zeichnen sich dann durch eine hohe Qualität aus. Allerdings ist diese Entwicklung sehr langsam und kostenintensiv. Beim schnellen Prototyping werden sehr viele Modelle hergestellt, was eine Kostensenkung mit sich bringt und dadurch einen Wettbewerbsvorteil bietet.
Qualitative Aspekte für eine gute Software sind die Benutzerfreundlichkeit, welche durch die Kommunikation mit dem Kunden und zahlreiche Test gewährleistet werden soll, die Wartungsfreundlichkeit, die eine leichte Erweiterung des Systems ermöglicht und die Zuverlässigkeit.

Beim folgenden Planspiel konnte man selber erfahren wie wichtig Prototypen sind. Im Allgemeinen stellt ein Expertenbenutzer das Model dem Kunden vor und simuliert seine Funktion.

Kapitel 12: *Design as Practiced (Profile: The Design of Everyday Things)*

Im folgenden Kapitel 12: Design as Practiced (Profile: The Design of Everyday Things) wurde das Buch The Design of Everyday Things von Donald A. Norman vorgestellt. In diesem Buch geht es darum heraus zu finden warum manche Dinge so schwierig sind und wie man sie verbessern kann.

Am Beispiel „Licht Anmachen“ wurden die Seven Stages of Action erläutert. Diese sind als Zyklus aufgebaut. Am Anfang steht ein Ziel. Danach folgen die Schritte Absicht, Bewegungssequenz und Durchführung. Diese haben Folgen auf die Umwelt, welche in den nächsten 3 Schritten analysiert werden (Bemerken, Interpretieren, Vergleichen). Wenn das Ergebnis nicht zufriedenstellen ist wird von vorne angefangen.

Es gibt verschiedene Eigenschaften, die die Qualität eines Objekts beeinflussen. Zum einen ist es notwendig dem Benutzer eine gewisse Affordance zu bieten, also genügend Anhaltspunkte zu geben damit er die Funktion des Objekts versteht. Zusätzlich unterliegt ein Objekt verschiedenen Zwängen und Konventionen, welche am Beispiel des Lego-Motoradfahrers erklärt wurden. Desweiteren ist ein gutes bzw. natural Mapping erforderlich (Zusammenhang zwischen der Steuerung und den Auswirkungen). Auch das Conceptual Model spielt eine große Rolle. Außerdem ist ein ausreichendes Feedback wichtig, damit der Benutzer die Resultate seiner Handlungen erkennt.

Zusammenfassend lässt sich sagen, dass ein Objekt „gut“ ist wenn es dem Benutzer nur eine „richtige“ Lösung zum Handeln bereitstellt.

In **Kapitel 13**: *Organizational Support of Software Design (Profile: Quicken)* wurden die organisatorischen Aspekte von Software Design betrachtet. Es wurden drei wichtige Ziele verdeutlicht: deutliche Ziele bestimmen und beibehalten, Kundenorientierung und die Gestaltungsfreiheit des Designers. Am Beispiel *Manusoft* wurde gezeigt, dass es Probleme geben kann, wenn Ziele mehrdeutig sind und keine Prototypen-Kultur existiert. Bevor ein Design geändert wird, muss geprüft werden ob das Design noch zu den Zielen passt, oder ob vielleicht die Ziele angepasst werden müssen. Das Beispiel *Datasoft* hat gezeigt, dass nicht genug Kundenorientierung stattgefunden hat. Ein neues Produkt sollte immer einen Gewinn für den Kunden darstellen. Dazu gilt es den Kunden genau zu studieren und mit ihm

in Interaktion zu treten. *Intuit* erleichtert die Kommunikation mit dem Kunden für den Entwickler und verwendet die Methode des *follow me home*, um potentielle Probleme vor Ort entdecken zu können. Ganz besonders wichtig für erfolgreiche innovative Produkte ist die Kreativität der Designer. Der Designer braucht Sicherheit und Unterstützung vom Unternehmen und seine Arbeit muss anerkannt werden. Am Schluß wurde die Finanzsoftware Quicken vorgestellt, welche von Intuit entwickelt wurde. Hervorzuheben ist hier die ausführliche Kommunikation mit dem Kunden auch während der Entwicklung. Abschließend lässt sich sagen, dass Software Design ein langwidriger komplizierter Prozess ist und daher Organisation benötigt.

In diesem **Kapitel 14**: *Design for people at work (Profile: Participatory Design)* wurde dargestellt welche Effekte Computer und computergesteuerte Maschinen auf den Arbeitsplatz haben. Es ist wichtig ein geistiges Konzept von der Arbeit zu haben, das soweit wiemöglich mit der Realität übereinstimmt. Sonst kann es zu hohe Kosten, sowohl für die Manager durch mangelnde Produktivität, als auch für die Arbeiter durch schlechte Arbeitsbedingungen kommen. Um diese Probleme zu beheben wurden arbeitsorientierte Ansätze für Software Design vorgestellt. Das Participatory Design konzentriert sich speziell darauf, dass die Angestellten ein größeres Mitspracherecht erhalten.

4.3 Versuch einer Antwort

Nun stellt sich natürlich die Frage ob das Buch sein Ziel erreicht hat. Haben wir Antworten auf die Fragen erhalten? Außerdem kann man überlegen, ob und wie diese Antworten das weitere Vorgehen beeinflussen.

Wir denken, dass das Buch durchaus Antworten geben konnte, oder doch zumindest zahlreiche Anregungen. Man kann diese in seine Gedanken zu eigenen Projekten mit einfließen lassen. Software Design ist ein sehr weiter Bereich, so umfasst er zum Beispiel die Software in einem Office PC, einem MP3-Player oder einer Digitalkamera, aber auch Fahrkartenautomaten, Buchungs- und Reservierungssoftware für Hotels und computergesteuerte Maschinen zum CNC Fräsen. Dies zeigt die vielen Anwendungsmöglichkeiten von Software. Jede Situation und vor allem jeder Benutzer ist anders und das Konzept muss sich diesen Gegebenheiten anpassen.

Das wichtigste was man aus diesem Buch mitnehmen sollte, ist dass der spätere Endverbraucher in das Designkonzept mit einbezogen werden muss. Dabei sollte darauf geachtet werden, dass man *mit* dem Benutzer designt und nicht nur für ihn. Nur dann ist es möglich gute Software zu entwickeln. Aber was zeichnet gute Software aus? Um diese Frage zu beantworten beschäftigt sich der nächsten Abschnitt noch einmal mit der Qualität von Software Design.

4.3.1 **Qualität von Software Design**

Qualität ist ein sehr schwer fassbarer Begriff. Sie ist objektiv und subjektiv zu gleich. Besonders die Qualität von Softwaredesign ist schwer zu erfassen, da Softwaredesign weder eine Wissenschaft noch Kunst ist. Und trotzdem ist es notwendig diese zu beurteilen. In anderen Bereichen des Designs wird Qualität immer von Gleichgesinnten bewertet; Leuten, die entweder selber entwickeln oder sich sehr dafür interessieren. So geschieht es auch beim Software Design. 1996 wurde vom ACM (Association for Computing Machinery) zum ersten Mal der *interactions design award* für „gute“ Software vergeben. Natürlich ist es sehr schwer fest zulegen was die Qualität einer Software ausmacht. Vielleicht kann man gute Software so erkennen: „[…] But when it comes to awards, we care about quality of interaction. We want to recognize products, services and environments that enhance people`s lives. We´re looking for designs that effectively help people work, learn, live, play or communicate.” (vgl. Winograd 1996). Außerdem spielen *Public Assessments* eine große Rolle. Durch Online- Kommunikation ist es immer mehr Menschen möglich sich an den Diskussionen über „gute“ Software zu beteiligen. So gibt es zahlreiche online Plattformen, auf denen sich die Software Designer austauschen können. Zusätzlich findet jährlich, wie auch dieses Jahr vom 1.-5. August, die Participatory Design Conference statt. Dieses Jahr hat sie das Thema *Expanding Boundaries in Design*, welches: “[…] focuses attention on the multiple contexts in which design takes place and on an expanding range of possible design outcomes.” (vgl. CPSR)
Nun reicht ein Qualitätsbegriff allein nicht aus. Es muss auch Leute geben, die diese Anforderungen umsetzen können.

4.3.2 Auf dem Weg zu einem neuen Beruf

Mitchell Kapor sagte in seinem Manifesto (Kapitel 1): „we need to create a professional discipline of software design“. Im Laufe des Buches wurden verschiedene Aspekte vorgestellt, dies zu realisieren. Eine Software Industrie ohne diese spezielle Disziplin kann Probleme bekommen. Erfolge sind nicht so einfach zu wiederholen, weil man nicht genau

weiß warum ein entsprechendes System gut oder schlecht ist. In jedem Beruf gibt es erprobte Methoden und Techniken für bestimmte Prozesse. Erst sie ermöglichen ein wissenschaftliches Vorgehen und ein Weitergeben an zukünftige Software DesignerInnen. Design kann aber nicht nur auf standardisierte Prozeduren und formale Methoden reduziert werden; Kreativität und Erfahrung spielen eine ganz entscheidende Rolle.

Die Komplexität von Software Design zeigt sich dadurch, dass Software Design viele verschiedene Disziplinen umfasst: Software Engineering, Softwarearchitektur, Programmieren, menschliche Faktoren, Graphikdesign, Soundproduktion, Ästhetik, Psychologie und andere. Also was für eine Expertise brauchen Software DesignerInnen? Sie müssen gestalten und entwickeln können, sie müssen die Sprache aller beteiligten Personen sprechen, sie müssen ihre Ideen kommunizieren können und sie müssen jede Disziplin gut genug verstehen, um vernünftige Entscheidungen treffen zu können.

4.3.3 **Ausbildung zum/zur SoftwaredesignerIn**

Durch die Entstehung eines eigenen Berufs, ist es aber auch nötig sich Gedanken über eine spezielle Ausbildung zum/zur Software DesignerIn zu machen. Auch heute ist es noch so, dass die meisten etwas anderes gelernt haben, aber dann auf irgendeinem Weg doch beim Software Design gelandet sind. Allerdings entstehen immer mehr Programme um Software Design zu unterrichten. Es stellt sich die Frage welches die beste Ausbildungsumgebung ist; Universität, spezielle Kurse oder im Job. Desweitern wird zu klären sein, welcher Fachbereich diese Aufgabe übernehmen sollte, oder ob man die Ausbildung interdisziplinär anlegen sollte. Soll es Kurse oder Praktika geben, soll man die Studierenden von Anfang an ausbilden, oder erst, wenn sie schon ein anderes Studium absolviert haben? Hier ergeben sich sehr viele Fragen, die noch zu beantworten sein werden.

5. Und Heute?

Nun ist es schon 10 Jahre her seitdem das Buch geschrieben wurde. Deshalb soll sich das letze Kapitel mit der Frage beschäftigen, was sich bis heute getan hat. Wurden die Ansätze umgesetzt? Gibt es inzwischen neue, bessere Methoden? Wie sieht es mit dem Berufstand aus?

5.1 Neue Ansätze

Für viele Software DesginerInnen besteht der Kreis der NutzerInnen heutzutage aus einer unüberschaubaren Menge von Tausenden, oft Millionen Menschen. Wie kann hier das Prinzip der NutzerInnenpartizipation bei der Entwicklung im ursprünglichen Sinne

hinreichend beachtet und umgesetzt werden? Pruitt und Grudin (2002) beschreiben zur Lösung dieses Problem Alan Coopers Konzept der Personas; fiktive aber in vielen Details ausgearbeitete Personen die eine unspezifische Mengen von NutzerInnen zusammenfassen, anhand deren Szenarien durchgespielt werden können.

Aber auch politisch wird weitergedacht. Die CPSR – Computer Professionals for Social Responsibility arbeitet daran, ein Bewusstsein dafür zu etablieren, welche weitreichenden Konsequenzen die Entwicklung und Implementierung neuer Softwaresysteme haben und dass der/die DesignerIn Verantwortung hat für die Menschen, die mit seiner/ihrer Software umgehen.

Als ein weiterer Punkt sollen die sogenannten *Participatory Design Workshops* angeführt werden, die als Weiterentwicklung der EndnutzerInnenbeteiligung ganz konkret alle Beteiligten – sprich EntwicklerInnen/DesignerInnen, NutzerInnen und auch KundInnen – zusammen an Lösungen arbeiten.

5.2 Ausbildung

Der Berufsstand des/der Software DesignerIn hat sich inzwischen etabliert. Die Nachfrage nach Software DesignerInnen ist groß, so gibt es zahlreiche Stellenangebote. Zum Aufgabenbereich einer/s Software DesignerIn gehören viele Tätigkeiten, etwa der Entwurf, die Realisierung und anschließende Tests, aber auch enger Kundenkontakt. Wenn man sich die Voraussetzungen an eine/n BewerberIn anschaut stellt sich heraus, dass fast immer ein Studium der Informatik und/ oder mehrjährige Berufserfahrung in der Softwarenentwicklung gefragt sind. Außerdem ist praktische Projekterfahrung, Teamgeist, Sozialkompetenz und Kommunikationsfähigkeit erwünscht. Hier stellt sich die entscheidende Frage: Wie kommt man zu praktischen Erfahrungen und den anderen Kenntnissen, die verlangt werden? Das Informatikstudium befasst sich meist nur mit den theoretischen Komponenten, wie Algorithmen und Datenstrukturen. Wie kommt man also zu den weiteren notwenigen Fähigkeiten eines Software Designers? Hier ist Eigeninitiative gefragt, denn die Ausbildungsangebote sind nicht so zahlreich und umfangreich, wie man glauben könnte.

Einen Studiengang mit dem Namen Software Design scheint es nicht zugeben. Aber es gibt zahlreiche verwandte Studienfächer, hauptsächlich Software Engineering, oder Softwaretechnik. Vielfach gibt es entsprechende Vorlesungen oder Nebenfächer, die ins Konzept passen. In Österreich und der Schweiz ist man etwas fortschrittlicher. An der FH Joanneum in Österreich ist es möglich „die berufsbegleitende Vertiefungsrichtung *Software*

Design […]“ (vgl. Internetseiten der FH Johanneum) zu studieren. In der Schweiz gibt es die sogenannte Software Schule, welche zahlreiche Weiterbildungsmöglichkeiten anbietet. Weiterbildung ist das entscheidende Stichwort. Wenn man im Bereich des Software Design bestehen will, muss man sich ständig weiterbilden. Hierzu gibt es verschiedenste Möglichkeiten; zahlreiche Online-Angebot, Fortbildungskurse und Aufbau- oder Ergänzungsstudiengänge.

Berufserfahrung kann man sammeln wenn man eine Berufsausbildung macht. Allerdings ist man dann nicht so hoch qualifiziert wie ein Hochschulabsolvent. Abhilfe schaffen hier Berufsakademien, die duale Studiengänge anbieten. „Die Berufsakademie verbindet ein wissenschaftliches Studium mit der praktischen Anwendung am Arbeitsplatz. Im dualen Prinzip übernehmen zwei Partner die Aufgabe, die Studierenden für den Beruf zu qualifizieren: das Unternehmen als Lernort für die Praxis und die Staatliche Studienakademie als Lernort für die Theorie.“ (vgl. Internetseiten der BA Stuttgart). Hier sind allerdings die Ausbildungsplätze sehr knapp.

Die meisten Studienangebote im Rahmen des Software Design finden sich eher an Fachhochschulen und weniger an Universitäten, was daran liegen mag, dass hier die Lehre praktischer orientiert da Erfahrungen sehr wichtig sind um eine gute Software zu designen.

Wenn man Software DesignerIn werden möchte, sollte man sich die Lehrpläne und Inhalte der Vorlesungen genau angucken und ein entsprechendes Nebenfach wählen. Oder man greift auf ein interdisziplinäres Angebot zurück. Auf jeden Fall wird von den Studierenden viel Eigeninitiative gefordert und der Wille zur Weiter- bzw. Ergänzungsbildung.

6. Literaturangaben

- Beyer H. R., and Holtzblatt K. (1999): *Apprenticing with the Customer. Communications of the ACM. S.* 45-52. http://www.itee.uq.edu.au/~comp4501/_2003/_Readings/Beyer.pdf
- Berufsakademie Stuttgart: http://www.ba-stuttgart.de/474.0.html
- CPSR (Computer Professionals for Social Responsibility): http://www.cpsr.org/issues/pd/pdc2006
- FH Joanneum: http://www.fachhochschulen.at/FH/Studium/Software_Design_2820.htm
- Grudin J., and Pruitt J. (2002): *Personas, Participatory Design and Product Development: An infrastructure for engagement.* Participation and Design Conference, 2002 (PDC2002), Sweden, S. 144-161. (http://www.itee.uq.edu.au/~comp4501/_2003/_Readings/GrudinPersonas.pdf)

- Winograd, T. (1996): *Bringing Design to Software.* Addison-Wesley.