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

Human Computer Interaction
SS 2006

A Software Design Manifesto

Hadi Leoparjo Loei (244580)
Norbert Wiechowski (250862)

13.April 2006

Tutor: Daniel Spelmezan

Contents

1 Vorwort

Das Ziel des Buches *Bringing Design to Software*[1] ist die Programmierung der Benutzerschnittstelle zu verbessern. Der Autor versucht diese Aufgabe zu lösen, indem er die Prioritäten eines Software-Designs aus den verschiedensten Blickwinkeln betrachtet. Aus diesem Grund wurden nicht nur die Meinungen ausgebildeter Experten eingeholt, sondern auch Ansichten von Lehrern. Gemeinsam wurde untersucht, auf welche Art und Weise Lehren aus allen Bereichen des Designs auf Software übertragen werden können.
Dieses Buch ist weniger ein Programmieranleitung für Benutzerschnittstellen, sondern eher eine Ansammlung von Ideen welches Design funtioniert und wie sich dieses noch optimieren lässt. Jedes Kapitel wirft zwei grundlegende Fragen auf:

- Was versteht man unter Design und dem zugrunde liegenden Prozess?
- Wie kann unsere Software durch das Verständnis von Design verbessert werden?

Eine Zielgruppe des Buches sind die Endanwender der Software. Hätten diese ein tieferes Verständnis über Anwendungen, wäre die Computerindustrie gezwungen, bessere Designs zu entwickeln. Das Ziel des Buches ist also unter anderem, dem Leser verständlich zu machen, wozu gute Software fähig ist.

2 Einführung

In der heutigen Welt werden PCs eine unverzichtbare Rolle zugeordnet. Betriebssysteme, Programme zur Textverarbeitung und ähnliche, für Benutzer vereinfachte Software sind schon seit Jahren auf dem Markt. In den 90er Jahren verlagerten sich die Schwerpunke, die essentiell für gute Software sind: Das Design rückte immer mehr in den Mittelpunkt.

> "Despite the enormous outward success of personal computers, the daily experience of using computers far too often is still fraught with difficulty, pain, and barriers for most people....The lack of usability of software and the poor design of programs are the secret shame of the industry."

Das Manifest von Mitchell Kapor bewirkte eine positive Änderung in der Softwareindustrie bezüglich des Software-Designs. Auf folgende Problematiken gehen wir detaillierter ein:

- Was ist eigentlich Software-Design?
- Wie unterscheidet sich Software-Design von der üblichen Programmierung?
- Wie unterscheidet sich Software-Design zu Designs aus anderen Bereichen?

2.1 Was ist Software-Design?

The Association for Software Design (ASD) formulierte eine Antwort auf diese Frage:

> Software design sits at the crossroads of all computer disciplines: hardware and software engineering, programming, human factors research, ergnomonics. It is the study of intersection of human, machine, and the various interfaces - physical, sensory, psychological - that connect them.

Im Vergleich zu den Entwicklungen der Computerwissenschaften, der man auch die Programmierung zuordnen kann, ist Software-Designs und seine Entstehung noch nicht genau definiert.
Ein Programm funktioniert genau dann richtig, wenn die algorithmische Grundlage des Programms für jede Eingabe die gewünschte Ausgabe liefert. Um ein *funktionierendes Design* zu erstellen, ist es nötig, den Blickwinkel von der Algorithmik der Programmkonstruktion abzuwenden und sich auf die Benutzerfreundlichkeit für den Endandwender zu konzentrieren. Die wichtigen Merkmale eines Programms aus der Sicht des Software-Designers sind also andere, als die des Programmierers.

2.2 Was ist Software?

Software dient als Medium zur Kreation einer virtuellen Welt, in der Anwender agieren und reagieren können. Denkt man an virtuelle Welten, denkt man häufig zunächst an Computerspiele. Im Folgenden wollen wir uns allerdings nicht nur darauf beschränken. Virtuelle Welten existieren ebenso in ähnlicher Weise in Benutzerschnittstellen (GUI) oder auch im Internet. Neben der Funktion als Vermittler zwischen Software und Anwender erzeugen Programme einen Arbeitsraum, in dem der Benutzer sein *virtuelles Leben* führt und die Software seinen persönlichen Bedürfnissen anpasst: Skins, Shortcuts usw.

2.3 Was ist Design?

Design ist nichts Greifbares. Vielmehr wird in dem Buch versucht, ein Blick für das Wesentliche am Design zu vermitteln. Um die Merkmale eines guten Designs zu verstehen, erläutern Personen aus verschiedenen Lebensbereichen ihre Ansichten dazu. Es gibt bisher kein festes Konzept für die Entstehung von Design.
Wir betrachten die Entwicklung einer Stadt: Eine Stadt entwickelt sich, ohne von Beginn an einen Plan für die gesamte Stadt zu verfolgen. Man kann niemals sagen: Diese Stadt ist fertig gebaut. Eine Stadt entwickelt sich, solange Menschen in ihr leben. Komplexe, aber sehr gut funktionierende Systeme können also auch entstehen, ohne ein Design von Beginn an in den Vordergrund zu stellen. Ist man gezwungen dem Design einen Wert zu zuordnen, kann dieser von verschiedenen Faktoren abhängen. Ein Beispiel: Durch das Blocksystem vieler Städte in Amerika wurde ein Versuch gemacht, Städte einer umfassenden Designstruktur zu unterwerfen. Dies auch durchaus erfolgreich. Aber kann man deswegen sagen, dass beispielsweise New York schöner bzw. besser ist als Rom oder Paris? Im Bereich der Software zählen zu den Werten eines guten Designs auch Eigenschaften, die beispielsweise

die Bedienung für Personen mit Behinderungen erleichtern uvm. Wo ist welches Design sinnvoll? Nach welchen Faktoren kann man Design bewerten? Was ist Design?

2.3.1 Design dient der Benutzerfreundlichkeit

Aufgrund der bisher gezogenen Schlüsse kann man Software auch als eine Benutzerschnittstelle betrachten, die üblicherweise mit Hilfe von Peripheriegeräten über einen Monitor bedient wird. Design spielt daher nicht nur bei Software eine große Rolle, sondern auch bei Peripheriegeräten. Sie unterliegen ständigen Optimierungen, Anpassungen an den Benutzer und müssen unter Umständen neu entwickelt werden.

2.3.2 Design ist ein kreativer Prozess

Design kann man nicht auf standardisierte Methoden reduzieren. Ebenso wenig ist Design ein Prozess zur Problemlösung. Design-Entwicklung ist sowohl ein kreativer als auch ein *chaotischer Prozess*. Es beginnt mit die Bloßstellung des Problems und endet, sobald der Designer erkennt was Menschen brauchen könnten. Im Allgemeinem sind sich die Menschen nicht einmal bewusst, dass ihnen die Neuentwicklung des Designers fehlt.

2.3.3 Design erzwingt Kommunikation

Design kann unerwartete und negative Reaktionen bei den Anwendern auslösen. Angenommen die Struktur des Programmmenüs ist für den Anwender unlogisch aufgebaut und damit unbedienbar. In solchen Fällen ist es von Bedeutung, die Kritik der Anwender zu akzeptieren und die Qualität des betreffenden Objektes zu erhöhen bzw. spezielle Teilbereiche neu zu gestalten.
Um als Software-Designer erfolgreich zu sein ist es wichtig, die Meinungen anderer Designer zu tolerieren und als Grundlage neuer Forschungen zu akzeptieren. Weiter ist es von immenser Bedeutung, einen gemeinsamen Nenner für offene Diskussionen mit Programmierern zu finden. Dies ist der richtige Weg, um sich um Erfahrungen zu bereichern. Ein gutes Design muss übermittelbar sein: Der Benutzer muss fähig sein, die Software intuitiv zu bedienen.

2.3.4 Design ist eine gemeinsame Leistung

Ein fertiges Design ist selten die Leistung einer einzelnen Person. Der Designer ist abhängig von Inspirationen. Die Ursachen für bleibende Eindrucke können verschiedenster Art sein: Von anderen Menschen und deren Handlungen, künstlichen oder natürlichen Gebilden uvm. Deutlich spürbar ist dies in großen Firmen, die es sich nicht leisten können ihre repräsentativen Leistungen von einer Person abhängig zu machen. Dort kommen viele verschiedene Faktoren und Meinungen zusammen, die Einfluss auf das endgültige Ergebnis haben.

3 Einleitende Worte über das Manifest

Mitchell Kapor kann man ohne Zweifel als einen Pionier des Software-Designs bezeichnen. 1980 entwickelte er zusammen mit Jonathan Sachs das Programm Lotus 1-2-3. Hauptberuflich beschäftigt er sich mit der Entwicklung neuer Software. Allerdings wusste er keine passende Bezeichnung für seine Tätigkeit. Daher wandte er sich im Jahre 1990 an die Öffentlichkeit und machte einen Aufruf, eine neue Berufsbezeichnung einzuführen: Die des Software-Designers. Dieses Manifest wollen wir im Folgenden genauer analysieren.

4 Das Manifest

Dank des stark wachsenden Erfolges der Heimcomputer wird die Computerindustrie nicht nur mit der Produktion der PCs gleichgesetzt, sondern sogar durch diese definiert. Große Firmen und Forschungszentren konzentrieren sich momentan hauptsächlich auf Standardisierungen. Dies betrifft sowohl Plattformen, Anwendungen, als auch die Möglichkeiten der Kommunikation. Eine dieser Standardisierungen wäre beispielsweise das Internetprotokoll (IPv4). Dabei wird die einfache Handhabung der Software für den normalen Anwender häufig außer Acht gelassen. Dies führt dazu, dass der Anwender teilweise nicht fähig ist, die Software zu bedienen und ihre Möglichkeiten aus zu schöpfen. Der Anwender gibt sich selbst die Schuld für seine Unfähigkeit, hält sich aber auch nicht für qualifiziert genug, seine Kenntnisse dementsprechend zu erweitern und nimmt es stillschweigend hin. Vorteile ergeben sich daraus weder für den Anwender, noch für die Entwickler der Software.

> *"The lack of usability of software and the poor design of programs are the secret shame of the industry"*

Dies legt den Schluss nahe, dass die wichtigste aller Entwicklungen innerhalb der Computerbranche die wäre, dem Beruf des Software-Designers eine entscheidende Rolle beizumessen: Die Ursache dafür liegt in der Technik des Programmierens. Die Fertigkeiten, die ein Programmierer mit bringen muss, um ein Programm zu entwickeln, unterscheiden sich stark von denen die nötig sind, um ein gutes Design zu kreieren. Mitchell Kapor nennt dies den

> *"Software Design Viewpoint"*

4.1 Die Architektur-Software-Analogie

Um eine Gebäude zu konstruieren, arbeiten Architekten und Ingenieure eng zusammen. Im eigentlich Prozess des Bauens nehmen die Ingenieure Anweisungen von den Architekten entgegen. Beide spielen eine wichtige Rolle, allerdings sind die Ingenieure den Architekten unter zu ordnen. Der Grund dafür ist, dass den Architekten die Verantwortung für das

fertige Konstrukt obliegt. Architekten planen und kontrollieren den gesamten Entwicklungsprozess. Ingenieure dagegen sind lediglich für die Realisierung spezieller Teilbereiche zuständig.
Dies lässt sich auch auf den Entwicklungsprozess von Software übertragen. Der Software-Designer sollte derjenige sein, der die Verantwortung für das Konzept und die Realisation eines Programmes trägt. Ebenso sind Werte der architektonischen Wissenschaft auf die Software-Entwicklung zu übertragen.

> *"The Roman architecture critic Vitruvius advances the notion that well designed buildings were those which exhibit firmness, commodity and delight"*

Ein Programm muss im Gebrauch stabil gegenüber Programmierfehlern reagieren, wirksam in seiner Funktion sein und es sollte dem Anwender eine Freude sein, damit zu arbeiten. Asus wirbt sogar mit diesen Fähigkeiten für seine Mainboards.

Software und ihr Design

> *"It's where you stand with a foot in two worlds - the world of technology and the world of people and human purposes - and you try to bring the two together"*

4.2 Software-Design heute

> *"Today the software designer leads a guerilla existence, formally unrecognized and often unappreciated"*

Software-Design wird von vielen Computerwissenschaftlern als ein spezieller Teilbereich der Computerwissenschaften selber angesehen. Dies vermittelt nach außen den Eindruck, dass jeder Programmierer auch ein Software-Designer ist. Überträgt man dies auf die Architektur, so könnte jeder Ingenieur das Design eines Gebäudes sein Eigen nennen. Deswegen stellt Mitchell Kapor die Forderung auf, Software Design als eine eigene Wissenschaft zu betrachten: Eine Brücke zwischen Computerwissenschaften und der Software-Entwicklung. Die Hauptursache einer benutzerunfreundlichen Software ist, dass Programme eher technisch entwickelt werden anstatt designt. Widersprüchlich zu dem Faktum, dass immerhin ca. 75% des Programmkodes lediglich dem Zweck einer Benutzeroberfläche dienen.

4.3 Mehr als nur Oberflächendesign

> *"Software design is not the same as user interface design"*

Ein häufig auftretendes Phänomen in der Welt der Programmierung ist, dass Benutzeroberflächen erst nach der Entwicklung des Programmes geschrieben werden. Diese explizite Trennung bewirkt, dass Software-Designer im Vergleich zu Programmentwicklern

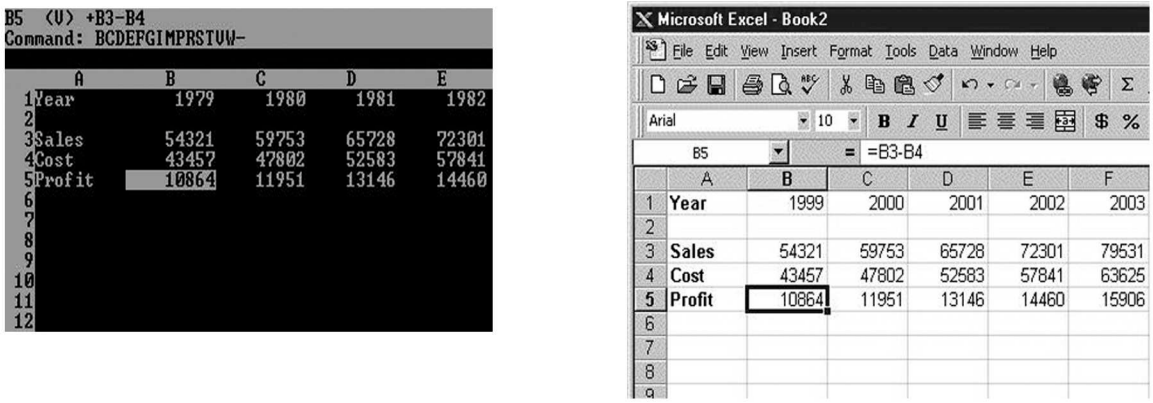

Figure 4.1: Ähnlichkeit zw. VisiCalc & Excel

schnell zu Programmierern zweiter Klasse degradiert werden. Der Software-Designer sollte dagegen viel mehr am eigentlichen Entwicklungsprozess des Programmes teilhaben. Ein krönendes Beispiel dessen, was Software-Designer erreichen können, ist das Tabellenkalkulationsprogramm VisiCalc von Dan Bricklin. Es war seinerzeit das erste für Personal-Computer entwickelte Programm, mit dessen Hilfe man kaufmännische Berechnungen ohne jegliche Programmierkenntisse durchführen konnte. Zudem bildete es die Grundlage für viele heute im Alltag benutzen Programme: Lotus 1-2-3, OpenOffice.org calc oder Microsoft Excel uvm. (siehe Figure 4.1).

4.3.1 Analyse der Analogie bzgl. des Designs

Es existieren durchaus detaillierte Ähnlichkeiten zwischen Software-Design und Architektur. Eine von Mitchell Kapor vorgeschlagene, getrennte Sichtweise zwischen Architekten und Ingenieuren ist weder in der Architektur, noch im übertragenen Sinne bei der Software-Entwicklung immer möglich. In der Architektur gibt es beispielsweise viele Heimwerker. Sie konstruieren eigene Gebilde ohne vollständige Ausbildung. Ebenso gibt es *Heimwerker* im Bereich der Software: Hobbyprogrammierer modifizieren vorhandenen Programmkode, entwickeln eigenständige Fragmente und stellen diese der öffentlichkeit Verwendung per Internet zur Verfügung.

4.4 Die Ausbildung zum Software-Designer

Unter einem guten Programmierer versteht man heute eine Person, die fähig ist effiziente Algorithmen zu schreiben, ein Programm logisch strukturiert auf zu bauen, eine beschreibende Dokumentation zu erstellen, in Teams zu arbeiten und mit dem aktuellen Stand der Technik mit zu halten. Die Fähigkeit, eine gute Software-Schnittstelle für den Benutzer zu erstellen, hat in der Ausbildung zu einem Programmierer nicht ausreichend Platz. Daher bezieht sich Mitchell Kapor erneut auf die Analogie zur Architektur. Dort lernt der Studierende zunächst mit den grundlegenden Mitteln, wie dem Freihandzeichen, das entstehende Gebilde zu modellieren bevor die eigentliche Konstruktion beginnt. Mit Hilfe der Modellierung lässt sich schnell feststellen, ob es sich um eine Fehlkonstruktionen handelt. In der Architektur ist dies lebenswichtig. Beispielsweise müssen Brücken auf Windtauglichkeit getestet werden. Bezüglich des Software-Designs nimmt die Technik des Modellierens ebenfalls eine wichtige Stellung ein. Programmentwürfe können dadurch schneller getestet werden. Dies verkürzt die Entwicklungszeit bis zur finalen Version und spart den entwickelnden Programmierern bzw. der Firma eine Menge Geld. 1987 en-

twickelte Bill Atkinson für die Firma Apple erstmals ein Programm, das diese Technik durch die Skriptingsprache Hypertalk zur Verfügung stellte: Hypercard. Mit Hypercard wurde es Anwendern sehr leicht gemacht, Querverweise aufzubauen und Datenbanken zu erstellen. Damit entwickelte Programme waren häufig Lernsoftware. Aber auch Spiele waren möglich - bekanntes Beispiel: Myst. Die Entwicklung von Hypercard wurde jedoch eingestellt. Dank der objektorientierten Programmierung gibt es sehr mächtige Alternativen. Darunter Runtime Revolution oder Macromedia Director, welches sogar fähig ist, Echtzeit 3D zu integrieren.

4.4.1 Das "Trainingslager" Software-Design

"Students learn software design by practising it"

Ein wichtiger Bestandteil des Ausbildung wäre, Studenten die Möglichkeit zu geben, Designs zu aktuellen Programmen zu entwickeln. Angehende Software-Designer sollten ebenfalls in der Lage sein zu forschen, Problemstellung der Software durch deren Gebrauch im realen Leben zu erkennen und darauf entsprechend zu reagieren. Um richtig reagieren zu können ist es essentiell, die existierenden Medien (z.B. Zeitung oder Fernsehen), die sich in der Gesellschaft durchgesetzt haben intensiv zu betrachten, die Gründe für ihre dauerhafte Existenz zu erkennen und die Merkmale auf Programme zu übertragen.

4.4.2 Design und ihr Entwicklungsprozess

Die Integration der Software-Designer in den Entwicklungsprozess des Programmes, ist von hoher Bedeutung. Die Zusammenarbeit zwischen Designern und Programmierern muss bereits während der Ausbildung zum Software-Designer erlernt werden. Denn während des Entwicklungsprozesses eines Programmes treten neue, ausschlaggebende Erkenntnisse auf, die sowohl für den Entwickler als auch für den Designer wichtig sind und zu Veränderungen der zugrunde liegenden Programmstruktur führen können.

4.4.3 Analyse der Analogie bzgl. der Ausbildung

Die Ausbildung eines Architekten unterscheidet sich ebenfalls von der eines Software-Designers. Studenten der Architektur sind umgeben von Meisterwerken und Erfolgen früherer Generationen. Sie besitzen ebenfalls Informationen über die größten Fehler. Etwas, das auf der Wichtigkeitsskala des Wissens mindestens gleichwertig einzuordnen ist. Ob die Lernmethode des Betrachtens bereits *kreierter Meisterwerke* ebenso effektiv auf die Entwicklung von Software zu übertragen ist, kann man noch nicht genau sagen. Einerseits ist die Wissenschaft des Software-Designs noch sehr jung, andererseits sind die Entwicklungszyklen von Software viel kürzer als die Entwicklungszyklen von Architektur. Sowohl die Ergebnisse der Architektur, als auch die der Softwareentwicklung sind öffentlich und jedermann zugänglich. Dementsprechend entwickeln sich beiderseitig neue Stile aus dem bisher Gewesenen: Bei dem Design des ersten Microsoft Windows OS wurde versucht, das Oberflächendesign des Apple-Rechners so gut wie möglich zu übernehmen.

4.5 Eine stabile Grundlage in der Technologie

Es ist wichtig, dass Software Designer ebenfalls eine intensive, grundlegende Ausbildung in der Architektur von Computersystemen, Mikroprozessoren, Betriebssystemen, Netzwerkkommunikation, Datenstrukturen, Algorithmen, verteilten Systemen, Programmierumgebungen und objektorientierter Entwicklung erhalten. Nur auf diese Weise ist es dem Software-Designer möglich, sinnvolle Programmvorschläge zu machen und mit den Programmierern zu kommunizieren. Ansonsten wäre es gerechtfertigt, Designer als zweitklassige Programmierer zu bezeichnen.

4.6 Ein Aufruf zur Handlung

Sowohl für die Softwareindustrie als auch für den Endanwender kann es nur von Vorteil sein, einen neuen Berufsstatus zu kreieren, dessen Hauptziel das Software-Design ist. Es ist unabdinglich, dass Software-Designer ihre eigene Gemeinschaft brauchen, um den Entwicklungsprozess voran zu treiben. Software-Designer müssen in Entwicklungsteams eng mit den Softwareentwicklern zusammen arbeiten, um eine erfolgreiche und benutzerfreundliche Software zu erstellen. Das Konzept der Zusammenarbeit zwischen Architekten und Ingenieuren hat sich in der Architektur seit langer Zeit bewährt. In der Computerwissenschaft kommt man um die Kooperation von Designern und Programmierern auch nicht herum. Es ist lediglich noch nicht anerkannt. Dies ist die Kernaussage, die Mitchell Kapor in seinem Manifest vermitteln wollte. Ein Jahr nach Veröffentlichung konnte man starke Reaktionen auf sein Manifest beobachten, insbesondere im Silicon Valley, das als Motor der globalen High-Tech-Ökonomie gilt. Dort ansässige Firmen sind u.a: Apple Computer, Ebay und Intel. Aber auch berühmte Universitäten reagierten. In Stanford wurde unter anderem eine große nationale Wissenschaftsgrundlage geschaffen. Dort wird weiter an Software-Design geforscht und dieses ebenfalls unterrichtet. Mitchell Kapor hat es also geschafft, die Betrachtungsweise von Software zu ändern und ihre Entwicklungsschwerpunkte hinsichtlich des Designs zu verlagern.

4.6.1 Über die Analogie hinaus

Zusammengefasst kann man sagen, dass sich sehr viele vergleichbare Ansätze in der Architektur-Software-Analogie finden lassen. Jeder von ihnen bietet die Möglichkeit einer ausführlichen Diskussion. Das Ziel ist jedoch zu erkennen, welche Innovationen die Architektur im Laufe der Geschichte hervorgebracht hat und diese Konzepte auf die Software zu übertragen. Der Schlüssel zum Erfolg liegt also nicht nur darin, Bisheriges zu verstehen sondern insbesondere darin, die richtigen Fragen zu stellen. Möglicherweise existieren noch passendere Analogien, als die der Software-Architektur-Analogie, deren Innovationen zu einem tieferen Verständnis der Benutzerfreundlichkeit führen. Möglicherweise müssen aber auch erst die Anwender und deren Beziehung zu Software reifen, bevor eine neue, modernere Analogie fähig ist, die Software-Architektur-Analogie zu ersetzen.

4.7 Über Mitchell Kapor, Autor des Manifestes

Mitchell Kapor ist Gründer der Lotus Development Corporation, heute zugehörig zu IBM. Er schuf 1983 die Tabellenkalkulationssoftware Lotus 1-2-3. Innerhalb von zwei Jahren konnte damit ein Gewinn von über 200 Millionen US-Dollar erzielt werden. Des Weiteren ist Kapor Mitbegründer der Mozilla Foundation und dessen Vorsitzender. Diese noch recht junge Softwarefirma schaffte es trotz kostenloser Anbietung ihrer Produkte immerhin schon, Gewinne in zweistelliger Millionenhöhe zu erzielen.
Zur Zeit arbeitet Kapor an einem Produkt, das den Namen *Chandler* trägt und an dem Tage seiner Veröffentlichung gegen Microsoft Outlook antreten soll. Diese gigantischen Erfolge unterstreichen zusätzlich die Aussagekraft seines Manifestes, denn offensichtlich fällt die Resonanz der Endanwender auf seine Arbeit positiv aus.

Figure 4.2: Mitchell Kapor

5 Aktuelle Fakten

Seit der Veröffentlichung des Manifestes hat sich in der Computerwelt viel getan. Ein möglicher Studiengang, der die von Mitchell Kapor angesprochenen Fakten größtenteils beinhaltet, ist Software Systems Engineering (M.Sc.). Dieser wird unter anderem auch an der RWTH Aachen angeboten und beinhaltet beispielsweise folgende Themen: Modellierung von Software-Architekturen, Designing Interactive Systems oder Current Topics in Media Computing and Human Computer Interaction. Des Weiteren bietet die RWTH als Nebenfach zur Informatik Psychologie an. Dort werden speziell für Informatiker angepasste Vorlesungen angeboten, die sich mit der Benutzerfreundlichkeit beschäftigen: Software-Ergonomie.
Spezieller gehen wir hier auf den "Master of Science" Studiengang Human-Computer-Interaction (siehe Figure 5.1) am "Georgia Institute of Technologie" ein. Dieser ist dort dank des "Collage of Computing", der "School of Psychologie" und der "School of Literature, Communication and Culture" seit 1997 möglich. Die Studiendauer beträgt ca. vier Semester und beinhaltet 36 Semesterwochenstunden. Zunächst müssen drei Grundlagenfächer absolviert werden. Abhängig von der akademischen Ausbildung des Studenten steht eine variable Menge an Vertiefungsfächer zum Angebot. Anschließend folgt eine Spezialisierung. Zu guter Letzt wird das Gelernte an einem konkreten Projekt realisiert. Eine Berufsbezeichnung für Designer ist inzwischen Realität. Der offizielle Titel dafür lautet: *Software Design Engineer*

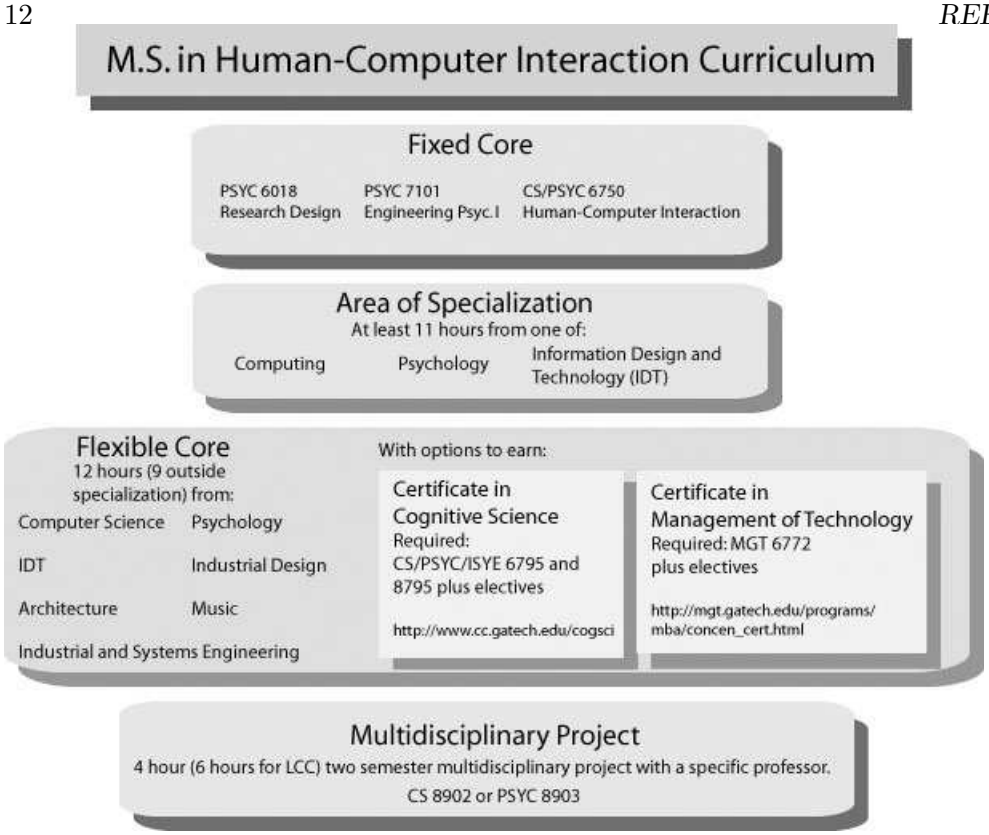

Figure 5.1: Georgia Institute of Technologie

6 Zusammenfassung

Durch die Definitionen von Software und Design erhalten wir einen Eindruck davon, was beim Entwurf einer Software zu beachten ist und welche Faktoren eine wichtige Rolle spielen. Software, unterstützt durch ein gutes Design, nimmt die Funktion einer Schnittstelle zwischen Technik und Benutzer ein. Ausgehend von verschiedenen Blickwinkeln, sind die Stärken von Software-Design unterschiedlich zu definieren. Es wird erklärt, was Informatiker oder Leute aus dem Bereich HCI beim Softwareentwurf zu beachten haben. Zur Verdeutlichung dient dabei die Architektur-Software-Analogie. Es gilt den Berufsstatus des Software-Designers öffentlich anzuerkennen, der im übertragenen Sinne mit dem des Architekten vergleichbar ist. Um diesen zu erhalten ist es zwingend nötig, dass der angehende Software-Designer eine intensive theoretische und praktische Ausbildung erhält. Seit dem Aufruf von Mitchell Kapor hat sich vieles verändert. Inzwischen gibt es diesen Berufsstatus: *Software Design Engineer*

References

[1] Terry Winograd. *Bringing Design to Software.* Addison-Wesley, ACM Member Services, 151 Broadway, 17th Floor, New York, August 2001.