Abschlussarbeit
zum Thema

Beispiel-Dokument für die LaTeX-Formatvorlage

Example document for the LaTeX-template

Vorgelegt der Fakultät für Wirtschaftswissenschaften der
Universität Duisburg-Essen

von: Max Mustermann
Musterstrasse 123
12345 Musterstadt

Gutacher: Prof. Dr. Stefan Eicker
Prof. Dr. John Doe

Betreuer: Dipl.-Wirt.-Inf Some Body

Wintersemester 2013/2014

Kurzfassung

Jede wissenschaftliche Arbeit sollte eine Kurzfassung am Anfang aufweisen. Diese Kurzfassung sollte etwa eine halbe Seite lang sein.

Abstract

As all theses have an English title, there should be an English abstract as well.

Folglich sollten auch deutsche Arbeiten einen englischen Abstract haben. Falls die Abstracts zu lang werden, gibt es im Handbuch eine Hilfestellung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Kapitel
Motivation

> „There are essentially two basic approaches to design: the artistic ideal of expressing yourself and the engineering ideal of solving a problem for a customer."
>
> (Nielsen 2000, S. 11)

Die Definition nach Nielsen deckt bereits mehrere für das Web relevante Aspekte des Design-Begriffes ab:

- Zum Einen den künstlerischen Aspekt, der unter anderem der Aussage Rechnung trägt, dass Software- bzw. Web-Entwicklung immer ein kreativer Prozess ist.
- Zum Anderen aber auch den ingenieurmäßigen Aspekt, bei dem es um strukturierte Vorgehensweisen geht.

Gleichzeitig zeigt es aber auch daran angelehnt zwei Ausgangspunkte für Design: Das eigene Selbst, sowie die Anforderungen des Kunden. Auch wenn im Rahmen der Wirtschaftsinformatik sicherlich das ingenieurmäßige Vorgehen und der Kunde die Aspekte der Wahl sind, so zeigt diese einfache Definition doch schon zwei Dinge auf: Es gibt unterschiedliche Auffassungen des Begriffs und unterschiedliche Einflussfaktoren auf das Design, die beachtet werden müssen.

Denn das Design der Web-Applikationen (und damit sei an dieser Stelle sowohl die Darstellungsform, als auch der technische Entwurf gemeint) hat eindeutig Einfluss darauf, ob potenzielle Kunden die Anwendung benutzen. Damit wird der eigentliche Kern der Betrachtung offensichtlich: Die Usability von Web-Anwendungen.

Um dieses Thema zu betrachten sind zunächst einige Grundlagen im Bereich des Designs notwendig. Neben einer Begriffsdefinition und den beiden grundsätzlichen Ansätzen *Product Design* und *Interaction Design* wird dann auf die im Web in diesem Rahmen wichtigen Bereiche *Accessibility* und *Usability* eingegangen. Dabei wird darauf aufbauend nicht nur ein Überblick über Methoden gegeben, die das Produkt selbst verbessern, sondern auch darauf eingegangen, wie solche Methoden als Prozess implementiert werden können.

2 Kapitel
Grundlagen des Design

In diesem Kapitel wird zunächst in die Begriffswelt des Design eingeführt. Neben der Betrachtung des Begriffes selber werden Stakeholder identifiziert und unterschiedliche Designansätze aus unterschiedlichen Bereichen angeführt.

2.1 Definition des Begriffes Design

Für den Begriff *Design* ist eine Definition notwendig, da es bspw. ein übliches Missverständnis ist, das *Design* pauschal mit *gutem Design* gleichgesetzt wird – auch wenn dies sicherlich ein erstrebenswerter Zustand ist, so schließt Design natürlich auch sub-optimale Zustände ein, bspw. während einer Ist-Analyse und vor einer Verbesserung (Ralph und Wand 2009, S. 104-105). Ralph und Wand haben eine formale Definition für *Design* entwickelt und ermöglichen so, diesen Begriff möglichst vollständig zu erfassen:

> „**Design**
> (*noun*) a *specification* of an *object*, manifested by some *agent*, intended to accomplish *goals*, in a particular *environment*, using a set of *primitive components* satisfying a set of *requirements*, subject to some *contraints*;
> (*verb*, *transitive*) to create a design, in an environment (where the designer operates)."
> (Ralph und Wand 2009, S. 108)

Die einzelnen Elemente der Definition können nun genauer geprüft werden, um zu verstehen was die Intention hinter der Wahl dieser Begriffe war (Ralph und Wand 2009, S. 105-108):

Specification: Als Spezifikation wird hier die detaillierte Beschreibung eines Objektes, bezogen auf seine Struktur und die in ihm genutzten Komponenten, angeführt. Folglich besteht ein Design immer aus bereits bestehenden Komponenten.

Design Object: Die Entität, die entworfen wird, muss – wie im Software Engineering üblich – kein physischen Objekt sein muss. Im Kontext des Software- und Web-Engineering ist dies das erzeugte Artefakt.

Design Agent: Ein Subjekt oder eine Gruppe von Subjekten erzeugt das Design. Dabei wird üblicherweise von Menschen ausgegangen, allerdings kann auch eine Software diese Aufgabe übernehmen.

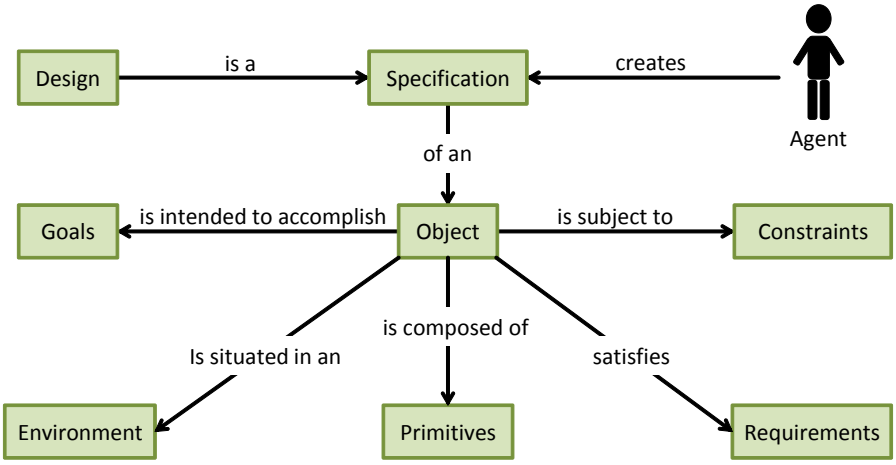

Abbildung 2.1: Definition für Design als Substantiv (Ralph und Wand 2009, S. 109)

Environment: Beim Kontext des Designs, lassen sich zwei Bereiche unterscheiden: Zunächst der Kontext, in dem das Artfefakt angewendet werden oder existieren soll und darüber hinaus auch der Kontext, in dem der Agent agiert um das Design zu erzeugen.[1]

Goals: Die Ziele eines Designs stellen erwünschte Aussagen über das Objekt dar, die auf unterschiedliche Abstraktionsgraden vorliegen können. Damit haben Sie natürlich einen Bezug zum Einfluss des Objektes in seinem Kontext[2].

Primitives: Die zur Verfügung stehenden Komponenten (bzw. Typen von Komponenten), aus denen das Design entwickelt wird.

Requirements: Strukturelle oder verhaltensorientierte Anforderungen an das Objekt. Strukturelle Eigenschaften sind dabei unabhängig von einem bestimmten Kontext oder von Stimuli. Verhaltensorientierte Eigenschaften dagegen definieren das Verhalten auf ein vorgegebenes Set an Kontextzuständen und Stimuli[3].

Constraints: Strukturelle oder verhaltensorientierte Einschränkungen für das Objekt. Also diejenigen Randbedingungen, die nicht vom Agenten, durch die Aufgabe oder durch die Komponenten vorgegeben sind.[4]

Design muss nicht in einem Produkt resultieren.

Interessant an dieser Unterscheidung ist vor allem das, was in der Disziplin des Software Engineering ebenso vorgefunden werden kann: Das Ergebnis eines Designs kann auch nur eine

[1] „For instance, the software created by a developer is intended to operate in a different environment than the developer“ (Ralph und Wand 2009, S. 107)

[2] Ziele sind dabei keine notwendige Voraussetzung für ein Design und daher optional. Allerdings bleibt anzumerken, dass ein Design, selbst wenn es kein artikuliertes Ziel verfolgt, immer noch eine Intention haben muss – sonst wäre es kein Design (Ralph und Wand 2009, S. 106-107).

[3] Folglich sind strukturelle Eigenschaften solche, die nicht von einem vorher definierten Kontext oder Stimuli abhängen

[4] „[...] physical design is still constrained by the laws of physics, virtual design by the speed and memory of the computational environment, and conceptual design by the mental faculties of the design agent.“ (Ralph und Wand 2009, S. 107)

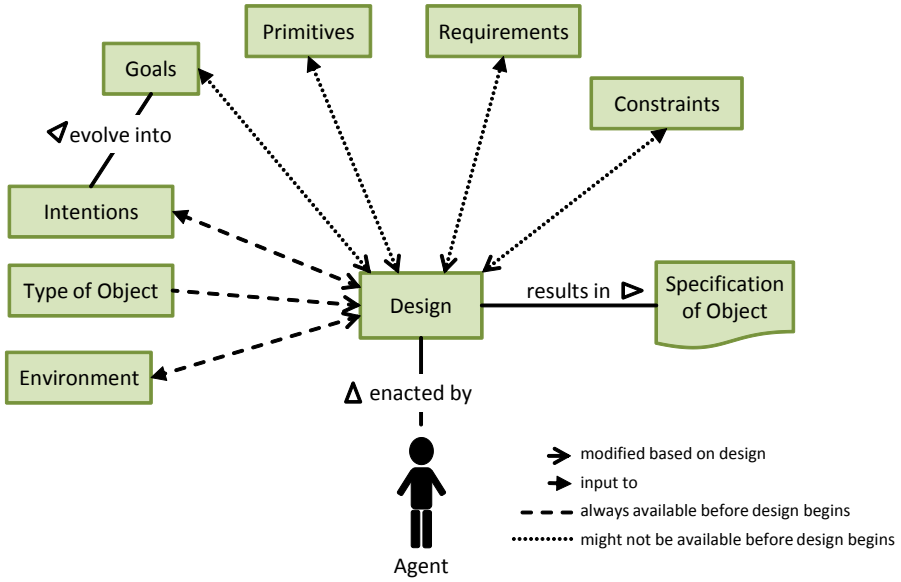

Abbildung 2.2: Definition für Design als Verb (Ralph und Wand 2009, S. 109)

Spezifikation sein, die das Objekt beschreibt und muss nicht zwingend das fertig umgesetzte Objekt sein.

Design existiert sowohl als Lebenszykluskonzept, als auch als abgeschlossene Phase.

Gleichzeitig ermöglicht das aber auch im Sinne eines Phasendenkens – wie es ja in den Vorgehensmodellen des Software Engineering vorliegt – den sowohl zeitlichen als auch rollengebundenen Abschluss einer Designphase noch bevor die Implementierung beginnt. Diese Möglichkeit der Trennung und Verschmelzung der angrenzenden Phasen gibt es natürlich auch für die Phase der Anforderungsdefinition (Ralph und Wand 2009, S. 110). Dementsprechend wird deutlich, dass der Zeit- bzw. Phasenbezug der Definition insgesamt fehlt und es damit sowohl möglich wird Design als abgeschlossene und eigenständige Phase zu definieren, als auch es als durchgängigen Prozess zu sehen. Erklärt werden kann dies durch die unterschiedliche Auffassung über die Faktoren Ziel, Kontext, Komponenten, Anforderungen und Randbedingungen. Je nach Annahme (und real in unterschiedlichen Kontexten auch so existierendem Zustand) können diese Faktoren extern vorgegeben und vollständig sein, oder erst während des Designs (oder noch später) festgelegt werden.

2.2 Verantwortliche im Design-Prozess

Betrachtet man nun das Zusammenspiel zwischen Produkt, Kunde, Unternehmen und Designer (vgl. Abbildung 2.3), so fällt zunächst auf, dass nur zwischen Unternehmen und Designer eine eindeutig notwendige Beziehung vorherrscht. Dies ist aber nur insofern auch zwingend wie das Design ein notwendiger Schritt in der Produkterstellung ist – es bedeutet nicht, dass der Designer zwangsweise auch außerhalb des Unternehmens sein muss. Darüber hinaus geschieht die notwendige Interaktion zunächst nur über das Produkt. Während aus dem Bereich des Geschäftsprozessmanagements und des Produktmarketings natürlich ganz klare Bestrebungen

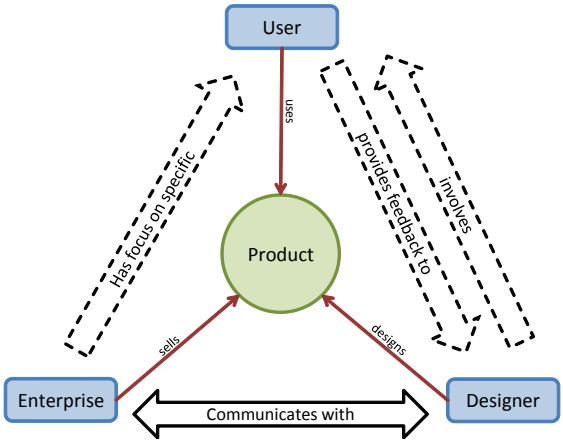

Abbildung 2.3: Beteiligungen im Design-Prozess in Anlehnung an Goodman et al. (2006, S. 42)

zu sehen sind, den Kunden in das Produkt einzubeziehen, sieht dies doch bei Software zunächst anders aus – unter anderem auch weil es oft die Programmierer sind, die Design-Entscheidungen treffen müssen:

> „Programmers have been successfully bluffing their way through the design process for so long that they are conditioned to disregard its value."
>
> (Cooper 2004, S. 117)

Diese verbreitete Einschätzung der Wichtigkeit des Designprozesses ist es auch, die dazu führt, dass viele Benutzer sich mit der verbesserungswürdigen Usability und Accessibility einer Software abfinden:

> „The difference between having a software solution for your problem and not having any solution is so great that we accept any hardship of difficulty that the solution might force on us."
>
> (Cooper 2004, S. 27)

Übertrieben gesehen führt das wie Cooper weiter ausführt sogar zu einer Haltung, dass Kunden schlechtes Design in Schutz nehmen:

> „[...] but it is surprising how many nontechnical users who are abused daily by bad interaction will excuse their oppressors [...]. [Those] are the ones who defend the computer because it can accomplish a task that was heretofore impossibly difficult."
>
> (Cooper 2004, S. 30)

Wenn es um Methoden für das Design im Bereich Web Engineering geht, werden in der Regel zwei Bereiche unterschieden: Erfahrungen, die aus dem Bereich des Design physischer Produkte resultieren (*Product Design* oder *Generic Design*) und denen, die aus dem Design von Software Produkten resultieren (*Interaction Design*).

DFX_{virtue}	$DFX_{lifephase}$
Design for environment	Design for manufacture and assembly
Design for quality	Design for end-of-life
Design for maintainability	Design for disassembly
Design for reliability	Design for recycling
Design for cost	Design for supply chain
Affective design	
Inclusive design	

Tabelle 2.1: Beispiele für „Design for X"-Techniken (Holt und Barnes 2010, S. 127)

2.3 Designansätze im Product Design

Die Designansätze des Product Design dienen der Gestaltung von Produkten allgemein und resultieren aus den Erfahrung in der Erstellung physischer Produkte. Daher treffen sie nicht unbedingt den Kern, wenn es um Software oder Web Engineering geht. Trotzdem lassen sich, über das im Software Engineering übliche Interaction Design hinaus viele Ansätze finden, die ein Potential für den Anwendungsfall zeigen. Tabelle 2.1 zeigt anhand der Einordnung unterschiedlicher „Design for X (DFX)" zeigt. Grundsätzlich geht es bei diesen Ansätzen entweder um die Gewichtung der Ziele oder Einschränkungen des Fokus.

Vergleich der Designansätze im Product Design mit dem Vorgehen im Web Engineering

Gerade Design-Ansätze die Holt und Barnes zum Bereich derer mit Lebenszyklusorientierung ($DFX_{Lifephase}$) zählen, werden im Software Engineering eher vernachlässigt, oder nur implizit berücksichtigt:

So kann man die Modularität von Softwarekomponenten grundsätzlich im montageorientiertes Design (engl.: *Design for assembly*), oder recyclingförderndes Design (engl.: *Design for recycling*) wiederfinden. Aber auch für das Fertigungs-orientierte Design (engl.: *Design for manufacture*) und Montage-orientierte Design, lassen sich weitere Parallelen sehen: So werden auf der einen Seite Richtlinien für das Design der einzelnen Komponenten eines Produktes benötigt (und sei es auch nur in Bezug auf Kosten, und ebenso werden bei der Zusammenführung Komponenten-übergreifende Richtlinien benötigt. Beispielsweise die Minimierung der Anzahl der benötigten Teile oder die Forderung, dass Teile – sofern möglich – symmetrisch gestaltet werden (Holt und Barnes 2010, S. 124). Beispielsweise durch Objektorientierung, Vererbung und die Implementierung von standardisierten Interfaces wird auch in der technischen Realisierung von Web-Anwendungen ein sehr ähnliches Vorgehen verfolgt, auch wenn es hier häufig nicht direkt um das Zusammenführen der Komponenten geht, sondern vor allem um die Wartbarkeit.Die Vorteile für Wartung und Austauschbarkeit, die sich durch so ein Vorgehen erzielen lassen sind offensichtlich; auch wenn sie im Grunde nur Nebenprodukte dieser Ansätze sind.

Ähnlich verhält es sich mit dem Ansatz des umweltfreundlichen Design (engl.: *Design for environment*). Auch wenn es einzelne Web-Projekte gibt, die spezifisch auf die Fragestellung von Umweltschutz eingehen[5], so sind die Gründe für ressourcenschonende Entwicklung in welcher Form auch immer doch meist eher in anderen Designansätzen wie Verlässlichkeits-

[5] Eine Version von Google mit schwarzem Hintergrund, mit dem Ziel den Stromverbrauch zu reduzieren, vgl. http://www.blackle.com/about/

orientiertem Design (engl.: *Design for reliability*) oder Wartungsorientiertem Design (engl.: *Design for maintainability*) zu finden.

Betrachtet man nun die Ziele dieser Ansätze (Holt und Barnes 2010, S. 127), die sich pauschal auf das Web Engineering übertragen lassen, so bleiben als Ziele des Designprozesses für das zu entwickelnde System:

- geringer Preis
- Wartbarkeit
- Verlässlichkeit
- Qualität
- Emotionale Befriedigung[6]
- Zugänglichkeit

Bei der Analyse dieser Punkte, werden zwei Kernbereiche klar, die sich im Design wiederfinden müssen: Kostenbewusstsein (geringer Preis, Wartbarkeit, Verlässlichkeit) und die Kundenzufriedenheit (Verlässlichkeit, Qualität, hohe Affektbetonung und Zugänglichkeit).

Wird die Kombination von Methoden aus den verschiedenen Ansätzen angestrebt, so ist diese solange ohne weiteres möglich, wie dasselbe Ziel verfolgt wird. So können Montage-orientiertes Design und Fertigungs-orientiertes Design, die beide primär einen geringen Preis als Ziel haben über die Gesamtkosten zusammengeführt werden. Wenn es sich jedoch um ungleiche quantitative oder gar qualitative Richtlinien handelt, gibt es keine direkte Möglichkeit zwischen den einzelnen Ansätzen und dem dadurch erzielten Nutzen abzuwägen (Holt und Barnes 2010, S. 124).

Wie findet das im Web Engineering verwendung?

Durch das Interaction Design und das Product Design werden sehr unterschiedliche Ansätze verfolgt, die sich allerdings – wie im späteren Verlauf zu sehen – zusammenführen lassen (vgl. Reed und Monk 2010). Neben diesen unterschiedlichen Ausgangspunkten der Methoden müssen aber auch die Ziele der Methoden betrachtet werden. Dies wird vor allem bei Methoden deutlich, bei denen es darum geht bestehende Designs zu verbessern: So beschäftigen sich die einen mit der Gebrauchstauglichkeit (also der Verbesserung des Nutzens eines an sich nutzbaren Produktes) während die anderen den Fokus auf die Zugänglichkeit legen (also die Vergrößerung der Nutzergruppe selbst).

Daher wird nun zunächst der Begriff der Akzeptanz aufgezeigt und darauf aufbauend auf die Gebrauchstauglichkeit und die Zugänglichkeit einzeln einzugehen, bevor diese dann in einem gemeinsamen Konzept der Universal Usability zusammengeführt werden.

[6] Eine hohe Affektbetonung soll positive Gefühle beim Nutzer während der Nutzung wecken.

3 Kapitel
Accessbility und Usability im Web

3.1 Akzeptanz

Systemakzeptanz ist nach Nielsen (1993, S. 24) die grundlegende Frage dahingehend, ob ein System gut genug ist, um jeden erwarteten Nutzen und alle Anforderungen des Benutzers (und darüber hinaus gehender Stakeholder) zu befriedigen. Dabei unterteilt Nielsen diese Akzeptanz in verschiedene Bereiche (vgl. Abbildung 3.1).

In der Kategorie *praktische Akzeptanz* finden sich die klassischen Punkte wie Kosten, Kompatibilität und Ausfallsicherheit. Zusätzlich wird hier aber auch die *Usefulness* (Nützlichkeit) betrachtet. Diese Nützlichkeit lässt sich weiter in *Utility* (Zweckmäßigkeit) und *Usability* (Gebrauchstauglichkeit) differenzieren. Zweckmäßigkeit beschreibt dabei die Erfüllung der funktionalen Spezifikation und Gebrauchstauglichkeit den Grad, wie gut die Benutzer die Funktionen nutzen können[7]. Gebrauchstauglichkeit ist für die Akzeptanz folglich nur ein notwendiger und nicht ein hinreichender Faktor (Eller 2009, S. 61). Daneben gibt es noch die *Soziale Akzeptanz*, also die Akzeptanz im sozialen Umfeld. Letztere lässt sich am einfachsten anhand eines Beispiels verdeutlichen:

Beispiel für Soziale Akzeptanz:

Ein System zur Erfassung der Pausenzeiten für Computerarbeitsplätze könnte zwar praktische Akzeptanz aufweisen, u. a. indem es die Pausenzeiten vollautomatisch erfasst und der Benutzer keine zusätzlichen Eingaben machen muss. Trotzdem könnte das System – gerade weil es seinen Zweck gut erfüllt – nicht akzeptiert werden, da die Benutzer unter Umständen nicht wollen, dass ihre Pausenzeit erfasst wird, unabhängig davon wie gut diese Erfassung funktioniert.

[7] Nielsen (1993, S. 25) schreibt, dass *Zweckmäßigkeit* bedeutet, dass das System tut was benötigt ist. Darüber hinaus führt er aber aus, dass dies nicht nur „harte“ Funktionen sein müssen, sondern auch bei einen Unterhaltungsprodukt bspw. „Spaß“ sein kann. Ob eine Systemeigenschaft also der Zweckmäßigkeit oder der Gebrauchstauglichkeit zugeordnet wird, hängt von den Anforderungen und vom Kontext ab.

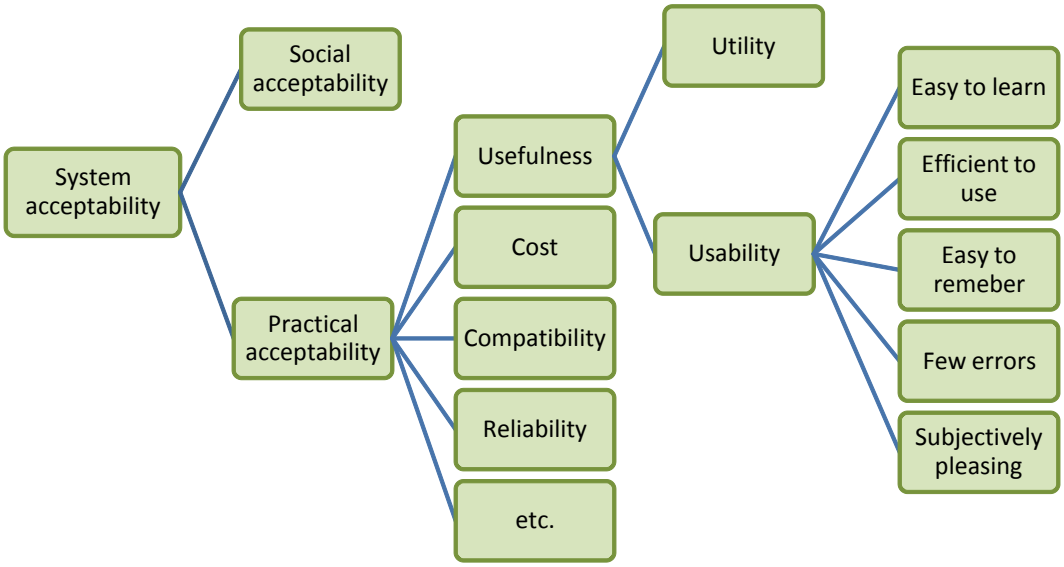

Abbildung 3.1: System Akzeptanz nach Nielsen (1993, S. 25)

3.2 Universal Usability

> „In principle, usability techniques should address the needs of the whole population – the ISO 9241 definition explicitly refers to 'specified users.' In practice though, there is a significant knowledge gap between applying usability for individuals and applying it for populations."
> (Keates und Clarkson 2003, S. 216)

Keates und Clarkson zeigt durch diese Unterscheidung bereits einen wichtigen Punkt bei der Betrachtung von Usability und Accessibility. Rein vom Grundsatz sind diese Ansätze sehr ähnlich und werden nur durch einen leicht unterschiedlichen Fokus voneinander getrennt, da die Usability auf spezifische Nutzer (bspw. die Kunden) ausgerichtet ist, während Accessibility einen von den Kunden unabhängigen Anspruch verfolgt. Betrachtet man allerdings die Ziele der beiden Ansätze und lässt externe Faktoren wie vorgegebene Kunden weg, so bleiben kaum noch wirkliche Unterschiede in der dahinterstehenden Ideologie.

Für die angewendeten Methoden macht dies aber einen Unterschied. Während die Usability mit meist klar definierten Benutzergruppen arbeitet, geht es bei der Accessibility darum eine im Grunde undefinierte Personengruppe einzubeziehen bzw. im Grunde alle Menschen. Dadurch kann ein Teil der Verfahren, die auf spezifische Benutzertypen abzielt unter Umständen nicht durchgeführt werden:

> „Many valid usability approaches exist for designing for specified users and for selecting representative end-users. Both of these are rightly considered to be important objectives that must be resolved successfully, otherwise the usability of the resulting design will be compromised. However, usability practices begin to struggle with the requirements of designing for accessibility."
> (Keates und Clarkson 2003, S. 216)

Universal Usability zielt in der Grundintention auf etwas ab, was die Disziplinen Usability und Accessibility zumindest für den Bereich IT verbindet. Dabei werden drei grundlegende Herausforderungen definiert: die technologische Vielfalt, die Diversität der Benutzer und Lücken im Wissen der Benutzer (Shneiderman 2000, S. 87). Durch diese drei Bereiche werden im Grunde alle Aspekte abgedeckt, die bisher zu den Bereichen Accessibility oder Usability genannt wurden. Dabei wird zusätzlich ein Fokus auf IT hinzugefügt, der in Kontext dieser Arbeit zusätzlich Sinn macht und dementsprechend von Lösungskonzepten außerhalb der IT abstrahiert[8].

3.3 Inclusive Design

Ein Design-Ansatz, der auf möglichst alle Nutzer fokussiert und dementsprechend der Universal Usability zuzuordnen ist, ist das *Inclusive Design*[9].

> „[Inclusive design] goes beyond user-centred design in that it caters for a wider population rather than ad hoc user groups."
>
> (Dong et al. 2003, S. 106)

Hierbei liegt nicht das Produkt direkt im Fokus und auch nicht die Interaktion mit dem Produkt, sondern die späteren Nutzer. Dadurch kann eine nutzungs- und bedarfsorientierte Sicht eingenommen werden, anstatt ein eher funktional-abstraktes Problem zu betrachten (Reed und Monk 2010, S. 295). Methodisch wird dabei die Einbeziehung der Endbenutzer in den Design-Prozess betont und ein iteratives Feedback-Verfahren vom Beginn bis zum Ende des Design-Prozesses verfolgt (Dong et al. 2003, S. 106-107).

Zur Identifizierung und Einordnung von Ursachen des Ausschlusses von bestimmten Benutzern stellen Keates und Clarkson (2003) im Rahmen des Inclusive Design die *WINIT-Waage*[10] vor (vgl. Abbildung 3.2).

Ausgangslage sind zunächst alle Menschen, auch wenn prinzipiell eine Zielgruppe geplant ist

Als größte Gruppe gibt es demnach natürlich die *Gesamtbevölkerung* (engl.: *Whole Population*). Diese stellt die Summe aller Personen dar, unabhängig Ihrer Unterschiede. Diese Gruppe ist natürlich weit größer als die angestrebte Zielgruppe bzw. *Zielgesamtheit* (engl.: *Target Population*). Aber auch die Zielgesamtheit muss in Bezug auf die Anforderungen des Produktes verkleinert werden: So gibt es Personen die Aufgrund der Problemstellung hinter dem Produkt, durch Gesetze, darüber hinausgehende Sicherheitsbestimmungen oder ihrer motorischen, kognitiven oder sensorischen Fähigkeiten, das Produkt nicht nutzen werden – und dies unabhängig von der eigentlichen Umsetzung. So schreibt das Gesetz beispielsweise ein Mindestalter für die Benutzung von bestimmten Produkten vor – bspw. bei der Steuerung von PKWs. Gehört das betrachtete Produkt zu einer solchen Kategorie, so ist beispielsweise die Konsequenz, das jüngere Personen nicht betrachtet werden müssen.

Es bleiben diejenigen, die die Anforderungen haben, die mein Produkt lösen soll

[8] Zwar kann aus den anderen Lösungskonzepten auch Ideen für die IT abgeleitet werden – ähnlich wie bei der Anwendung des Produkt Designs auf das Software Engineering – trotzdem macht diese Einschränkung aufgrund der spezifischen Anforderungen an IT- und Web-Lösungen Sinn.

[9] Die Basis bildet das Product bzw. Generic Design und Ansätze des Interaction Design wurden aufgenommen

[10] Das Kürzel *WINIT* entspricht den Anfangsbuchstaben der in der Abbildung in Beziehung gesetzten Personengruppen

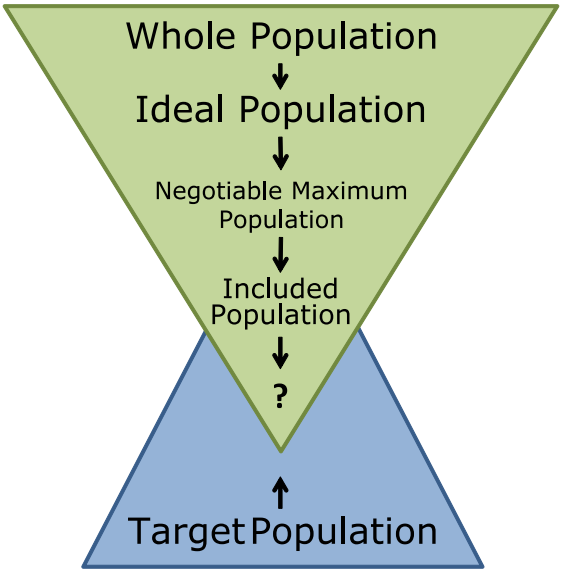

Abbildung 3.2: WINIT-Waage als Benutzergruppenkonzept im Inclusive Design (Keates und Clarkson 2003, S. 221)

So entsteht die *ideale Gruppe* (engl.: *Ideal Population*) – also der Teil, der unabhängig von den konkreten Design-Entscheidungen oder zusätzlichen Anforderungen in der Lage wäre, das Produkt zu nutzen. Für jede einzelne der Personen in dieser Gruppe gäbe es folglich einen möglichen Weg eine Lösung zu entwickeln. Allerdings gibt es in der Regel kein Produkt was für die gesamte ideale Gruppe entwickelt wird. Entweder gibt es mehrere unterschiedliche Produkte, oder es wird nur eine Teilmenge der Produkte betrachtet. Das resultiert aber lediglich daraus, dass im Verlauf des Designs des Produktes Entscheidungen getroffen werden, die bestimmte Personen benachteiligen oder ausschließen.

Aber selbst von denen, werden einige meine konkrete Lösung nicht benutzen können

Die ideale Gruppe (also die möglichen Nutzern, gemessen am Problem) unterscheidet sich dementsprechend noch von den einbezogenen Personen (engl.: *Included Population*; also der theoretisch möglichen Nutzergruppe, gemessen an der Lösung). Die Schnittmenge zwischen diesen ist die *aushandelbar-maximalen Gruppe* (engl.: *Negotiable Maximum Population*). Diese Gruppe ist nicht starr definitert, sondern richtet sich nach den Anforderungen und deren Umsetzung. Es handelt sich also um die theoretisch größte Nutzergruppe für eine Lösung, die den Anforderungen gerecht wird. Eingeschränkt wird diese dann durch Entscheidungen bei der Umsetzung der Anforderungen.

Jede einzelne Designentscheidung schließt also prinzipiell Nutzer aus

Diese Trennung der unterschiedlichen Personengruppen über die eigentliche Zielgruppe hinaus ermöglicht eine getrennte Betrachtung für die Gründe des Ausschlusses einer Person (Keates und Clarkson 2003, S. 221-222):

Verlust durch Formulierung der Anforderungen
: Das Verhältnis zwischen der aushandelbar-maximalen Gruppe und der idealen Gruppe.

Verlust durch Entscheidungen in der Umsetzung
: Das Verhältnis zwischen den einbezogenen Personen und der aushandelbarmaximalen Gruppe

Verlust vom theoretischen Problem zur praktischen Umsetzung
Das Verhältnis zwischen den einbezogenen Personen und der idealen Gruppe

Darüber hinaus lassen sich diese Bereiche auch nur auf die geplante Zielgruppe beziehen:

Zielgruppen-Verlust durch Formulierung der Anforderungen
Dies ist das Verhältnis zwischen der aushandelbar-maximalen Gruppe und der Zielgruppe

Zielgruppen-Verlust durch Entscheidungen in der Umsetzung
Dies ist das Verhältnis zwischen den einbezogenen Personen und der Zielgesamtheit

Folglich existieren im Grunde drei Fehlerkategorien beim Ausschluss der Benutzer:

1. Die Zielgruppe wird ungeschickt definiert
2. Die Anforderungen werden ungeschickt formuliert
3. Die Anforderungen werden ungeschickt umgesetzt

Diese Fehler lassen sich nur verringern

Dabei muss allerdings davon ausgegangen werden, dass Fehler in diesen Kategorien unumgänglich sind. Die Entscheidungen die im Rahmen der Anforderungsformulierung oder der Umsetzung getroffen werden, sollten deshalb stets begründet und wohlüberlegt sein. Aber selbst unter dieser Prämisse muss beachtet werden, dass ein Teil der Entscheidungen, die Personen ausschließen notwendig sind, um eine Lösung für einen Großteil der anderen Personen zu realisieren.

Für die Definition der Zielgruppe gibt es zwei unterschiedliche Dimensionen, einmal die betriebswirtschaftliche Sicht auf den gewünschten Absatzmarkt (Korrekturverfahren für Fehler hierbei sind Thema der Betriebswirtschaftslehre) und auf die einzelnen Benutzergruppen und -typen.

Betrachtet man die Definition und die Umsetzung der Anforderungen muss zunächst versucht werden diese beiden Bereiche entsprechend zu differenzieren. Man könnte anmerken, dass durch formale Methoden und Softwaretests im Software Engineering seit längerem die Umsetzung von Anforderungen geprüft wird. Selbst unter der Prämisse einer vollständigen Prüfung wäre immer noch das Problem, dass die Anforderungen nicht korrekt formuliert sind. Gleichzeitig ist eine vollständige Prüfung bei Faktoren die eindeutig subjektiv beeinflusst werden (wie die Akzeptanz) kaum vollständig oder automatisiert möglich.

Folglich sind die Anforderungen entweder so detailliert, dass Sie gut geprüft werden können, wodurch die Qualität der Anforderungsformulierung kritisch wird, oder aber die Umsetzung hat so viel Entscheidungsspielraum, dass diese kaum korrekt geprüft werden kann. Beide Szenarien zeigen wie komplex Entwurfsentscheidungen in ihren Konsequenzen sind, unabhängig davon ob es um den Entwurf eines Konzept oder schon die konkrete Implementierung geht.

Dementsprechend werden Methoden zur Erhöhung der Usability notwendig, die für unterschiedliche Phasen des Entwurfs Möglichkeiten bereitstellen Benutzer nicht auszuschließen bzw. sie angemessen zu unterstützen.

4 Kapitel
Methoden zur Erhöhung der Usability

Bei den Methoden für die Verbesserung der Usability eines Produktes kann man prinzipiell mehrere Bereiche unterscheiden: Zum einen Verbesserungen im Vorhinein, also bereits im Rahmen des Produktentwurfs. Neben den passenden Konzepten zur Betrachtung der Benutzer, wie bspw. schon durch das Inclusive Design vorgestellt, gibt es Regeln, Heuristiken und Patterns, wie bestimmte Designentscheidungen getroffen werden sollten um eine möglichst hohe Usability zu gewährleisten. Da Design aber häufig auch ein iterativer Prozess ist, was auch gerade in Bezug auf das Web Engineering durch das häufige Prototyping der Fall ist, kommen dazu auch Ansätze um ex-post, basierend auf einem bestehenden Entwurf die Usability zu verbessern. Für dieses Gebiet des Usability Testing existieren sowohl qualitative als auch quantitative Methoden.

4.1 Gestaltregeln

An dieser Stelle sei auf eine Auswahl von Gestaltregeln und Patterns verwiesen, um einen kurzen Einblick in diese zu geben. Dabei fällt natürlich auf, dass diese Regeln auf den bisherigen Ausführung, bspw. von Nielsen zur Akzeptanz aufbauen: Shneiderman und Plaisant (2010, S. 88-89) nennt bspw. acht goldene Regeln für das Interface Design:

Um Konsistenz bemühen
: In vergleichbaren Situationen sollte die Abfolge von Aktionen immer gleich sein. Darüber hinaus sollten Terminologie, Farbe und Layout für ähnliche Dinge konsistent sein. Alle Ausnahmen von dieser Regel sollten selten sein (bspw. zusätzliche Bestätigung beim Löschen, oder das Passwort-Felder keinen Klartext anzeigen (Shneiderman und Plaisant 2010, S. 88).

Beispiel für Konsistenz:

Beispielsweise zeigt das Design der Amazon-Seite in 2012 alle Schaltflächen die mit dem Kaufabschluss zu tun haben in orangen Farbtönen, während alle anderen grau hinterlegt sind. Dabei sind selbst die orangen Schaltflächen abgestuft, so dass der Kaufvorgang leicht dunkler hinterlegt ist, als Schaltflächen die nur den Einkaufswagen betreffen.

Auf universelle Usability ausrichten
: Dies entspricht den bisherigen Ausführungen in Abschnitt 3.2.

Informatives Feedback anbieten

Für jede Benutzeraktion sollte das System Feedback liefern in Abhängigkeit von der Häufigkeit und der Schwere der Aktion. So sollten komplexe oder seltene Aktionen markanteres Feedback liefern (Shneiderman und Plaisant 2010, S. 88).

Beispiel für informatives Feedback:

Der Benutzer eines Onlineshops kann erwarten, dass jede seiner Aktionen (Produkt in den Einkaufswagen legen, kaufen, etc.) entsprechend bestätigt wird. Für den Kaufabschluss selbst muss darüber hinaus ein besonderes Feedback, bspw. zusätzlich in Form einer E-Mail erfolgen.

In sich geschlossene Dialoge entwerfen

Abfolgen von Interaktion sollten gruppiert werden, so dass der Benutzer weiß, wann eine Aktionsfolge erledigt ist und sich auf die nächste Interaktion vorbereiten kann (Shneiderman und Plaisant 2010, S. 88).

Beispiel für in sich geschlossene Dialoge:

Ein übliches Beispiel ist hier der Kaufabschluss in einem Onlineshop. Hierbei wird in der Regel ein mehrstufiges Verfahren genutzt, wobei man jederzeit durch Angaben wie „Schritt 1 von 4“ oder einem Fortschrittsbalken den aktuellen Status kennt und schließlich den erfolgreichen Kauf deutlich angezeigt bekommt.

Fehlervermeidung

Das System sollte so gut es geht Fehler vermeiden. Im Falle eines Fehlers sollten Möglichkeit und Anleitung gegeben werden, um diesen zu korrigieren. Dabei sollten weitere, korrekte Eingaben nicht noch einmal eingegeben werden müssen und der Systemzustand nicht durch die Falscheingaben negativ beeinflusstg werden (Shneiderman und Plaisant 2010, S. 88).

Beispiel für Fehlervermeidung:

Üblicherweise ist eine Empfehlung (auch von Shneiderman und Plaisant) hierzu bspw. bei Adresseingaben fehlende Postleitzahlen zu monieren und die anderen Adressdaten zu bewahren. Stattdessen müsste anhand der anderen Adressdaten die korrekte Postleitzahl ermittelt werden und durch den Benutzer bestätigt werden.

Erlauben, die Aktionen rückgängig zu machen

Benutzer sollten in der Lage sein, Aktionen oder Gruppen von Aktionen rückgängig zu machen. Dadurch wird der Stress beim Benutzer vermindert und er ist eher bereit sich mit unbekannten Funktionen auseinanderzusetzen (Shneiderman und Plaisant 2010, S. 89).

Beispiel für Rückgängig-Funktion:

So sollte ein Online-Shop bspw. sowohl die Möglichkeit bieten einzelne Produkte aus dem Warenkorb zu entfernen, als auch gesamte Bestellungen zu stornieren. Datengetriebene Systeme sollten dementsprechend eine Versionierung der einzelnen Datensätze ermöglichen, oder zumindest die letzte Version vorhalten.

Interne Kontrollüberzeugung bewahren

Erfahrene Benutzer wünschen, dass das System nach ihren Erfahrungen reagiert. Dementsprechend fühlen Sie sich in der Kontrolle über das System und verlangen, dass es sie entsprechend bei komplexen Eingabevorgängen unterstützt. Genauso finden Sie es ärgerlich wenn Sie ein gewünschtes Ergebnis nicht realisieren können (Shneiderman und Plaisant 2010, S. 89).

Beispiel für Kontrollüberzeugung:

Im Rahmen von Design-Änderungen von Webpräsenzen und -applikationen werden häufig immer geringere Änderungen durchgeführt, bspw. nur Änderungen im Layout, oder nur in der Farbgebung, oder nur in der Reihenfolge des Inhalts Nielsen (2012). Dadurch kann der Benutzer trotz einer Änderung immer noch gewohnt mit der Seite interagieren.

Kurzzeitgedächtnis schonen

Der Benutzer soll sich möglichst wenig Dinge merken müssen. Beispielsweise soll verhindert werden, dass zuviele Elemente auf der selben Seite vorhanden sind (Die Faustregel besagt, dass 7 ± 2 Dinge im Kurzzeitgedächtnis speicherbar sind) (Shneiderman und Plaisant 2010, S. 89).

Beispiel für Kurzzeitgedächtnis schonen:

Dementsprechend sollte jede Navigationsebene nicht mehr Elemente enthalten. Auch bei mehrseitigen Dialogen, sollte die Anzahl der Seiten möglichst reduziert werden, und bei Bedarf entsprechende Informationen wiederholt werden.

Ähnlich zu den Bausteinen der Akzeptanz nach Nielsen, stellen Dix et al. (2004, S. 260-273) zusätzliche Prinzipien vor die einzuhalten sind, die drei Kategorien zugeordnet werden können: **Erlernbarkeit** betrachtet die Leichtigkeit mit der ein neuer Benutzer in der lage ist effektiv mit dem System zu interagieren und bestmögliche Leistung zu erzielen. Dazu zählen die Prinzipien *Vorhersehbarkeit*, *Nachvollziehbarkeit*, *Vertrautheit*, *Verallgemeinbarkeit* und *Konsistenz*. **Flexibilität** betrachtet die Vielzahl der Möglichkeiten mit denen Benutzer und System Informationen austauschen können. Dazu zählen die Prinzipien *Dialog-Initiative*, *Nebenläufigkeit*, *Aufgaben übergeben*, *Austauschbarkeit* und *Anpassbarkeit*. **Robustheit** betrachtet den Grad der Unterstützung für den Benutzer um sein Ziel zu erreichen und die Zielerreichung zu prüfen. Dazu zählen die Prinzipien *Überprüfbarkeit*, *Wiederherstellbarkeit*, *Antwortverhalten* und *Aufgabenangemessenheit*. Zum besseren Verständnis werden diese Prinzipien nun verdeutlicht:

Vorhersagbarkeit

Ein Nutzer sollte die Konsequenzen seiner nächsten Aktion im Kontext der bisherigen und im Kontext des aktuellen Systemzustandes vorhersagen können. Dabei kann der Grad des notwendigen Wissens über den Kontext variieren. Darüber hinaus muss bekannt sein, welche Aktionen im aktuellen Zustand überhaupt verfügbar sind Dix et al. (2004, S. 261-262).

Beispiel für Vorhersagbarkeit:

Im Rahmen der Kauf-Transaktion in einem Online-Shop sollte der Benutzer jederzeit wissen, welche Schaltfläche welche Konsequenz hat, also bspw. das nach dem Legen eines Produktes in einem Warenkorb immer noch die Anzahl verändert werden kann (bspw. da Sie vorher nicht angegeben werden kann). Vor allem im Rahmen der Angabe von Lieferadresse und Zahlungsdaten sollte klar sein, ob noch eine letzte Bestätigungsseite kommt, bevor die Transaktion durchgeführt wird. Dies könnte bspw. durch einen Fortschrittsbalken mit beschrifteten Schritten erfolgen.

Nachvollziehbarkeit

Ein Benutzer sollte die Konsequenzen seiner Aktion prüfen können. Dies dient als Grundlage für das Verständnis des Systems und damit für die *Vorhersagbarkeit* (Dix et al. 2004, S. 262).

Vertrautheit

Benutzer sollten aufgrund ihrer Erfahrung in der Realwelt und mit anderen Systemen auch mit neuen Systemen interagieren können (Dix et al. 2004, S. 263-264).

Verallgemeinbarkeit

Das Verhalten einer Applikation oder der Applikations-Umgebung wird durch den Benutzer auch auf andere Funktionen übertragen (Dix et al. 2004, S. 263).

Beispiel für Verallgemeinbarkeit:

Wenn also bspw. das zugrundeliegende Betriebssystem für Symbole Drag-and-Drop unterstützt, kann der Benutzer davon ausgehen, dass Applikationen die Symbole ähnlich nutzen, auch Drag-and-Drop unterstützt.

Konsistenz

Da Konsistenz immer einen Bezugspunkt haben muss, könnte man nach Dix et al. (2004, S. 264-265) viele der anderen Prinzipien der Konsistenz zuordnen . Grundsätzlich entspricht dieses Prinzip auch hier der obigen Definition nach Shneiderman und Plaisant.

Dialog-Initiative

Die Aktion im Rahmen der Interaktion zwischen Benutzer und System können von beiden Partnern aus initiiert werden. Dementsprechend kann es Situation geben in denen der Benutzer eine Aktion frei startet, aber auch Situationen in denen das System einen Dialog öffnet und eine explizite Antwort durch den Benutzer erwartet (Dix et al. 2004, S. 266-267).

Beispiel für Dialog-Initative:

Häufig hat der Benutzer die Initiative wenn es darum geht eine Entscheidung bspw. für eine Funktion zu treffen. Bei Systemmeldungen, die unabhängig von der aktuellen Aufgabe sind, hat das System die Initiative.

Nebenläufigkeit

Ein System unterstützt Nebenläufigkeit, wenn es aus der Sicht des Benutzers in der Lage

ist, mehrere Dinge gleichzeitig zu erledigen. Dadurch wird ermöglicht, dass der Benutzer einer primären Aufgabe nachgehen kann und andere Aufgaben weiterhin nebenbei durchgeführt werden können (Dix et al. 2004, S. 267).

Beispiel für Nebenläufigkeit:

Ein einfaches Beispiel dafür wäre ein regelmäßiges, automatisches Speichern eines Dokuments im Hintergrund während der Bearbeitung. Wichtig hierbei ist, dass die Nebenläufigkeit aus Sicht des Benutzers betrachtet wird. Es ist folglich nicht relevant ob wirklich unterschiedliche Prozesse, Threads, Prozessoren oder Systeme verwendet werden.

Aufgaben übergeben

Das Übergeben von Aufgaben ist vor allem notwendig bei Aktionen, die grundsätzlich automatisiert werden können, aber trotzdem an einigen Stellen menschlichen Eingriff benötigen, bspw. aufgrund von kritischen Entscheidungen (Dix et al. 2004, S. 268).

Beispiel:

Selbst wenn ein System Versionierung unterstützt (bspw. ein Wiki) kann es Fälle geben in denen einzelne Versionsstände, zum Beispiel aus Gründen des Urheberrechts gelöscht werden müssen. Dabei könnte die Löschung von mehreren Versionsständen vom Benutzer an das System übergeben werden. Das System ermittelt das für einen Großteil der gelöschten Objekte kein Problem besteht, da die Änderungen lediglich die Zeichensetzung betreffen und nicht kritisch sind. Ein gelöschtes Objekt umfasst aber mehrere Absätze als Änderung. Hierbei gibt das System eine Rückfrage ob die Löschung wirklich gewünscht ist.

Austauschbarkeit

Die Austauschbarkeit betrifft sowohl den Input als auch den Output. Im Rahmen des Inputs sollte das System unterschiedliche Formen der inhaltlich selben Eingabe unterstützen. Bezogen auf den Output sollte das System auch dort unterschiedliche Formen unterstützen (Dix et al. 2004, S. 268-269).

Beispiel für Austauschbarkeit:

Ein gutes Beispiel für die Austauschbarkeit von Input-Werten sind bspw. Größenangaben in einer CSS-Datei. Hier werden unterschiedliche Größenangaben für Elemente oder Abstände, bspw. in absoluten Größen (Pixel, Zentimeter) oder relativen Größen (in Bezug zur Schriftgröße oder zur Größe des Elternelements) erlaubt. In CSS3 sind diese Werte sogar beliebig rechnerisch kombinierbar[11], also bspw. „80% - 40px“.

Ein gutes Beispiel für die Austauschbarkeit des Outputs sind Web-Content-Management-Systeme. Hier werden die selben Inhalte in unterschiedlicher Form ausgegeben, bspw. für die HTML-Darstellung und für die RSS-Darstellung.

Individualisierbarkeit

Hierbei geht es um Veränderungen des Systems in Abhängigkeit vom Benutzer. Dabei wird unterschieden nach Anpassungsmöglichkeit (engl.: *Adaptability*), also der Möglichkeit,

[11] Bezogen auf die Funktion *calc* im CSS3 Candidate Recommendation vom 28. August 2012, die Funktion könnte in späteren Versionen entfernt sein.

dass der Benutzer aktiv das System anpasst, und Anpassungsfähigkeit (engl.: *Adaptivity*), also der Möglichkeit dass sich das System automatisch an den Benutzer anpasst (Dix et al. 2004, S. 269-270).

Beispiel für Individualisierbarkeit:

In einem Einstellungsdialog wäre die Auswahl „Expertenoptionen anzeigen“ eine Anpassungsmöglichkeit zu realisieren. Ein weiteres prägnantes Beispiel wären Dashboards mit frei positionierbaren Widgets.

Die Anpassungsfähigkeit dagegen zielt explizit auf passives Verhalten des Benutzers ab, also darauf, dass das System selbst entscheidet welche Anpassung vorgenommen wird. Dementsprechend wäre ein einfaches Rollenmodell, bei dem Benutzer unterschiedliche Rollen mit unterschiedlichem Funktionsumfang innehaben, noch keine wirkliche Anpassungsfähigkeit. Eine komplexe Eingabezeile im Browser, die für die Autovervollständigung die bisherige URLs und Suchanfragen kombiniert und nach Abrufdatum und Häufigkeit priorisiert dagegen schon.

Überprüfbarkeit

Der Benutzer sollte in der Lage sein den aktuellen Zustand des Systems zu erfassen. Hilfreich dabei ist, wenn das System navigierbar ist, wodurch unterschiedliche Sichten auf den aktuellen Zustand ermöglicht werden. Dabei muss klar sein auf welche Sichten zu welcher Zeit Zugriff besteht. Zusätzlich helfen Standard-Vorgaben dabei den aktuellen Zustand zu ermitteln. Gleichzeitig ist es wichtig, dass alle System-Meldungen einen gewissen Grad an Persistenz haben, damit Sie nicht vergessen werden und der aktuelle Zustand des Systems weiterhin ermittelbar ist.

Wiederherstellbarkeit

Benutzer sollten in der Lage sein Fehler ungeschehen zu machen. Dabei ist es mindestens notwendig von einem Fehler ausgehend solange Korrekturen vornehmen zu können bis der gewünschte Zustand erreicht ist. Im besten Fall soll aber zusätzlich der vor dem Fehler vorhandene Zustand wiederherstellbar sein. Dabei kann die Fehlerkorrektur automatisiert durch das System erfolgen oder duch den Benutzer initiiert werden.

Antwortverhalten

Das Antwortverhalten eines Systems wird durch die Reaktion eines Systems auf Aktionen des Benutzers definiert. Dabei sind auf der einen Seite möglichst sofortige Reaktionen wünschenswert. Darüber hinaus ist es aber genauso wichtig, dass die Reaktionsgeschwindigkeit über mehrere Aktionen hinweg stabil ist - folglich ein Aspekt der gerade im Web besondere Betrachtung benötigt.

Aufgabenerfüllung

Das System sollte in der Lage sein, den Benutzer bei seinem Ziel zu unterstützen. Dabei wird unterschieden nach der Vollständigkeit, also dem Grad der Aufgaben bei denen das System Unterstützung bietet und der Angemessenheit, also dem Grad wie gut die Aufgabe erfüllt wird. Dabei wird nicht die Sicht der Spezifikation eingenommen, sondern die des Benutzers. Folglich könnte ein System entsprechend seiner Spezifikation vollständig definiert sein und trotzdem gibt es Benutzer die Defizite im Funktionsumfang bemängeln.

Dabei müssen all diese Regeln und die darunterliegenden Prinzipien für jede Umgebung neu interpretiert, aufbereitet und erweitert werden (Shneiderman und Plaisant 2010, S. 89). Dabei wird klar, dass sich die daraus ableitbaren Heuristiken auch in Bezug auf die fortschreitende technische Entwicklung des Webs hin anpassen müssen.

Die Anwendung dieser Regeln bezogen auf konkrete Problemstellungen findet sich in Web Patterns wieder. Dabei muss allerdings beachtet werden, dass diese aufgrund der fortschreitende technische und sozialen Entwicklung des Webs jedoch häufig das Problem haben, dass Sie veralten können. Solche Pattern-Kataloge finden sich bezogen auf das Web allgemein bspw. bei Graham (2003) oder bezogen explizit auf das Social Web bspw. bei Crumlish und Malone (2009).

Literaturverzeichnis

Cooper A (2004) *The Inmates Are Running the Asylum.* 6. Aufl., Sams, Indianapolis. ISBN:9780672326141

Crumlish C, Malone E (2009) *Designing Social Interfaces: Principles, Patterns and Practices for Improving the User Experience.* 1. Aufl., O'Reilly & Associates, Sebastopol. ISBN:978-0-596-15492-9

Dix A, Finlay J, Abowd GD, Russell B (2004) *Human-Computer interaction.* 3. Aufl., Pearson Education, Harlow. ISBN:9780130461094

Dong H, Keates S, Clarkson PJ, Cassim J (2003) *Implementing Inclusive Design: The Discrepancy between Theory and Practice.* In: Carbonell N, Stephanidis C (Hrsg.) Universal Access (Lecture Notes in Computer Science (LNCS), 2615) (Lecture Notes in Computer Science (LNCS), 2615). Springer, Tokyo, S. 106–117. doi:10.1007/3-540-36572-9_8

Eller B (2009) *Usability-Engineering in der Anwendungsentwicklung: Systematische Integration zur Unterstützung einer nutzerorientierten Entwicklungsarbeit: Zugl.: Dissertation, TU Darmstadt, 2009.* 1. Aufl., Gabler, Wiesbaden. doi:10.1007/978-3-8349-8459-3

Goodman J, Langdon PM, Clarkson PJ (2006) *Providing Strategic User Information for Designers: Methods and Initial Findings.* In: Clarkson PJ, Langdon P, Robinson P (Hrsg.) Designing Accessible Technology, S. 41–51. Springer, London. doi:10.1007/1-84628-365-5_5

Graham I (2003) *A pattern language for Web usability.* Addison-Wesley, London. ISBN:0201788888

Holt R, Barnes C (2010) *Towards an integrated approach to "Design for X": an agenda for decision-based DFX research.* In: Research in Engineering Design 21(2):123–136. doi:10.1007/s00163-009-0081-6

Keates S, Clarkson PJ (2003) *Countering design exclusion: bridging the gap between usability and accessibility.* In: Universal Access in the Information Society 2(3):215–225. doi:10.1007/s10209-003-0059-5

Nielsen J (1993) *Usability engineering.* Reprint 2008, Kaufmann, San Diego. ISBN:0-12-518406-9

Nielsen J (2000) *Designing web usability.* Reprint, New Riders Publ, Indianapolis. ISBN:156205810X

Nielsen J (2012) *Homepage Design Changes.* In: Jakob Nielsen's Alertbox ISSN:1548-5552. http://www.useit.com/alertbox/homepage-design-change.html (Abruf am 1. August 2015)

Ralph P, Wand Y (2009) *A Proposal for a Formal Definition of the Design Concept.* In: Lyytinen K, Loucopoulos P, Mylopoulos J, Robinson B (Hrsg.) Design requirements engineering, S. 103–136. Springer, Berlin. doi:10.1007/978-3-540-92966-6_6

Reed D, Monk A (2010) *Inclusive design: beyond capabilities towards context of use.* In: Universal Access in the Information Society S. 295–305. doi:10.1007/s10209-010-0206-8

Shneiderman B (2000) *Universal usability.* In: Communications of the ACM 43(5):84–91. doi:10.1145/332833.332843

Shneiderman B, Plaisant C (2010) *Designing the user interface: Strategies for effective human-computer interaction.* 5. Aufl., Pearson Education, Upper Saddle River ISBN:9780321601483

Eidesstattliche Versicherung

Ich versichere an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Essen, 9. Dezember 2025
Ort, Datum

Unterschrift