

Kapitel 2: Oberfläche

... in dem die Struktur und die Bestandteile einer Benutzeroberfläche unter die Lupe genommen werden und eine allgemeine Gliederung für User Interfaces aufgestellt wird.

Kapitelziele

- **Wissen, was eine Softwareoberfläche ist.**
- **Wissen, woraus eine Softwareoberfläche besteht.**
- **Gliederung der Bestandteile einer Oberfläche.**

Umgangssprachlich werden die vom Anwender wahrnehmbaren Teile der Softwareanwendung mit vielen synonymen Bezeichnungen bedacht. Üblich sind Bezeichnungen wie Benutzerschnittstelle, Bedieneroberfläche, Anwendungsoberfläche, Softwareoberfläche, Dialogmasken oder Dialogschnittstelle. In Spezialistenkreisen werden weiter differenzierte Fachbezeichnungen verwendet: Präsentationsschicht, UI (User Interface), GUI (Graphical User Interface), Human Computer Interface (HCI) oder Mensch-Maschine-Schnittstelle (MMS).

In diesem Kapitel machen wir das, was Ingenieure am liebsten tun: Wir nehmen uns einen Schraubenzieher, schrauben den Deckel ab und gucken gründlich und tief in die Zahnräder eines User Interfaces hinein.

Eine Softwareoberfläche ist kein Monolith und auch keine Verpackung um die Softwarefunktionen, sondern eine Maschine mit verschiedenen Teilen, die gemeinsam eine komplexe Vermittlungsaufgabe erfüllen.



Bild 14: Eine Softwareoberfläche besteht aus verschiedenen Teilen

Eine **Softwareoberfläche** ist eine **komplexe** Maschine, die zwischen Anwender und Softwarefunktionen **vermittelt**.

Üblich ist bei heutigen Oberflächen auch, dass sie den Anwender anleiten. Dazu gehören entsprechender Aufbau der Dialoge selbst, aber auch kontextsensitive Hilfefunktionen und On-Screen-Hilfe. *Benutzerdokumentation* kann aus diesem Blickwinkel als Teil der Softwareoberfläche gesehen werden.

Dieses Kapitel zeigt, welche verschiedenen Bestandteile eine Oberfläche hat und wie diese gegliedert werden können. Die Bestandaufnahme liefert einen Überblick über Zusammenhänge und Unterschiede zwischen den verschiedenen Teilen einer Softwareoberfläche.

Das Ergebnis dieser *informationellen Analyse* eines User Interface ist ein *Gliederungsgerüst* für das Beschreiben der Benutzerschnittstellen von Softwareanwendungen.

Zu dieser Grundgliederung gehören Abläufe und Inhalte sowie Wechselwirkungen und Beziehungen zwischen den Oberflächenbestandteilen, Anwendungsfunktionen und Anforderungen.

UI-Perspektiven

Auf diese Strukturierungsgrundlage, die ich *UI-Perspektiven* nenne, wird die User Interface-Architektur in allen Kapiteln zurückgreifen und aufbauen. Sie können lernen, mit diesen Perspektiven in unterschiedlichen Phasen des Softwareprojekts umzugehen und so verschiedene Aspekte der Oberfläche zu sehen und zu entwerfen.

Eine UI-Gliederung nach UI-Perspektiven kann zum Beispiel zum

- Vergleichen des Informationsgehalts von verschiedenen umgesetzten Oberflächen oder zum
- Bewerten der Vollständigkeit einer zur Prüfung vorgelegten Oberflächenbeschreibung

herangezogen werden.

Sie können die UI-Perspektiven auch verwenden, um

- oberflächenrelevante Informationen in einen Gesamtzusammenhang einzuordnen und die
- UI-Beschreibung schrittweise von der ersten Skizze bis zu einer umsetzungsreifen Detailspezifikation der Softwareoberfläche zu ergänzen.

2.1 Historische Entwicklung der Softwareoberflächen

Seit den Anfängen der Softwareentwicklung gibt es gute und schlechte Softwareanwendungen und damit gute und schlechte Benutzeroberflächen. Ende der 60er Jahre hat die so genannte **Softwarekrise** die Entwicklung von ingenieurmäßigen Methoden und Vorgehensmodellen für den Softwarebau eingeleitet. Die Softwareindustrie begann, analog zur Bau- und Automobilindustrie, Software systematisch zu planen und zu entwickeln.

Die Softwarekrise war der Motor für systematisches Entwickeln von Software

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),**
- **IDEF (Integrated Computer Aided Manufacturing Definition Languages),**
- **ERM (Entity Relationship Model),**
- **OOD (Object Oriented Design),**
- **UML (Unified Modelling Language) oder**
- **XML (Extensible Markup Language).**

Die Verbreitung und die fortschreitende Standardisierung dieser Hilfsmittel und Methoden helfen, Kommunikationsbrüche in Prozessen der Softwareentwicklung zu vermeiden und zu schließen.

Benutzeroberflächen sind bis heute nur nachrangig (XForms, XUL, UIML), in diese Methodenentwicklung eingeschlossen.

Von Stapelverarbeitung zu Dialogtransaktionen

In der Batchwelt der frühen EDV kamen bis in die späten 50er Jahre 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 Datenkartenstapeln. Tastatur und Bildschirm wurden 1960 zum ersten Mal an einen Computer angeschlossen und lösten unter den Großrechnerherstellern Panik aus [Rose 1985, S.48].

In prozeduralen Anwendungen gibt es keinen Dialog mit dem Anwender

Mit dem Aufkommen der ersten dialogisierten transaktionalen Anwendungen in den frühen 60ern änderte sich an der Bedienung der Softwareprogramme zunächst wenig, da hinter den Dialogseiten meist die bisherigen Batchaufrufe maskiert wurden. In den Masken wurden die Parameter für den Stapellauf eingetragen und mit der Maskenfreigabe an die Prozedur übergeben. Oberflächen ergaben sich damit „von selbst“ aus den Prozedu-

Bei dialog-transaktionalen Oberflächen wird der Anwender am Fluss der Transaktionen entlang geführt

ren. Als Abfallprodukt des Funktionsdesigns stellte die Anwendungsoberfläche keine eigenen Ansprüche an die Entwicklung von Methoden zu ihrer systematischen Entwicklung.

Mit der Zunahme von Dialogtransaktionen wurden Maskenbeschreibungssprachen entwickelt und in die Programmiersprachen und Entwicklungswerkzeuge integriert, z.B. **SDD (Structured Dialog Design)** in IBMs **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)** wie **Oracle** oder **DB/2** und in anderen integrierten Werkzeugen zur Entwicklung von datenorientierten Geschäftsanwendungen.

Grafische Benutzeroberflächen

Softwareoberflächen veränderten sich stark mit dem Markteintritt der Computermaus und mit dem Erfolg grafischer Betriebssysteme (bzw. Betriebssystemaufsätze) wie

- **Mac OS**,
- **X-Windows**,
- **GEM (Graphical Environment Manager)**,
- **GEOS (Graphic Environment Operating System)**,
- **MS Windows (Microsofts Antwort auf GEM)** oder
- **KDE (K Desktop Environment)**.

Grafische User Interfaces arbeiten ereignisgesteuert

Durch **GUIs (Graphical User Interfaces)** wurden Anwendungen zunehmend reaktiv. Die wesentliche Neuerung der grafischen Oberflächen zu bisherigen Dialogbildschirmen ist die **ereignisorientierte Arbeitsweise**.

In transaktionsorientierten Dialoganwendungen haben Oberflächen an bestimmten Stellen im Programmablauf bestimmte Eingaben des Benutzers gefordert. Mit der Einführung der Ereignisorientierung wurde diese strikte Führung des Benutzers gelockert. Der Benutzer kann die angebotenen Bedienungselemente in beliebiger Reihenfolge verwenden. Die Anwendungsoberfläche reagiert auf diese Verwendungsereignisse. In gleicher Weise kann die Anwendungsoberfläche auf anwendungsinterne Ereignisse reagieren. Eine kompakte Beschreibung der ereignisgesteuerten Programmierung finden Sie z.B. in [Lauer 2002, S.22ff].

Bei grafischen Oberflächen verflochten sich Transaktionen mit den für den Benutzer sichtbaren Steuerungs- und Rückkopplungselementen. Auf Entwicklungsseite wurden die Werkzeugkästen der bisherigen Maskengeneratoren um grafische Kontrollelemente (GCI, Graphical Control Items) und um Ereignisbehandlung erweitert. Es wurden grafische SDKs (Software Development Kits) und GCLs (Graphical Control Libraries) entwickelt.

Die Entwickler begannen, *Transaktionen und Transaktions-teile an einzelne Kontrollelemente zu koppeln*. Die Transaktionen wurden feinkörniger, um die neuen Kontrollelemente mit Statusinformationen und Wertebereichen beliefern zu können. Zum Beispiel benötigte man jetzt statt der Eingabe einer Artikelgruppe in einem Schlüsselfeld eine Combobox mit Auswahl der Artikelgruppe und somit die neue Transaktion „hole die Namen aller Artikelgruppen“.

Es wurden immer vielseitigere Kontrollelemente entwickelt und ihre Interaktionen immer dichter an den Datenhaushalt der Anwendungen gekoppelt. Dabei umgingen die Entwickler gerne die hinderliche Transaktionsschicht. Die so entstandenen Anwendungen, deren Hauptmerkmal eine hochinteraktive Benutzeroberfläche mit sehr feinkörnigen Interaktionen ist, werden als *Fat Clients* bezeichnet.

*Enge Kopplung
zwischen
Transaktionen und
Kontrollelementen*

Trotz der hohen Komplexität der in einer grafischen Oberfläche auftretenden Wechselwirkungen von Arbeitsabläufen, Darstellung, Eingaben und Transaktionen blieb die Methodik der UI-Entwicklung zunächst ein Randthema.

Auch bei Fat Client GUIs wird die Entwicklung der Oberflächen oft weiter als Abfallprodukt der Entwicklung von Funktionalitäten angesehen.

Von Fat Clients zu Thin Clients

Die Anwendungsentwickler haben sich erfolgreich mit grafischen Oberflächen und mit der Ereignisorientierung arrangiert. Das Paradigma vom Aufsetzen der Oberfläche als „Masken“ über Funktionen und anwendungsinterne Abläufe wurde beibehalten.

Durch die 1991 begonnene weltweite Verbreitung und kommerzielle Nutzung des WWW (World Wide Web) und damit der HTML (Hyper Text Markup Language) hat sich die Funktions- und Bauweise von Softwareoberflächen - zunächst langsam und weitgehend unbemerkt - abermals dramatisch verändert.

Anfangs diente die HTML zur standardisierten Darstellung wissenschaftlicher Texte. Die Kommunikation zwischen UI (Web Browser) und Anwendungs-Backend (Web Server) erfolgte zu-

nächst nur auf Ebene der Web-Seiten über HTTP (Hyper Text Transfer Protocol). Die Möglichkeit, Anwendungen anzusteuern und damit dynamische Inhalte darzustellen, kam später dazu, zunächst als CGI (Common Gateway Interface). Bei CGI wurde sozusagen die Ereignisverarbeitungsschleife eines Fat Clients als Serverprogramm nachempfunden. Danach folgten viele weitere Möglichkeiten für interaktive Web-Anwendungen, z.B. Applets, Servlets oder Java Server Pages (JSP).

Der große Unterschied zwischen einer auf HTML und Web Browser basierten und einer vom Fat Client gewohnten Anwendungsoberfläche ist, dass nach dem HTTP-Konzept die Grenze zwischen Transaktionen und Präsentation hart, weil physisch, ist. Die im Fat Client üblichen *Intensivkopplungen zwischen einzelnen Controls und den feinkörnigen Transaktionen* sind bei einem Thin Client teuer, weil für jeden Kopplungszyklus eine Servertransaktion mit Datenübertragung notwendig ist.

Thin Web Clients haben Ähnlichkeiten mit dialog-transaktionalen Oberflächen, in der man den Zentralrechner hinter der Bildschirmmaske spüren kann. Für Anwender, die sich an die beliebig interaktive Fat Client-Welt gewöhnt hatten, in der man das Gefühl haben konnte, dass man die Anwendung mit allen Daten zur alleinigen Nutzung auf seinem Rechner hat, kann sich das wie ein Rückschritt anfühlen.

Natürlich schränken Thin Clients die unbegrenzten Kopplungsmöglichkeiten zwischen Teiltransaktionen und Controls der Fat Clients ein. Und natürlich möchten die Anwender nicht eingeschränkt werden.

Das Web zeigt die Grenze zwischen Oberfläche und Funktionalität auf.

Das Web hat die Grenze zwischen der Präsentationsschicht und der Transaktionsschicht wieder spürbar gemacht. Dadurch ist auch sichtbar geworden, wieviel von der Anwendung potentiell Präsentationsschicht ist. Damit ist auch spürbar geworden, welche komplexen IO- und Interpretationsvorgänge in einer Oberfläche enthalten sind.

Das Web bricht Grenzen zwischen Anwendungen auf.

Das Web durchbricht aber auch die Grenzen zwischen Einzelanwendungen und ermöglicht die Fokussierung auf Prozesse und Workflows. Anwendungen werden über das Web global miteinander vernetzt.

Von Thin Clients zur SOA und weiter zur User Interface-orientierten Softwarearchitektur

Die Welten der (zentralen und funktionsorientierten) Transaktionen und der (lokalen und anwenderorientierten) Interaktionen mit ihren jeweiligen Vorteilen und Wechselwirkungen sind durch Web-Oberflächen in ein produktives Spannungsfeld getreten.

Die Entwickler von Web-Anwendungen arbeiten zum Beispiel mit .Net und Active-X, DHTML und JSP oder mit ZOPE und Python zwar in verschiedenen Lagern, doch zieht man dabei durchaus am gleichen Strang: Nämlich wieder den Komfort der Fat Client Verhältnisse einzuführen.

Die Bemühungen der Softwarehersteller zielen darauf ab, das aus der Fat Client-Welt gewohnte Interaktionslevel in die Web Client-Welt zu übertragen. Dies ist an sich kein systematisches Problem, sofern das Netz genügend Bandbreite zur Verfügung stellt.

Wenn jedoch zum Beispiel durch .Net die Grenze zwischen Transaktion und Präsentation wieder weich wird, könnte das negative Folgen haben. Eine nicht mehr beherrschbare, weil nicht exakt formulierbare Vermengung von Präsentations-, Prozess-, Anwendungs- und Geschäftslogik könnte ein systematisches Problem sein.

Die Service-orientierte Architektur (SOA) mit Web Services strebt die Bereitstellung von Funktionalitäten und Prozessbausteinen als Dienste an. Web Services ermöglichen das Bereitstellen von verteilten Funktionalitäten, die über XML-Schnittstellen angesprochen und damit leicht in Web User Interfaces eingebunden werden können.

Auf der Systemseite werden die Möglichkeiten der Aufbereitung und Darstellung von Informationen, auf der Anwenderseite die Möglichkeiten, mit diesen Informationen zu interagieren, komplexer [Khzaeli 2005].

Die Antwort auf die Frage, was sich eigentlich an der Anwendungsoberfläche abspielt und wie sich das mit der Transaktionslogik verzahnt, hat im Kontext der immer komplexer werdenden Anwendungen und der neuen technischen Möglichkeiten des Web eine neue Dringlichkeit erhalten.

Mit geeigneten User Interface-Modellen, die über das Aufsetzen von Masken auf Funktionen hinausgehen, kann das Risiko einer neuen komplexitätsbedingten Krise in der Softwareentwicklung begrenzt werden.

2.2 Oberfläche gliedern

Wer den Auftrag hat, eine Softwareoberfläche zu erstellen, muss Verschiedenes über die für ein ordentliches UI benötigten Dialogabläufe, Ablaufschritte, Kontrollelemente und Interaktionen in Erfahrung bringen.

Unabhängig davon, ob die Oberfläche für eine Kundenverwaltung, für ein Navigationssystem, für ein Verwaltungsprogramm oder für eine Vertriebsunterstützungssoftware ist, kann man die

Aufgabenstellung nach den Merkmalen der Softwareoberfläche analysieren und thematisch gliedern.

Die im Folgenden (in Form von Leitfragen) beschriebenen Klassifizierungsmerkmale können als Anleitung für eine UI-orientierte Anforderungsanalyse verwendet werden.

Gliedern der UI-Beschreibung mit Leitfragen

Leitfragen der UI-Beschreibung

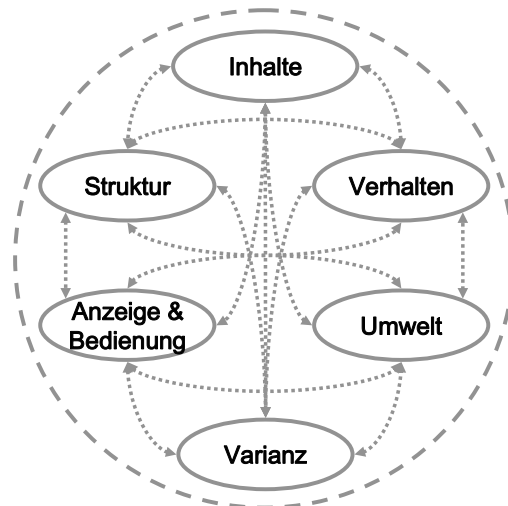
Um eine Anwendungsoberfläche entwerfen zu können, stellt der Entwickler bzw. der Analytiker eine Menge Fragen. Die Fragen können das generelle Aussehen, Verhalten und Funktion der Anwendung, einzelne Dialogseiten, Buttons oder Eingabefelder oder den Dialog zwischen Anwender und Oberfläche in speziellen Situationen betreffen. Sowohl die Fragen als auch die Antworten können schnell ein komplexes und schwer zu überblickendes Informationsgemenge bilden.

Die Gliederung der Fragen in merkmalsbezogene Themen kann helfen, die Antworten in eine strukturierte Form zu bringen und so den Überblick über eine UI-Beschreibung zu behalten.

Grobgliederung für eine UI-Beschreibung

Die beim Entwickeln eines User Interface aufkommenden Fragenkomplexe lassen sich wie folgt zu einer Grobgliederung für eine UI-Beschreibung formen.

Bild 15 Themengliederung für das Beschreiben einer Oberfläche



Die obige Abbildung zeigt, dass sich die Beschreibung eines User Interface in sechs verschiedene Themen gliedern lässt und dass die verschiedenen Themen einer UI-Beschreibung miteinander zusammenhängen.

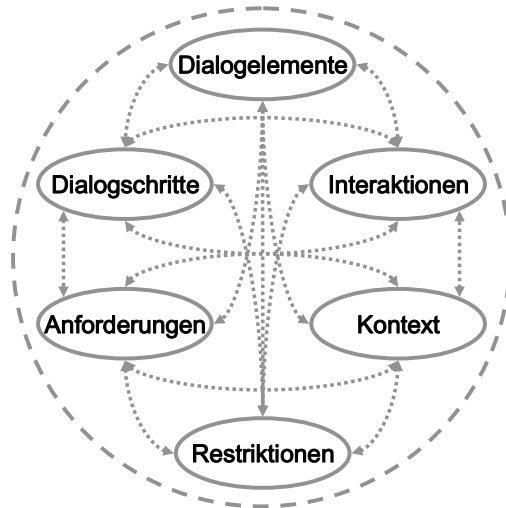


Bild 16:
Informationsgeflecht
einer UI-
Beschreibung

Die obige Abbildung zeigt das Informationsgeflecht, das in der Themengliederung enthalten ist und in einer UI-Beschreibung wiedergegeben wird.

Inhalte / Dialogelemente

Gegenstand der Anwendung, Funktionalität, Ablaufsequenzen

- **Was beinhalten die Abläufe und Inhalte der Anwendung aus fachlicher Sicht?**
 - **Standard-Dialogabfolgen (zuerst Normalfälle)**
 - **Inhalte von Dialogmasken (zuerst Hauptelemente)**
- **Wodurch sind diese Abläufe und Inhalte gekennzeichnet?**
 - **Varianz und Vermischung von Dialogabläufen**
 - **Maskenaufbau, typische Anwender-Interaktionen**

Struktur / Dialogschritte

Ablaufgliederung, Anwendungsaufbau, Ablaufzusammenhänge

- **Wie hängen die Aktivitäten und ihre Inhalte strukturell in der Anwendung zusammen?**
 - **Gliederung der Tätigkeiten**
 - **Zuordnung von Dialogmasken zu Ablaufschritten**
 - **Tätigkeitsauswahl**
 - **Wiederholung und Unterbrechungen**
 - **Datenorganisation**

Anzeige und Bedienung / Anforderungen

Darstellung, Bedienungssystematik, Anzeige-, Bedien- und Navigationsregeln

- **Wie wird eine konsistente Darstellung der Anwendungsinhalte erreicht?**
- **Wie wird der Anwender durch den Prozess und die Arbeitsschritte geführt?**
 - **Navigationselemente**
 - **Layouts**
 - **Widgets**

Verhalten / Interaktionen

Präsentationslogik, Schnittstelle zur Geschäftslogik

- **Nach welchen Regeln erfolgt die Steuerung der Anwendungsabläufe?**
- **Wie wird die Dynamik der Abläufe zwischen den Arbeitsschritten und innerhalb der Arbeitsschritte kontrolliert?**
 - **Ereignisse**
 - **Auswerten von Situationen**
 - **Reaktionen**
 - **Dialogfortschritt**

Umwelt / Kontext

Skalierung, Präsentations- und Funktionskontext

- **Wie werden die verschiedenen Sichten (rollen- und bearbeitungsstandbedingt) auf den gleichen Anwendungsinhalt abgebildet?**
- **Wie reagiert die Oberfläche auf sich ändernde Umweltbedingungen?**
- **Wie werden in Abhängigkeit vom Bearbeitungsstand die auszuführenden Funktionen und Funktionsparameter bestimmt?**
 - **Ermitteln von Situationen**
 - **Verwenden von Anwendungsfunktionen**
 - **Verwenden von Anwendungsdaten**

Varianz / Restriktionen

Umgebung, Varianten, Mandanten, Zulieferer