RWTH Aachen University
Media Computing Group
Prof. Dr. Jan Borchers

Mensch-Machine-Interaktion
SS 2006

Action-Centered Design

Tim Just, Max Gelmroth

Matrikelnummer 257928, 257810

18. Mai 2006

Tutor: Daniel Spelmezan

Inhaltsverzeichnis

1 Einleitung

Als Grundlage für unsere Ausarbeitung diente das sechste Kapitel, *Action-Centered Design*, des Buches *Bringing Design to Software* [1].
Bei diesem Buch handelt es sich um eine Sammlung verschiedener Veröffentlichungen einzelner Fachautoren, wobei das von uns behandelte Kapitel von *Peter Denning* und *Pamela Dargan* verfasst wurde.

1.1 Über die Autoren

Peter J. Denning ist zur Zeit Professor und Vorsitzender des *Computer Science Departement* an der *Naval Postgraduate School* in Monterey, Kalifornien.
Zuvor war er an der *George Mason Univeristy* in Fairfax, Virginia als Professor tätig. Ferner war er der Präsident der *Association for Computing Machinery (ACM).*
Seine Forschungsbemühungen gelten innovativen Prinzipien, Betriebssystemen, der Zukunft des IT-Business, dem *Performance Modeling*, der Arbeitsablaufsoptimierung, dem Hochleistungrechnen und der Sicherheitsproblematik.

Pamela Dargan ist eine Softwareentwicklerin, die in allen Bereichen ihres Fachs eine langjährige Erfahrung vorzuweisen hat.
Aktuell arbeitet Sie bei einer Nonprofit-Gesellschaft in Washington, D.C. und beschäftigt sich mit der Entwicklung von Architekturen offener Systeme für die US-Regierung.

1.2 Konzept des Action-Centered Design

Beim *Action-Centered Design* handelt es sich um eine Disziplin der Softwarearchitektur und eine Unterkategorie des *Human-Centered Designs*. Bei diesem Design-Konzept konzentrieren sich die Entwickler auf den Benutzer, dessen Fähigkeiten und die von ihm gewünschten Programm-Funktionen. Daher ist es unumgänglich, mit den späteren Nutzern während der Entwicklung der Software in permanentem Kontakt zu stehen.
Wichtigstes Maß für die Qualität einer Software ist somit die Kundenzufriedenheit.

2 Softwaredesign

2.1 Wichtigkeit des Softwaredesign

Für den Erfolg einer Software ist deren Design, neben Funktionsumfang und Umsetzung, die wahrscheinlich wichtigste Entscheidung vor der Entwicklung.

Denn auch eine Software mit großem Funktionsumfang wird wenig Erfolg haben, wenn ihre Nutzer die vorhanden Features nicht nutzen können, oder sie nicht finden.
Als Anhaltspunkt für die Wichtigkeit des Designs sei hier eine Statistik des *U.S. Government Accounting Office* von 1979 angeführt:

Von den zugeteilten Mitteln für neun Software-Projekte des *Department of Defense* wurden

- **2 Prozent** ausgegeben für Software, die ausgeliefert und genutzt wurde
- **25 Prozent** ausgegeben für Software, die entwickelt und nie ausgeliefert wurde
- **50 Prozent** ausgegeben für Software, die ausgeliefert und nie genutzt wurde

Der Schlüssel zu guter Software ist nicht nur ein hohes Entwicklungsbudget, sondern vielmehr die Überwindung der ,,fundamental blindness to the domains of action in which the customers of software systems live and work"[1]. Es ist also elementar, die sich wiederholenden Prozesse im Arbeitsumfeld der Benutzer zu erkennen, diese zu abstrahieren und in die unterstützenden Softwaretechnologien einzubinden. Dieses Vorgehen wird als *Action-Centered Design* bezeichnet.

2.2 Verschiedene Varianten des Softwaredesigns

Beim Softwaredesign geht es um die Festlegung der Struktur und Funktion des zu entwickelnden Systems. Grundsätzlich gibt es zwei verscheidene Sichtweisen:

- **Software Engineering**
  Eine fest strukturierte Vorgehensweise. Die Spezifikationen des zu entwickelnden Programms stehen vor Beginn fest und es besteht nur geringer Kommunikationsfluß zwischen Entwickler und Kunde.
- **Human-Centered Design**
  Ein Konzept bei dem die Kundenzufriedenheit im Mittelpunkt steht. Die Spezifikationen entstehen hier während der Entwicklung und der Kommunikationsfluß zwischen Entwickler und Kunde ist besondern wichtig.

2.2.1 Software Engineering

Das *software engineering*, oder zu Deutsch die *Softwaretechnik*, entstand Mitte der 1960er Jahre und hat seine Wurzeln in den Ingenieurswissenschaften. Der zentrale Aspekt besteht darin, die vom Benutzer vorgegebenen Spezifikationen genau einzuhalten bzw. sie umzusetzen. Um dies zu gewährleisten, richten sich

[1] Zitiert nach Denning / Dargan in Bringing Design to Software, S. 107

die meisten Entwickler nach dem *software-lifecycle model*. Die beiden bekanntesten Varianten sind das *waterfall model* und das *spiral model*.

Das *waterfall model* (siehe Abb. 1) setzt voraus, daß die einzelnen Phasen von der Spezifikation bis hin zur alltäglichen Nutzung des Systems in einer fest vorgegebenen Reihenfolge abgearbeitet werden können. Es sind also nur Schritte von Vorne nach Hinten vorgesehen. Zwar sind theoretisch auch Rückschritte möglich, doch sind diese in der Regel zu zeitaufwendig und zu teuer. Daher kommt dieses Model heute kaum noch zum Einsatz.

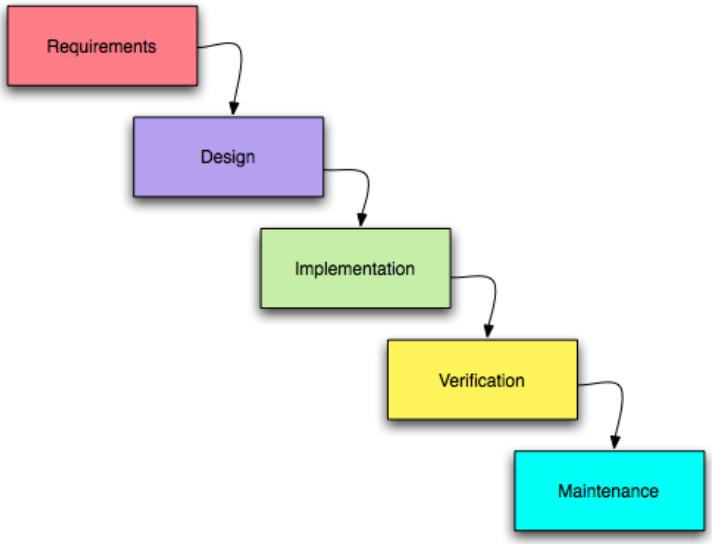

Abbildung 1: waterfall model. Entnommen aus: http://en.wikipedia.org

Das *spiral model* (siehe Abb. 2) ist eine Weiterentwicklung des *waterfall model*, bei der mehr Wert auf mögliche Wiederholungen einiger Phasen im Entwicklungsprozeß der Software gelegt wird. Dementsprechend ist dieses Modell komplexer. Es konzentriert sich allerdings immer noch auf den Entwickler und das Produkt, nicht etwa auf den Anwender bzw. die Vorgehensweisen des Anwenders.

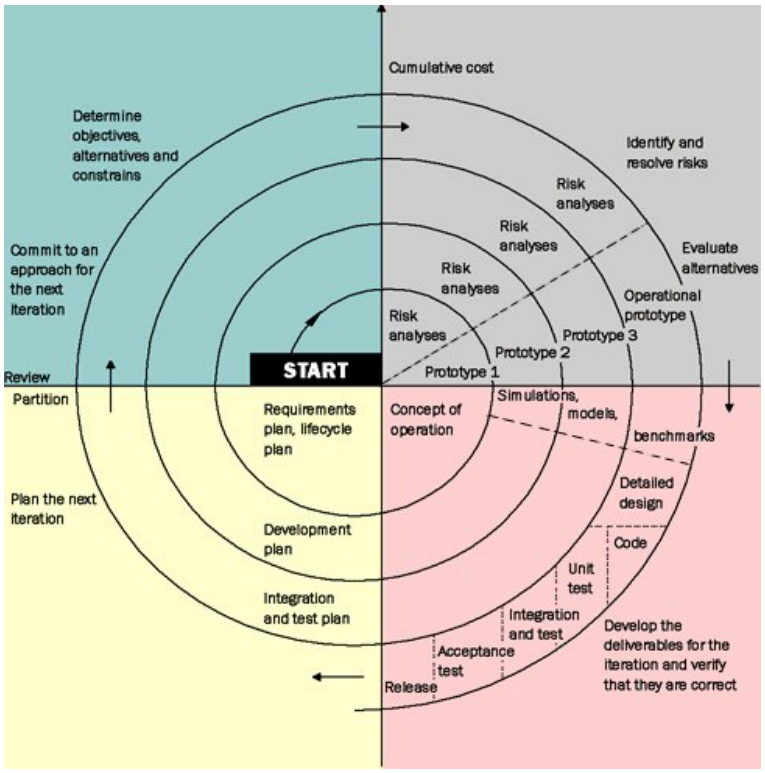

Abbildung 2: spiral model. Entnommen aus: http://en.wikipedia.org

Diese technische Betrachtungsweise der Softwareentwicklung ist abgeleitet aus den klassischen Ingenieurwissenschaften wie zum Beispiel Bauingenieurwesen, Maschinenbau und Elektrotechnik. Bei diesen ist es das Ziel, eine Systematik zu entwickeln, um die praktische Umsetzung zu realisieren.
Eine solche Systematik konnte für die Softwaretechnik jedoch bis heute nicht gefunden werden. Dies liegt wahrscheinlich daran, dass der Softwareentstehungsprozeß zu komplex ist. Dennoch existieren drei wichtige Annahmen[2], die die Softwaretechnik ausmachen:

1. Das Ergebniss des Designprozesses ist ein Produkt (ein Artefakt, eine Maschine oder ein System).
2. Das Produkt entsteht aus den vom Kunden vorgegebenen Spezifaktionen. Prinzipiell könnte man mit genug Wissen und Rechenleistung diesen Enstehungsprozeß automatisieren.
3. Sobald sich Entwickler und Kunde hinsichtlich der Spezifikationen einig sind, besteht wenig Kommunikationsbedarf zwischen ihnen.

[2] Übersetzt aus Bringing Design to Software, S. 108

2.2.2 Human-Centered Design

Dieser Ansatz entstand gegen Ende der 1980er Jahre. Im Gegensatz zum *Software Engineering*, bei dem man sich während der Entwicklung auf die Maschine und ihre Effizienz konzentriert und vom Nutzer erwartet, dass er sich den Gegebenheiten anpasst, steht beim *Human-Centered Design* der Nutzer im Vordergrund.
Beim *Human-Centered Design* werden zunächst Ideen gesammelt, wie die vom Nutzer gewünschten Funktionen umgesetzt werden können. Anschließend werden grobe Design-Entwürfe in Form von einfachen Prototypen hergestellt. Diese werden dem Kunden zur Bewertung gegeben. Mit dem erhaltenen Feedback überarbeitet der Designer seinen ersten Entwurf und verfeinert so sein Designkonzept. Dieser Kreislauf wird über mehrere Iterationen durchgeführt, wobei das Design des Produktes mit jedem Schritt konkreter wird, bis der Kunde zufrieden ist.
Dieses Vorgehen wird als *Design-Implement-Analyze-Cycle (DIA-Cycle)* bezeichnet (siehe Abb. 3).
Allgemein können Menschen kreative und kognitive Problemstellungen besser als Maschinen lösen, während Maschinen fehlerfrei Vorgänge beliebig oft wiederholen und komplexe Berechnungen durchführen können.
Daher steht es beim *Human-Centered Design* im Vordergrund die anfallenden Aufgaben zwischen Mensch und Maschine sinnvoll aufzuteilen.
Auch hier existieren drei wichtige Annahmen[3]:

1. Das Ergebnis eines guten Designs ist ein zufriedener Kunde.
2. Der Designprozeß ist eine Zusammenarbeit zwischen den Designern und dem Kunden. Das Design muss sich an veränderte Ansprüche des Kunden anpassen. Dieser Prozeß erzeugt eine Spezifikation als wichtiges Nebenprodukt.
3. Kunde und Designer stehen während des gesamten Prozesses in permanentem Kontakt.

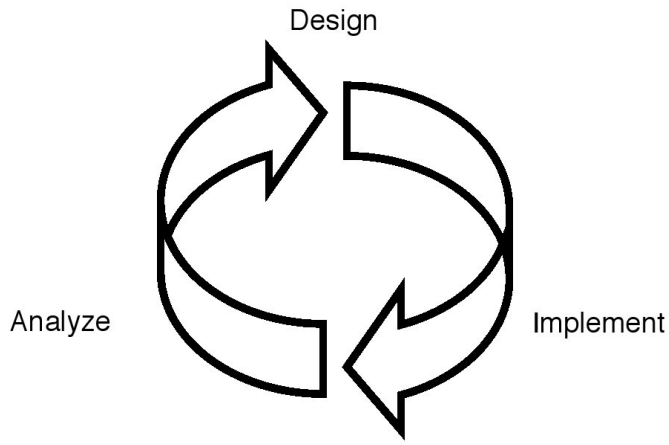

Abbildung 3: DIA-Cycle.

[3] Übersetzt aus Bringing Design to Software, S. 111

Dieses Modell verdeutlicht, dass die Kundenanforderungen nicht statisch sind und sich mit jeder Iteration des *DIA-Cycle* verändern (können), da das Design mit jedem Durchlauf konkreter wird und sich den Vorstellungen des Kunden immer mehr annähert.

Diese neue Konzeption der Softwareentwicklung verlangt den Entwicklern somit eine ausgeprägte Beobachtungsfähigkeit ab. Sie müssen das Arbeitsumfeld ihrer Kunden kennenlernen und verstehen, damit sie fähig sind Software zu entwerfen, die die Kunden effektiv bei ihren Arbeitsschritten unterstützen kann. Softwaredesigner müssen also nicht einfach nur versuchen Spezifikation umzusetzen, sondern vielmehr Standardarbeitsschritte der Benutzer in Prozesse der Software transferieren. Zu beachten gilt, dass die Frage, ob Software brauchbar ist von ihren Nutzern beantwortet wird und **nicht** von den Designern. Daher sollten bei der Softwareentwicklung immer die Nutzer im Mittelpunkt stehen. Diese wichtige Vorgabe wird beim *Human-Centered Design*, im Gegensatz zu den traditionellen Ansätzen, eingehalten.

2.3 Anhaltspunkte für erfolgreiches Design

Fragt man erfolgreiche Softwaredesigner nach dem Konzept für ein überzeugendes Design, so sind sich die meisten darüber einig, dass folgendes Vorgehen sinnvoll ist:

- Man sollte Software für einen Arbeitsbereich entwickeln, in dem viele Menschen tätig sind und in dem es viele Arbeitsschritte gibt, die von den Nutzern als kompliziert, oder nicht realisierbar erachtet werden.
- Es ist wichtig, sich mit genau diesen Arbeitsschritten auseinander zu setzen und heraus zu finden worin die Probleme der Benutzer bestehen.
- Erfolgsversprechend ist es, Funktionen zu entwickeln, die sich die Nutzer schon immer gewünscht haben oder von denen sie dachten, sie wären nicht realisierbar.
- Frühzeitiges Verteilen von Prototypen bringt hilfreiche Resonanz der Benutzer.
- Ziel ist es nicht einen schönen Code zu generieren, sondern die Nutzer zufrieden zu stellen.
- Kundenzufriedenheit ist nicht dauerhaft - Die Ansprüche wachsen desto weiter fortgeschritten die Entwicklung ist.

2.4 Ontologie und Pattern Language

Der eigentliche Entwicklungsprozeß eines *Action-Centered Designs* besteht darin eine *Ontologie* zu erstellen, diese formal als *Pattern Language* darzustellen

und die Programmierer zu koordinieren.

Eine *Ontologie* ist in diesem Zusammenhang ein System von Konzepten und Relationen zur Wissenspräsentation oder kurz: eine „Spezifikation einer Konzeptualiesrung"[4] Eine *Pattern Language* wird eingesetzt, um folgende Bestandteile einer *Ontologie* formal festzuhalten:

Die Menge der im Arbeitsumfeld

- gängigen Fachausdrücke.
- üblichen Vorgänge.
- gebräuchlichen Werkzeuge.
- möglichen Problemsituationen.
- kontinuierlichen Interessen und Bestrebungen.

2.5 Business-Process Maps

Die Zusammenhänge und Prozesse in einem Arbeitsumfeld können auch durch eine *Business-Process Map (BPM)* dargestellt werden. In ihr wird die zeitliche und logische Abfolge der Geschäftsprozesse wiedergegeben. Der Softwaredesigner kann an anhand einer BPM die sich wiederholenden Abläufe und Informationsflüsse erkennen und diese in seinen Designentwurf einbinden. Ferner kann er durch Analyse der BPM vorhandene Schwachstellen der Prozeßabläufe aufzeigen und auf diese in seinem Designentwurf reagieren.

[4] T.R. Gruber: A translation approach to portable ontologies[3].

3 Fazit

Das wichtigste Kriterium für den Erfolg einer Software ist die Kundenzufriedenheit und nicht etwa die Einhaltung fetsgelegter Spezifikationen. Daher sollte der Benutzer und sein Arbeitsumfeld fest in den Entwicklungsprozeß eingebunden werden.
Ein guter Softwaredesigner muss folglich nicht nur gute Programmierkenntnisse besitzen, sondern vielmehr auch fähig, sein fremde Arbeitsumfelder zu verstehen, die dort auftretenden Probleme zu erkennen und in seinem Softwaredesignentwurf entsprechend auf sie zu reagieren.
Um dies zu gewährleisten ist ständige Kommunikation zwischen Designer und Benutzer notwendig. Dazu ist ein Vorgehen nach dem klassischen Ansatz des *Software engineering* nicht ausreichend. Schon anhand des *waterfall models* ist zu sehen, dass Wiederholungen der Arbeitsschritte nicht vorgesehen sind.
Beim *Human-Centered Design* hingegen werden mehrere Iterationen des *DIA-Cycle* durchlaufen. Das Produkt wird so vom Prototyp bis hin zum funktionierenden System mit direktem Einfluss des Kunden verfeinert.

Literatur

1 Terry Winograd: Bringing Design to Software, Addison-Wesley, 1996

2 http://cs.gmu.edu/cne/denning/

3 T.R. Gruber: A translation approach to portable ontologies. In: Knowledge Acquistion, Band 5, Nummer 2, Seite 199-200, 1993