Designansätze

Benutzerzentriertes Design

Johannes Rössel
Universität Rostock
johannes.roessel@uni-rostock.de

Zusammenfassung

Der Entwurf komplexer Systeme, insbesondere Softwaresysteme, hat sich in den vergangenen Jahren um Einiges gewandelt. Der Schwerpunkt auf den Benutzern statt auf dem System an sich ist gerade in letzter Zeit immer wichtiger geworden und hier sind verschiedene Techniken entstanden, um das benutzerzentrierte Design methodisch und „richtig“ zu machen. Einige davon werden hier vorgestellt.

Einleitung

Der Computer war lange Zeit zwar ein sehr nützliches Werkzeug, allerdings war dessen Bedienung lediglich ein paar wenigen Experten vorbehalten. Dies änderte sich in den 80er und 90er Jahren gravierend mit dem Aufkommen von Heim- und Personal Computern. Nur ein Problem war zu diesem Zeitpunkt noch nicht gelöst: Es waren nun normale Menschen, die vor diesen Maschinen saßen und sie konnten, wollten und vor allem sollten nicht die Kenntnisse besitzen, die vormals Leute auszeichnete, die Computer bedienen durften. Nicht immer gelang dies.

Seitdem hat sich dieser Zustand gebessert. Benutzerfreundlichkeit kommt langsam auch in Systeme, die sich seit Dekaden gerühmt haben, ein Werkzeug für Profis zu sein statt für das gemeine Volk. Aber es ist ein schleichender Prozess. Allerdings ist Benutzerfreundlichkeit auch etwas, was sich durch verschiedene Methoden im Entwicklungsprozess der Software bzw. des kompletten Systems fast automatisch ergibt.

Nun ist die Interaktion des Benutzers zwar auf eine Benutzeroberfläche *(User Interface)* beschränkt, jedoch umfasst die Interaktion mit einem Softwaresystem notwendigerweise auch die Erfahrung beim Benutzen, eventuellen Frust der von unnötigen Pausen oder Umständen im Arbeitsfluss herrührt, etc. *(User Experience)*. Für die Schaffung einer guten Interaktion mit dem Nutzer ist es daher nötig, nicht nur darauf zu achten, dass die Benutzeroberfläche hübsch anzusehen ist, sondern auch, dass der Benutzer Freude daran hat, damit zu arbeiten und effizient arbeiten kann. Reine Richtlinien für UI-Design, wie „Diese Schaltfläche gehört hierhin und diese Liste dorthin“ reichen also nicht aus.

Ich werde allerdings auf solche Dinge hier auch nicht eingehen. Der Fokus dieses Tex-

tes soll auf Methoden zum Schaffen guter Benutzbarkeit liegen, nicht auf kleinen Nebenprozessen, die zwar sicher wichtig und unersetzlich sind, aber an anderer Stelle ausreichend und erschöpfend behandelt werden. Hierzu gehört unter anderen *Usability Testing,* da es einerseits Bestandteil eigentlich aller hier vorgestellten Methoden ist, andererseits aber auch eine Ebene zu niedrig angesetzt ist. Ich will mich hier eher auf umfassendere Ziele beschränken als auf die einzelnen Techniken zum Erreichen dieser. Ebenfalls nicht angesprochen werden einzelne Details von GUI-Design. Wie schon erwähnt, ist es zwar unerlässlich für konsistente und benutzbare grafische Oberflächen, aber es reicht nicht aus, um die gesamte Interaktionserfahrung zu kontrollieren.

Contextual Design

Contextual Design ist ein von Karen Holtzblatt und Hugh Beyer in der zweiten Hälfte der 90er Jahre entwickelter aufwendiger Workflow zur völligen Neugestaltung von Aktivitäten, Arbeitsabläufen, etc. Der Kernpunkt hierbei ist die Einbeziehung des Arbeitskontextes in den Designprozess und infolgedessen auch eine Abkehr von dem üblicherweise Feature-basierten Designprozess, der sonst gerade bei Software üblich ist. Das Ziel ist am Ende ein Produkt oder System zu haben, welches den Bedürfnissen der Anwender so gut wie möglich entspricht. Um dies zu erreichen, sind mehrere Schritte nötig:

Datenerhebung. Dies geschieht im sogenannten *Contextual Interview* bzw. *Contextual Inquiry* und findet im Arbeitskontext der Benutzer statt – also während sie arbeiten –, nach Möglichkeit ohne sie zu stören oder zu beeinflussen (UsabilityNet, 2006) und dient dazu, Rohdaten zu sammeln. Es sollte bei diesem Schritt noch keinerlei Analyse der Daten vorgenommen werden (UsabilityNet, 2006).

Üblicherweise beginnt man mit einem normalen Interview, wo der Interviewer einen Überblick über die Arbeit des zu Interviewenden bekommt und eine Vertrauensbasis herstellt. Danach folgt die Umstellung auf eine Art Lehrer-Schüler-Verhältnis, wo der Interviewer der Schüler ist und im Wesentlichen beobachtet, allerdings auch unterbrechen kann um Fragen zu stellen. Am Ende folgt eine Zusammenfassung seitens des Interviewers, allerdings im Beisein des zu Interviewenden. Er sollte hier darstellen, was er gelernt hat und versuchen, die Sichtweise des zu Interviewenden widerzuspiegeln.

Hier schließt sich das *Erstellen der Arbeitsmodelle* an. Arbeitsmodelle dienen dazu, die Arbeit Einzelner oder ganzer Unternehmen anschaulich zu machen und verschiedene Perspektiven darauf zu ermöglichen. Es gibt hier fünf verschiedene Modelle (Holtzblatt, 2001):

Flussmodell: modelliert Kommunikations- und Koordinationswege, inklusive Rollen, Hierarchien und Verantwortlichkeiten.

Kulturmodell: Einflüsse und Hemmnisse auf den Arbeitskontext, einschließlich Normen, Unternehmenspolitik, Arbeitsklima, etc. Ziel hier ist, herauszuarbeiten, was für den Einzelnen wichtig ist und wie das die Arbeit beeinflusst.

Sequenzmodell: modelliert konkrete Arbeitsschritte bzw. Aufgaben, einschließlich ihrer Abfolge. Mit einbezogen werden hier häufig auch besondere Vorkommnisse, Interaktionen im Ablauf und Fehlersituationen.

Physikalisches Modell: Dient zur Darstellung physikalischer Gegebenheiten: Plätze, Räume, Entfernungen, Höhen, Anordnungen, Artefakte (s. u.). Hier wird allerdings ausschließlich das Erscheinungsbild modelliert und wie es sich auf die Strukturierung der Arbeit auswirkt, jedoch keinerlei Funktionalität.

Artefaktmodell: Arbeitsmittel, die verwendet werden, sowie ihre Funktionsweise, beispielsweise Dokumente, Terminkalender, etc. Hieraus lässt sich ableiten, wie Menschen ihre Arbeit strukturieren und organisieren, was in die Erkenntnisse mit einfließt, die man aus dem Interview gewonnen hat.

Diese Arbeitsmodelle werden aus den in den Interviews gewonnenen Daten erstellt und sogenannte Interpretationssitzungen dienen dazu, dass alle Beteiligten ihre Gedanken und Ideen einbringen können.

Hiernach folgt die *Konsolidierung* der Daten, also das Finden von Strukturen und Mustern in den Abläufen. Hierfür kann man sogenannte *Affinitätsdiagramme* verwenden (Beyer & Holtzblatt, 1998): Jede Beobachtung wird auf einen Zettel geschrieben, diese Notizzettel werden nach Ähnlichkeit gruppiert und mit farbigen Klebezetteln markiert und daraus dann nach Wichtigkeit geordnete Hierarchien erstellt.

Es folgt das *Benutzer-Umgebungs-Design (User Environment Design).* Hier wird ein potentiell neuer Arbeitsablauf bzw. ein System geschaffen, aufgrund der erfassten Daten, der möglichst gut auf selbige passt. Hilfsmittel sind hier Storyboards und Skizzen, die illustrieren, wie Benutzer mit dem neuen System arbeiten. Es wird nach Möglichkeit jeder Bestandteil des neuen Systems dargestellt, wie er die Arbeit der Benutzer unterstützt, welche Funktion genau vorhanden ist und wie ein Benutzer von einem Bestandteil des Systems zu einem anderen gelangt.

Als letzter Schritt bleibt dann nur noch das *Testen mit den Anwendern.* Dies geschieht üblicherweise mit Papier-Prototypen, um das neue System testen zu können, bevor man es „richtig" entwickelt und einrichtet.

Zusammenfassend ist Contextual Design gerade aufgrund der Datensammelphase sehr zeit- und, durch den hohen Personalaufwand in dieser Zeit, auch recht kostenintensiv und damit nicht für alle Bedürfnisse oder Projekte anwendbar. Allerdings ermöglicht gerade dieser Teil ein umfassendes Verständnis der Bedürfnisse der betroffenen Benutzer und infolgedessen auch eine Lösung, die diese Bedürfnisse gut erfüllt.

Einen etwas leichtgewichtigeren Ansatz werde ich im Folgenden vorstellen.

Iterative Development in the Field

Ist *Contextual Design* im Ganzen zu aufwendig ist, ist *Iterative Development in the Field* (Greene, Jones, Matchen, & Thomas, 2003) vielleicht eine Methode, die etwas brauchbarer auch für kleinere Projekte ist, obwohl sie immer noch viel Zeit kostet.

Die Kerngedanken bei *IDF* leiten sich schon aus dem Namen ab: Es ist ein iterativer Prozess, der *„in the field",* also direkt bei und mit den späteren Benutzern des Systems durchgeführt wird. Im Zentrum steht hier eine sogenannte *„proto-application",*[1] eine Art

[1] Nach (Greene, Jones, Matchen, & Thomas, 2003) wurde der Begriff „proto-application" gewählt, um zu verdeutlichen, dass es sich eben nicht um

Prototyp, der zum fertigen System weiterentwickelt wird. Man arbeitet direkt mit den Benutzern und anderen Stakeholdern[2] zusammen und es gibt gewisse Ähnlichkeiten zu *Contextual Design,* allerdings ist IDF deutlich mehr auf die Implementierung zentriert und entwickelt sich somit eher evolutionär als analytisch. Weiterhin wird im Gegensatz zu *Contextual Design* in jeder Phase direkt mit den Stakeholdern zusammengearbeitet statt nur ganz am Anfang.

Der Entwurfs- und Entwicklungsprozess gliedert sich in vier Phasen:

Discovery: Hier wird zusammen mit Entwicklern, Usability-Experten und den Stakeholdern versucht, ein erstes Verständnis der Umgebung, Probleme und Möglichkeiten zu erlangen. Hierbei werden erste Systemanforderungen festgestellt.

Es folgt die *Proof-of-concept*-Phase, in der ein kleiner Teil der Anforderungen und Funktionen sowie eine frühe Version der Benutzeroberfläche realisiert werden. Diese *proto-application* wird dann sofort den späteren Benutzern zur Verfügung gestellt und erste Erfahrungen werden aufgenommen.

Diese *proto-application* wird dann in der *Pilot*-Phase weiter entwickelt. Dieses ist die Hauptphase der iterativen Entwicklung und Funktionen werden hinzugefügt, verändert oder anders priorisiert, basierend auf den Erkenntnissen, die man mit den Benutzern „im Feld" gesammelt hat. Hierfür ist immer noch die ständige Zusammenarbeit mit selbigen nötig. In dieser Phase nehmen Umfang, Tiefe, Anzahl der Benutzer und Installationsorte des Systems ständig zu.

Schlussendlich folgt die *Deployment*-Phase, in der dem System der letzte Schliff gegeben wird und es ausgeliefert wird.

Iterative Development in the Field ist, ähnlich wie auch *Contextual Design,* eine Methode für große Projekte für einen sehr eng eingegrenzten Zweck (beispielsweise einen bestimmten Kunden). *IDF* hat seine Ursprünge bei IBM Research und die Forscher dort haben es unter anderem für interaktive Touchscreen-Terminals auf der EXPO '92, für die Stadt Illinois, welches Arbeitslosen ermöglichte, herauszukriegen, ob sie Arbeitslosengeld erhalten konnten (1993–94) sowie für das *Museum of Modern Art* in New York, welches Besuchern Informationen über Kunst und Geschichte auf ansprechende Weise vermitteln sollte.

Wie man sieht, folgen diese Beispiele alle dem gleichen Schema: Ein großer Kunde, ein eng abgegrenzter Einsatzbereich und viel Zeit und Geld. Einige dieser Projekte waren beispielsweise über mehrere Jahre in Entwicklung. Nicht für jedes Projekt ist ein solcher Zeit- und Geldrahmen angemessen oder möglich.

Goal-directed design

Goal-directed Design ist ein Ansatz der in der zweiten Hälfte der 90er Jahre von Alan Cooper entwickelt wurde. Der Kerngedanke ist, dass man herausfindet, wer genau die Nutzer sind und was sie wollen, was ihre Ziele sind. Die Idee ist also, dass, wenn ein Produkt einem Menschen hilft, sein Ziel zu erreichen, er glücklicher ist und es kauft.

Das Problem ist allerdings, dass, wenn man einen Menschen fragt, was genau er will bzw. was sein Ziel in einem bestimmten

einen Prototyp handelt, der verworfen, sondern stattdessen immer weiter verfeinert wird.

[2] beteiligte Personen

Szenarium ist, er selten eine konkrete Antwort geben kann. Weiterhin ist ein Phänomen, welches man beobachten kann, wenn Menschen längere Zeit mit einer bestimmten Software arbeiten, dass sie unbewusst Problemsituationen oder Programmfehler vermeiden sobald sie einmal oder mehrfach aufgetreten sind. Meist findet man einen Workaround, der das Problem nicht auslöst oder ähnliches. Das liegt daran, dass das Lösen eines plötzlich auftretenden Problems üblicherweise nicht zum Ziel des Benutzers gehört. Folglich sind Fragen an einen Benutzer der Art „Was gefällt Ihnen an Produkt X nicht und was müsste man besser machen?“ ebenfalls selten von Erfolg gekrönt.

Goal-directed Design verwendet hier sogenannte *Personas,* um Interaktion mit „echten“ Benutzern und die Fehleranfälligkeit dessen zu vermeiden, zumindest für Designentscheidungen. Personas sind fiktive Benutzer, für die man die Software entwirft. Alan Cooper beschreibt sie folgendermaßen:

Personas are not real people, but they represent them throughout the design process. They are hypothetical archetypes of actual users. Although they are imaginary, they are defined with significant rigor and precision.[3]

Personas sind also im Grunde Archetypen der Benutzer, die man versucht, mit der Software zu erreichen. Sie sind zwar fiktiv aber sie kriegen Namen, Gesichter, Aufgabengebiete, Kenntnisse, etc. um sie möglichst genau festzulegen.

Versucht man, ein Produkt für jeden nur möglichen Benutzer zu entwerfen, hat man am Ende eins, welches keinem zusagt. Daher ist es wichtig, die Personas sorgfältig zu wählen bzw. vor allem die Anforderungen des Produktes exakt festzulegen.

Personas lösen verschiedene Probleme, die man sonst im Entwurfs- und Entwicklungsprozess oft hat (Cooper & Reimann, About Face 2.0, 2003):

Der „elastische Benutzer“: Das Ziel ist es zwar, den Benutzer zufriedenzustellen, allerdings ist es oft nicht hilfreich, bei Designentscheidungen oder ähnlichem von „dem Benutzer“ zu sprechen, da jeder seine eigenen Vorstellungen von „dem Benutzer“ und seinen Bedürfnissen hat. Der Benutzer wird also „elastisch“, da er bei einer Entscheidung vielleicht als technisch versierter fortgeschrittener Benutzer gehandhabt wird, bei einer anderen allerdings als Anfänger ohne weitergehende Kenntnisse. Echte Benutzer haben hingegen sehr genaue Anforderungen aufgrund ihrer Ziele, Fähigkeiten und Kontexte, mit einem elastischen Benutzer zu arbeiten wird also selten ein Produkt hervorbringen, welches für wirkliche benutzbar ist.

Selbst-referenzierendes Design: Entwickler und Designer verfallen oft der Versuchung, ihre eigenen mentalen Modelle in die Software einzuarbeiten. Für den Entwickler ist se vollkommen logisch und er sieht auch meist kein Problem darin, allerdings hat der Benutzer am Ende meist Schwierigkeiten, das Programm zu verwenden, da verschiedene Mechanismen nicht *seinem* mentalen Modell entsprechen.

Design für Randfälle: Entwickler tendieren oft dazu, häufige Szenarien im Programm zu ignorieren und stattdessen mehr Aufwand in die Behandlung von Randfällen zu stecken.

[3] (Cooper, The Inmates Are Running the Asylum: Why High-tech Products Drive Us Crazy and How to Restore the Sanity, 2004)

Sicher, diese müssen auch behandelt werden, aber Benutzer bewegen sich meist eher im Rahmen des vorhersehbaren und versuchen nicht, 2 Milliarden Exemplare eines Dokuments zu drucken. Mit Personas kann man direkt fragen „Würde Julia diese Funktion oft verwenden? Würde sie sie überhaupt benutzen?“. Sie sind daher eine große Hilfe, um Funktionen gegeneinander zu priorisieren.

Personas sollten in jedem Falle auf realen Daten basieren und nicht einfach zufällig zusammengestellt werden. Mittel hierfür sind beispielsweise das bei *Contextual Design* vorgestellte *Contextual Inquiry* sowie andere Methoden, die die Benutzer direkt im Arbeitskontext beobachten und evaluieren. Weiterhin möglich, um Personas zu ergänzen, sind zum Beispiel Interviews mit den Benutzern außerhalb ihres Arbeitskontextes, Daten von anderen Stakeholdern über die Benutzer und das Hinzuziehen von Marktanalysen. Um deutlich zu machen, dass Personas wirklichen Nutzern sehr dicht nachempfunden sind, schreibt Cooper: *Almost every word in a well-developed persona's description can be traced back to user quotes or observed behaviors.* (Cooper & Reimann, About Face 2.0, 2003)

Weiterhin muss man beachten, dass es einen Unterschied zwischen Aufgaben *(tasks)* und Zielen *(goals)* gibt. Wenn Entwickler an Ziele eines Benutzers denken, fallen ihnen meist Aufgaben ein, jedoch keine Ziele. Ziele sind immer Endpunkte eines Weges oder einer Schrittfolge. Die Einzelschritte sind Aufgaben. Im Gegensatz zu Zielen, die normalerweise stabil bleiben, können Aufgaben sich ändern. Es ist daher wichtig, die Ziele im Auge zu behalten und nicht die Aufgaben. Die Software, die man erstellt, sollte sich im Endeffekt auf so wenig individuelle Aufgaben wie möglich berufen und stattdessen das Erreichen des Ziels klar in den Vordergrund stellen.

Value-centered design

Value-centered design ist ein noch sehr junges Verfahren, welches in großem Maße Bezug nimmt auf bisher existierende (unter anderem auch auf die hier vorgestellten). Grundgedanken sind hier folgende: Design ist die bewusste Erzeugung von *Value*[4] (übersetzbar wohl hier mit Wert bzw. Nutzen), Nutzen wird in der echten Welt erzeugt, nicht im System und muss daher auch dort evaluiert werden. Für diese Evaluation muss vor dem gesamten Designprozess eine Strategie erarbeitet werden und die Evaluation muss sich auf den jeweils erzeugten Nutzens konzentrieren.

Der Ablauf konzentriert sich um vier einzelne Prozesse (Cockton, 2005): *Opportunity Identification,* wo der angestrebte Nutzen für das Produkt spezifiziert wird, *Design,* wo untersucht wird, wie man durch Interaktion mit dem System eben jenen angestrebten Nutzen erreichen kann, *Evaluation* bewertet die Auswirkungen des vorgeschlagenen Designs auf die Erzeugung von Nutzen, sowie *Iteration,* wo negative Auswirkungen untersucht, verstanden und Änderungsvorschläge gemacht werden. Iteration kann dazu führen, dass ein beliebiger Schritt im Ablauf noch einmal besucht wird.

In der *Opportunity-Identification*-Phase werden zunächst die *Nutzungskontexte* untersucht, woraus Modelle und Beschreibungen der angestrebten Benutzungskontexte resul-

[4] Ich werde im Weiteren von „Nutzen“ als „Value“ sprechen, in Ermangelung einer geeigneteren Übersetzung ins Deutsche.

tieren. Als Hilfsmittel werden hier *Personas* (siehe Goal-directed design) und *Kulturdiagramme* (siehe Contextual Design) angegeben. Hieraus leiten sich dann auch die *Festlegungen des angestrebten Nutzens* ab, die im weiteren Verlauf immer wieder benötigt werden.

Die Planung der *Evaluation* sollte schon vor dem Design beginnen und beinhaltet am Anfang eine Umsetzung der gerade erstellten *Festlegungen des angestrebten Nutzens* in konkrete und messbare *Evaluationskriterien.* Diese Kriterien sind nicht immer mit *Usability Testing* messbar und man muss sich daher Gedanken machen, wie man diese in der echten Welt (und nicht im Labor) misst. Eben jenes passiert im Folgenden beim Erstellen einer *Evaluationsstrategie* und *-methoden,* wie man Nutzen überwacht und misst. User Testing (im Labor) sollte hier ebenfalls eingeplant werden, aber wirklich relevant ist im Grunde nur die Effektivität des Systems im produktiven Einsatz. Weiterhin werden daraus *Evaluationsprozeduren* erstellt, die sorgfältig dokumentiert sind und auch festhalten, welche Instrumente und Maße benutzt werden.

In der *Design*-Phase wird ein neuer Prototyp des Produkts geschaffen. Hierzu gibt es drei Einzelpunkte: *Value-delivery scenarios,* die im Grunde Ideen sind, wie man den beabsichtigten Nutzen erreichen kann, *Interaction design,* welches ein Konzept der Interaktion erstellt, sowie *Design implementation,* was ein nutzbares Produkt hervorbringt, welches wieder evaluiert wird.

Hat man einen Prototyp, so wird dieser hinsichtlich des Erreichens beabsichtigten Nutzens bewertet. Hierzu werden die vorher erstellten *Evaluationsprozeduren* angewandt, sowie in Tests mit den Benutzern Schwierigkeiten herausgearbeitet, die die Benutzer bei der Verwendung des Systems hatten. Diese *user difficulty reports* werden hinsichtlich ihrer (negativen) Auswirkung auf den Nutzen bewertet. Nur Schwierigkeiten, die sich auf den erreichten Nutzen auswirken, werden weiter betrachtet.

In der *Iterationsphase* schließt sich daraufhin die *Ursachenanalyse* an, um die Ursachen der Benutzerprobleme zu ergründen, die Nutzen entweder verringern oder ganz zunichte machen. Hier werden auch Designer und Entwickler mit einbezogen, um mögliche Ursachen zu finden. Es ist hier wichtig, tatsächlich die wirklichen Ursachen zu finden, da eine Fehleinschätzung das gesamte Design im nächsten Iterationsschritt noch schlechter machen kann. Im Folgenden werden Vorschläge für Designänderungen gemacht, an denen sich auch Designer, Entwickler, Marketing und Produktmanagement beteiligen, da alle eine unterschiedliche Sicht auf die Auswirkungen der Änderung mitbringen: Designer könnten noch einige Ideen haben, die bislang noch nicht probiert wurden, Entwickler können Schätzungen hinsichtlich der Kosten und Umsetzungsdauer einer Änderung machen, Marketing und Produktmanagement haben eine genauere Vorstellung von der Vision und dem Ziel des Produktes und wie die vorgeschlagene Änderung da hineinpasst.

Hat man nun einen konkreten Änderungsvorschlag für das Design, so fließt dieser wieder in *Design implementation* in der *Designphase* ein. Der Zyklus schließt sich dort und geht so lange weiter bis man keine Änderungen mehr machen muss.

Da die Iteration *Opportunity Identification* sowie die Planungsphase der *Evaluation* nicht beachtet, müssen diese Teile des Ent-

wurfsprozesses sehr sorgfältig durchgeführt werden, da man darauf im späteren Verlauf ständig wieder Bezug nimmt. Ein Fehler hier kann recht teuer werden.

Abschluss

Ich habe im obigen Text eine Reihe von benutzerzentrierten Designmethoden vorgestellt, die in den letzten zwei Jahrzehnten entstanden, umgesetzt und genutzt wurden. Sie unterscheiden sich hinsichtlich ihres Ansatzes sowie im Anspruch an Zeit und Budget zum Teil wesentlich, daher kann es für ein bestimmtes Problem auch keine optimale Methode geben. Wichtig ist allerdings, dass man den Benutzer in keinem Falle im Design- und Entwicklungsprozess eines Softwareproduktes vernachlässigen sollte.

Literaturverzeichnis

Beyer, H., & Holtzblatt, K. (1998). *Contextual Design: Defining Customer-Centered Systems.* San Francisco: Morgan Kaufmann.

Cockton, G. (2005). A Development Framework for Value-Centred Design. *CHI '05: CHI '05 extended abstracts on Human factors in computing systems* (S. 1292–1295). Portland: ACM.

Cooper, A. (1996). *Goal-directed Design.* Retrieved September 26, 2008, from Computer-Human Interaction in South Africa: http://www.chi-sa.org.za/articles/goal-directed.htm

Cooper, A. (2004). *The Inmates Are Running the Asylum: Why High-tech Products Drive Us Crazy and How to Restore the Sanity.* Indianapolis: Sams.

Cooper, A., & Reimann, R. M. (2003). *About Face 2.0.* New York: Wiley & Sons.

Greene, S. L., Jones, L., Matchen, P., & Thomas, J. C. (2003). Iterative Development in the Field. *IBM Systems Journal , 42* (4).

Holtzblatt, K. (2001). Contextual Design: Experience in Real Life. *Mensch & Computer* (pp. 19–22). Stuttgart: B. G. Teubner.

Johnson, J. (2000). *GUI Bloopers – Don'ts and Do's for Software Developers and Web Designers.* San Francisco: Morgan Kaufman.

UsabilityNet. (2006). *Contextual inquiry*. Retrieved September 21, 2008, from UsabilityNet: http://www.usabilitynet.org/tools/contextualinquiry.htm