

Benutzeroberflächen – ein Stiefkind der Software-Industrie?

Ein Überblick über die Entwicklungsgeschichte von Benutzeroberflächen für Anwendungssoftware und ihren Methodenbedarf.

Methoden der Softwareentwicklung

Seit den Anfängen der Softwareentwicklung gibt es gute und schlechte Softwareanwendungen und damit gute und schlechte Benutzeroberflächen.

Die Softwarekrise Ende der 60er Jahre hat den Entstehungsprozess von Anwendungssoftware entmystifiziert und die Entwicklung von ingenieurmäßigen Methoden und Vorgehensmodellen für den Softwarebau eingeleitet. Man begann, analog zur Bau- und Automobilindustrie, Software systematisch zu entwickeln.

Die so begonnene kontinuierliche Auseinandersetzung mit den Schwächen der Methoden und Werkzeuge der Softwareentwicklung hat zu einer Vielzahl nützlicher Hilfsmittel geführt, z.B. SADT (structured analysis and design technique), ERM (entity relationship model), OOD (object oriented development), UML (unified modelling language) oder XML (extensible markup language).

Die Verbreitung und die fortschreitende Standardisierung dieser Mittel hilft dabei, Kommunikationsbrüche in Prozessen der Softwareentwicklung zu vermeiden und zu schließen.

Bemerkenswerterweise werden Benutzeroberflächen bis heute nur vereinzelt (XForms, XUL, UIML) und ohne praktische Flächendeckung (es herrschen domänenspezifische Formate und Generatoren vor) in diese Methodenentwicklung mit eingeschlossen. Meist werden Oberflächen mit wenigen Stilvorgaben oder einem UI-Styleguide dem Gestaltungsspielraum des Entwicklers überantwortet.

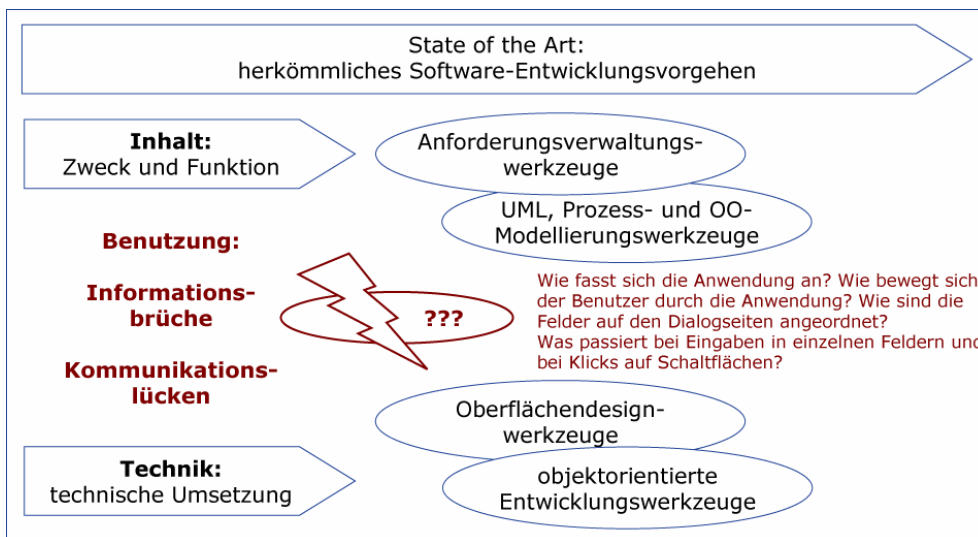


Abbildung 1: Informationsbrüche in der Oberflächenentwicklung

Diese Nichtbeachtung von Oberflächenmethoden ist deshalb merkwürdig, weil die Anwendervertreter und Auftraggeber von Software ihre Anforderungen und Anwendungsfälle anhand von Masken und Dialogabläufen ausdrücken und daher eindeutige Formulierungsmittel hier in beiderseitigem Interesse wären.

Hierzu lässt sich die nachfolgende Erklärungstheorie aufstellen:

Von Lochkarten und Stapelläufen kommen wir ...

In der Batchwelt der frühen EDV kamen bis in die späten 50er Anwendungsoberflächen mit interaktiven Dialogseiten gar nicht vor. Prozedurale Anwendungen erledigten die Verarbeitung von gestapelten Eingabedaten ohne Anwenderinteraktion und lieferten die Ergebnisse meist in Form von Listen oder neuen Datenstapeln.

Mit dem Aufkommen erster dialog-transaktionaler Anwendungen in den frühen 60ern änderte sich daran zunächst wenig, da hinter den Dialogseiten meist die bisherigen Batchaufrufe maskiert wurden. In den Masken wurden die Parameter für den Batchlauf eingetragen und mit der Maskenfreigabe an die Prozedur übergeben. Oberflächen ergaben sich damit „von selbst“ aus den Prozeduren und stellten als Abfallprodukt des Funktionsdesigns keine eigenen Ansprüche an die Entwicklung von Methoden zu ihrer systematischen Entwicklung.

Mit der Zunahme von Dialogtransaktionen wurden Maskenbeschreibungsformalismen entwickelt und in die Programmiersprachen und Entwicklungswerkzeuge integriert, z.B. SDD (structured dialog design) in RPG (report program generator), Input- und Display-Befehle in COBOL (common business oriented language) oder IO-Makros in Assembler auf BS2000-Systemen. Später folgten diesem Ansatz fortgeschrittene Formen von Maskengeneratoren in dBase, FoxPro, in verschiedenen RDBMS (relational database management system) und anderen integrierten Werkzeugen für Geschäftsanwendungen.

Die Situation änderte sich mit dem Markteintritt der Computermaus und grafischer Betriebssystemaufsätze wie X-Windows, GEM (graphical environment manager), GEOS dramatisch, auch durch die sich entwickelnde Marktdominanz von Microsoft Windows.

Die transaktional-nach-interaktiv-Verschiebungsfalle

Durch GUIs (graphical user interfaces) wurden Anwendungen zunehmend reaktiv; Transaktionen verflochten sich mit Steuerungs- und Rückkopplungselementen der Oberflächen. Als Antwort darauf wurden die Werkzeugkästen der Maskengeneratoren um grafische Kontrollelemente und Ereignisbehandlung auf diesen typisierten GCIs (graphical control items) erweitert.

Die Entwickler begannen, Transaktionen an einzelne Controls zu koppeln. Die Transaktionen wurden feingranularer, um die neuen Kontrollelemente beliefern zu können. Statt der Eingabe einer Artikelgruppe in einem Schlüsselfeld benötigte man jetzt z.B. eine Combobox mit Auswahl der Artikelgruppe und somit die bisher nicht benötigte Transaktion „hole die Namen aller Artikelgruppen“.

Man schmiedete weitere Controls und koppelte ihre Interaktionen immer dichter an den Datenhaushalt der Anwendungen. Dabei umging man die hinderliche Transaktionsschicht - und landete überrascht beim Fat Client.

Aus Tradition wurde dabei die Entwicklung von Benutzeroberflächen weiter als selbstverständliches Abfallprodukt der Entwicklung von Funktionalitäten angesehen. Auch in der GUI-Welt nahm man die Komplexität der dort auftretenden Wechselwirkungen von Arbeitsabläufen, Darstellung, Eingaben und Transaktionen nicht als einen eigenen, zur Funktionsentwicklung parallelen Entwicklungsbereich wahr, sondern schob die Oberflächen weiterhin ans Ende des Entwicklungsprozesses.

Oberflächen, so argumentieren Entwickler oft, kann man eigentlich erst am Schluss implementieren, sonst müssen sie ständig geändert werden. Oberflächen, so diese Argumentation weiter, sind ja nur ein Aufsatz auf die wesent-

lich komplexere Funktionalität und daher ohne weiteres über die Funktionalität zu stützen.

State-Charts: Ein Aufbruch mit anschließender Stagnation

Ein Konsens über den Modellierungsbedarf im Bereich der Anwendungsoberflächen kristallisierte sich nur langsam heraus. Die von David Harel 1984-1985 und 1987 vorgeschlagene State-Charts-Notation trug den Verschiebungen von prozeduralen zu reaktiven Systemen Rechnung und lieferte erstmals eine standardisierte Beschreibungsform für Teile von Oberflächen.

Die State-Charts decken die Ablauf- und Strukturaspekte einer Oberfläche ab, nicht aber die Darstellungs- und Notifikationssicht und bieten auch keine explizite Schnittstellensicht zum funktionalen Teil der Anwendung.

Bemerkenswert ist, dass die State-Charts nicht in Konkurrenz mit anderen Formalismen für die Oberflächenentwicklung weiter spezialisiert und weiter entwickelt wurden, sondern seit nunmehr 20 Jahren in ihrer Ursprungsform stabil geblieben sind. Diese Stagnation steht ganz im Gegensatz zur Daten- und Funktionsmodellierung und der „normalen“ Ablaufmodellierung, wo sich die Konvergenz und Erweiterung der Formen und Methoden aus ihrem Wettbewerb heraus entwickelt.

Das Web bricht Grenzen zwischen Anwendungen auf

Die Anwendungsentwickler hatten sich also wieder arrangiert und im Fat Client Oberflächen „irgendwie“ auf Funktionen und Abläufe aufgesetzt, als sich - auch diesmal zunächst unbemerkt - 1991 durch die Markteinführung des WWW (world wide web) und des HTTP (hyper text transfer protocol) die Situation abermals dramatisch änderte.

Zunächst diente das HTTP nur der standardisierten Darstellung wissenschaftlicher Texte. Die Möglichkeit, Anwendungen anzusteuern und damit dynamische Inhalte darzustellen kam etwas später dazu, zunächst als CGI (common gateway interface).

Dramatisch am Web-Browser in Bezug auf die auf dem Fat Client gewöhnlich verwendeten Anwendungsoberflächen ist, dass nach dem HTTP-Konzept die Grenze zwischen Transaktionen und Präsentation hart, weil physisch, ist. Die im Fat Client übliche Intensivkopplung zwischen einzelnen Controls und „Minitransaktionen“ sind hier teuer, weil für jeden Kopplungszyklus eine Servertransaktion mit Datenübertragung notwendig ist.

Thin Web Clients sind gewissermaßen die Wiedereinführung der bürokratischen Transaktionswelt, in der man den Zentralrechner hinter der Bildschirmmaske wieder erahnen kann. Und das, nachdem sich Anwender an die beliebig interaktive Fat Client Welt gewöhnt hatten, in der man ruhig das Gefühl haben konnte, dass man die Anwendung mit allen Daten zur alleinigen Nutzung auf seinem Rechner hat.

Natürlich schränken Thin Clients die unbegrenzten UI-zu-Transaktion Kopplungsmöglichkeiten der Fat Clients ein. Und natürlich möchten die Anwender nicht eingeschränkt werden.

Das Web hat die Grenze zwischen der Präsentationsschicht und der Transaktionsschicht wieder sichtbar gemacht. Dadurch ist auch sichtbar geworden, wie viel von der Anwendung eigentlich Präsentationsschicht ist, welche komplexen IO- und Interpretationsvorgänge hier enthalten sind - und dass sie geeigneter Beschreibungsformen bedürfen. Die Welt der (zentralen und funktionsorientierten) Transaktionen und die Welt der (lokalen und anwenderorientierten) Interaktionen mit ihren jeweiligen Vorteilen und Wechselwirkungen sind durch Web-Oberflächen in ein produktives Spannungsfeld getreten.

Die Softwarekrise geht weiter

Die Frage „Wie beschreibt man eigentlich eine Benutzeroberfläche?“ hat im Kontext der immer komplexer werdenden Anwendungen und der technischen Möglichkeiten des Web eine neue Dringlichkeit erhalten. Das Web durchbricht Grenzen zwischen Einzelanwendungen und ermöglicht die Fokussierung auf Prozesse und Workflows. Abermals stößt die Informatik dabei darauf, dass sie kein Mittel zur Beschreibung der Antwort auf **„Was spielt sich an der Anwendungsoberfläche ab und wie hängt das mit der transaktionalen Funktionalität darunter zusammen?“** parat hat und dies nachholen muss, um Anwendungen weiter global miteinander vernetzen zu können.

Die Verdrängung des Problems ist verständlich, denn Oberflächen sind von den ursprünglich kargen Kommandokonsolen zu wahren Eisbergen über der eigentlichen Funktionalität gewachsen. Während Wassereisberge zu $\frac{2}{3}$ unterhalb der (Wasser-)oberfläche sind, ist das Verhältnis von User Interfaces zur transaktionalen Funktionalität einer Anwendung eher umgekehrt.

Bei einer dialogintensiven Anwendung sind oft $\frac{2}{3}$ und mehr Interaktion bzw. Kommunikation zwischen Anwender und Anwendungsoberfläche. Die tatsächliche funktionale Transaktion kommt hingegen erst am Schluss der „Verhandlung“ zwischen Anwender und UI zur Ausführung. Der Anteil der Funktionalität am Eisberg ist kleiner als wir bislang aus der funktionalen Sicht angenommen haben. Es lohnt sich also, Formulierungsmittel und Methoden zur Spezifikation von Oberflächen zu entwickeln.

Es ist durchaus möglich, dass die Notwendigkeit einer FOS (formale Oberflächenspezifikation) auch diesmal von den traditionellen Anachronismen wie „Oberflächen sind Peanuts“ oder „Form Follows Function“ verdrängt wird. Man arbeitet mit .Net und Active-X, DHTML und JavaScript zwar in verschiedenen Lagern, doch zieht man dabei durchaus am gleichen Strang, nämlich wieder Fat Client Verhältnisse einzuführen. Die Bemühungen gehen dahin, das aus der Fat Client Welt gewohnte Interaktionslevel in die Web Client Welt zu übertragen. Dies ist zunächst an sich kein systematisches Problem, sofern das Netz genügend Bandbreite zur Verfügung stellt.

Wird jedoch beispielsweise durch .Net die Grenze zwischen Transaktion und Präsentation wieder weich, ohne dass geeignete Formen der Oberflächenspezifikationen zur Verfügung stehen, ist eine komplexitätsbedingte Softwarekrise eine mögliche Folge.

Das verursachende systematische Problem der heutigen Softwareentwicklung in Bezug auf Oberflächen könnte dabei in eine nicht mehr beherrschbare, weil nicht exakt formulierbaren Vermengung von Präsentations- und Anwendungslogik münden.

Eine Messbarkeit, Exaktheit und Wiederverwendbarkeit von Oberflächen ist ohne FOS (formale Oberflächenspezifikation) nach Ansicht des Autors nicht möglich.

Von der freien Kunst zum systematischen Mehrwert

Die freischaffende Kunst des Oberflächenbaus als Anhängsel der objektorientierten Funktionalitätsentwicklung ist aus Sicht der heutigen Herausforderungen und Kosten beim Bau von Anwendungsoberflächen bei globalen Applikationen nicht beliebig fortführbar.

Eine systematische Herangehensweise an das Thema formale Spezifikation von Anwendungsoberflächen heißt, der Reihe nach die folgenden Fragen zu beantworten:

- Warum braucht man formale Oberflächenspezifikationen (FOS)?
- Welches Problem lösen sie, das von anderen Spezifikationsformen nicht gelöst wird?
- Welchen Vorteil bieten sie, den andere Spezifikationsformen nicht bieten?

- Was genau ist eine FOS? Welche Merkmale hat sie? Was ist ihr informativ-er Inhalt?
- Was macht man mit einer FOS, wenn man sie hat?
- Wie bewertet man die Vollständigkeit und die Qualität einer FOS?
- Wie schreibt man eine FOS?
- Wie beginnt man mit der Erstellung einer FOS? Wie fährt man fort?
- Wie weiß man, in wie weit man die Spezifikation fertig gestellt hat?
- Wie und mit wem stimmt man die geschriebenen Inhalte der FOS ab?
- Welche Hilfsmittel für das Erstellen und Auswerten einer FOS genügen welchen Zwecken?

Die oben skizzierten Fragestellungen decken die Motivation für formale Oberflächenspezifikationen, ihre inhaltliche Bestimmung, ihre Verwendung und Erstellung, sowie die hierfür einsetzbaren Methoden und Werkzeuge ab.

Die in den Whitepapers des HCI-Projekts www.benutzeroberflaeche.de vorgeschlagenen Antworten auf die oben eröffneten Fragen zeigen Beispiele für Strukturen, Formalismen, Vorgehensweisen und Unterstützungsmittel für formale Oberflächenspezifikation. Die vorgestellten Methoden, Verfahren und Werkzeuge beruhen auf praktischen Projekterfahrungen und auf Ergebnissen der eigenen HCI-Forschung.

pch, 07/2004
last revised 01/2006

Paul Chlebek
HCI • GUI • UML • OOD • MDA • XSLT

mailto: pc@benutzeroberflaeche.de
phone: +49 173 106 1830

<http://www.projectpeople.net/chlebek/>
<http://www.benutzeroberflaeche.de>

Links:

Softwarekrise

<http://de.wikipedia.org/wiki/Softwarekrise>

SADT

<http://www.fh-augsburg.de/informatik/vorlesungen/se1t/script/definition/sadt.html>

<http://www.sadt.info/> (save a dog today, nicht über SADT aber ok)

EDV-Geschichte

http://www.stefan-lenz.ch/glossareintrag_anzeigen.php?file=geschichte.htm

http://www.edgar-elsen.de/Historisches/Rechentechnik_und_DV_in_Jahreszahlen.htm

State-Charts

<http://www.wisdom.weizmann.ac.il/~dharel/SCANNED.PAPERS/Statecharts.pdf>

Web Erfindung

<http://www.sueddeutsche.de/computer/artikel/556/2554/>