Benutzungsschnittstelle und Handbuch

Andreas Zeller + Gregor Snelting + Christian Lindig

Lehrstuhl Softwaretechnik
Universität des Saarlandes, Saarbrücken







Benutzungsschnittstellen

Bei der Einführung eines neuen Software-Systems in die Praxis kommt es oft zu Problemen, da die Benutzungsschnittstellen nur unzureichend an die Aufgabenstellung angepaßt sind.

- Unix-Anfänger haben Probleme mit `vi` und bash, die ihnen von erfahrenen Benutzern empfohlen werden.
- Das Zeichenprogramm Adobe Illustrator bietet kein magnetisches Raster – wie also Objekte ausrichten?
- Umstellung von Formular auf Programm (Steuererklärung).







Empfehlungen

Die VDI-Richtlinie 5005 „Software-Ergonomie in der Büro-Kommunikation“ beschreibt drei wichtige Anforderungen an Software-Systeme.

Kompetenzförderlichkeit Das Software-System soll *konsistent* und *handlungsunterstützend* gestaltet sein und so das Wissen des Benutzers über das Software-System steigern.

Handlungsflexibilität Das System muß alternative Lösungswege und Aufgabenstellungen unterstützen.

Aufgabenangemessenheit Das System muß erlauben, die Aufgabe gut und effizient zu erledigen.

Jede dieser Anforderungen hat konkrete Auswirkungen auf die Gestaltung der Benutzungsschnittstelle.

Konsistenz und Kompetenz

Die Interaktion zwischen dem Benutzer und der Software soll so gestaltet sein, daß der Benutzer kompetent mit den Software-Systemen umgehen kann und dadurch seine Handlungskompetenz gefördert wird.

Handlungskompetenz bedeutet, daß der Benutzer Wissen über die Software-Systeme und ihre organisatorische Einbettung erworben hat und daß er dieses Wissen auf die von ihm zu erfüllenden Aufgaben beziehen kann.

Grundsätze:

1. Benutzeroperationen sollen *konsistent gestaltet* sein
2. Benutzeroperationen sollen *die Aufgabenstellung unterstützen*







Konsistenz und Kompetenz steigern

- Objekt-Aktivierung und -Bearbeitung *einheitlich, übersichtlich und durchschaubar* darstellen. Wenig syntaktische Fehler zulassen.
- Nur *im Kontext anwendbare Funktionen* darstellen; sinnlose Funktionen blockieren (z.B. grau darstellen)
- *Letzte Parametereinstellung* beim Aufruf einer Menüoption anzeigen (z.B. Formatieren einer Diskette)
- *Sicherheitsabfragen* bei Operationen mit schwerwiegenden Folgen. Folgen verdeutlichen.
- *Undo/Redo*-Funktion anbieten, d.h. alle durchgeführten Operationen sollten storniert werden können.

  Auf Wunsch muß die Stornierung wieder aufgehoben werden (*Redo*).

  Mindestens einstufig. Wunsch mehrstufig.







Konsistenz und Kompetenz steigern (2)

- *Inkrementelle Aufgabenbearbeitung* ermöglichen, d.h. kleine, unabhängige, nichtsequentielle Teilschritte mit jeweiliger Ergebnisrückmeldung.

  Keine Operation darf in einer „Sackgasse“ enden.

- *Rückmeldungen* auf alle Benutzeroperationen. Anzeigen, ob Eingabe erwartet wird oder gerade eine Verarbeitung stattfindet. Überdurchschnittliche Verarbeitungszeiten anzeigen (Art, Objekt, Umfang oder Dauer).

  Systembedingte Verzögerungen, Unterbrechungen oder Störungen explizit anzeigen.







Regelwerke für die Dialoggestaltung

Das wichtigste Hilfsmittel, einheitliche Programmbedienung zu erhalten, sind *Regelwerke* (*style guides*) für die Oberflächengestaltung.

Beispiel: Dialoggestaltung mit der GNOME-Bibliothek, einem Linux-Standard:

- *All dialogs should have at least one button that is labeled “Close”, “Cancel”, “OK”, or “Apply”.*
- *Modal dialogs should be avoided wherever possible.*
- *The default highlighted button for a dialog should be the safest for the user.*
- *All dialogs should default to a size large enough to display all of the information in the dialog without making a resize necessary.*
- *All dialogs should set the titlebar with an appropriate string.*







Regelwerke für die Dialoggestaltung (2)

- *A dialog which consists of a single entry box shall have its "OK" button be the default (which is to say that ENTER shall accept the entry), and the ESCAPE key shall be bound to the Cancel button.*
- *In a dialog the "OK" or "Apply" button should not be highlighted until the user has made a change to the configurable data in the dialog.*
- *If a dialog contains more than one button for the destruction of that dialog, (for example, an "OK" and a "Cancel"), the affirmative button should always fall to the left of the negative.*

Style Guides können mehrere hundert Seiten umfassen!







Bekannte Style Guides

Verbreitete Stile und Regelwerke der Dialoggestaltung sind

- *Windows* (Windows Interface Application Design Guide) – 580 Seiten
- *Motif* (OSF/Motif Style Guide) – 400 Seiten
- *MacOS X* (Aqua Human Interface Guidelines) – 310 Seiten

Manche Betriebssysteme (z.B. X Window System/UNIX), Programme (z.B. StarOffice), und Bibliotheken (z.B. Swing, Qt) unterstützen mehrere Stile – entweder wahlweise oder gleichzeitig.

Manche Widget-Sets sehen auf allen Plattformen gleich aus – ein Verstoss gegen die Erwartungen des Benutzers (z.B. Apple iTunes auf Windows)

Generell nähern sich Aussehen und Verhalten der Oberflächen (*look and feel*) einander an.



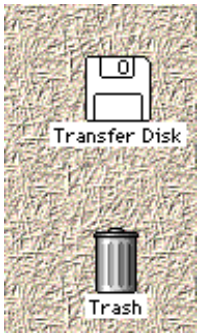




Inkonsistenzen - Beispiele

Den Sinn einheitlicher Regeln für Benutzungsschnittstellen erkennt man am besten, wenn man *Verstöße* betrachtet.

Auf einem *Macintosh*-Mülleimer kann man beliebige Objekte löschen, indem man sie auf den Mülleimer zieht.



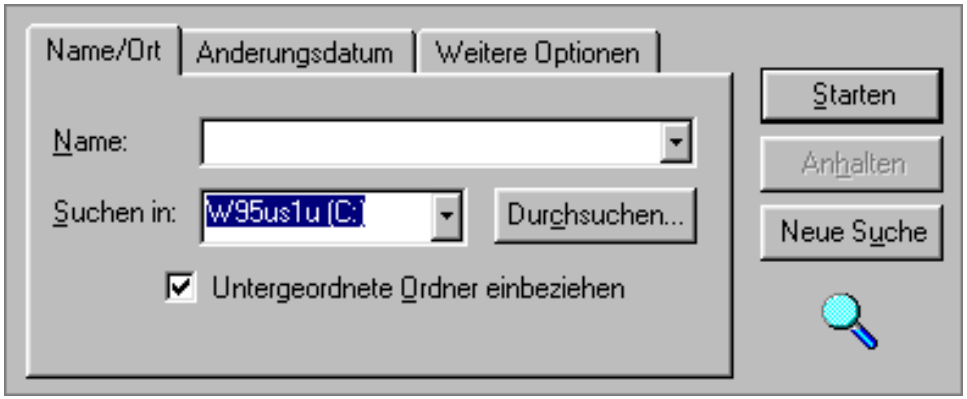
Zieht man jedoch eine Diskette auf den Mülleimer, wird sie nicht gelöscht, sondern ausgeworfen. Ein „magischer“ Mülleimer, der neue Benutzer verwirrt.







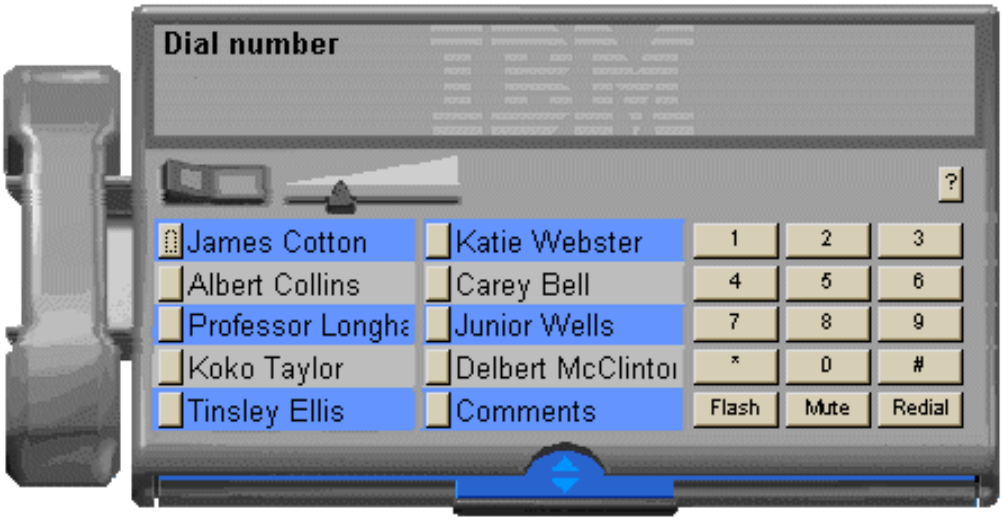
Suchen in Windows (seit WIN95)



Originellerweise wird die Suche mit *Starten* aktiviert – der *Durchsuchen*-Knopf erlaubt nur die Auswahl eines Startverzeichnisses.

Das IBM RealPhone

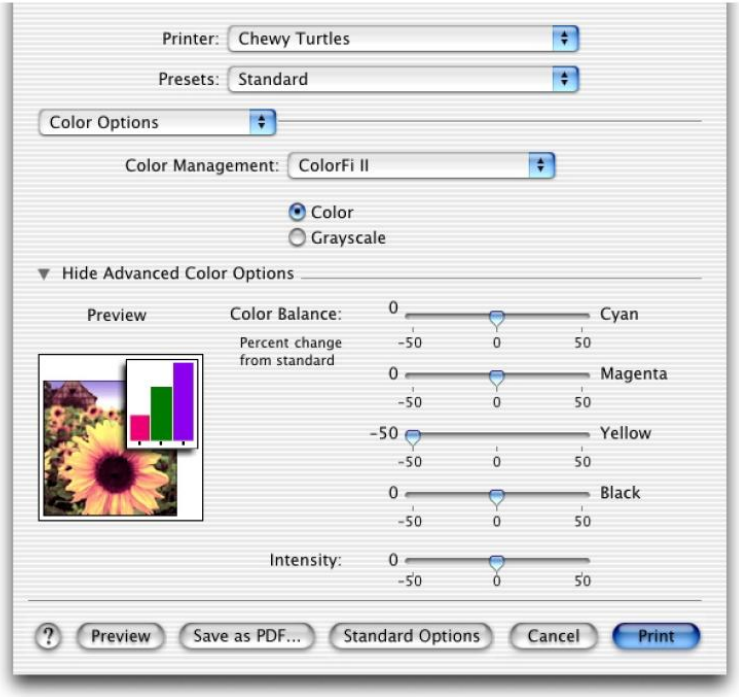
Das *IBM RealPhone*, ein Telefonprogramm, kommt ganz ohne Standard-Bedienungselemente aus:



In der wirklichen Welt mag es wichtig sein, anders auszusehen. Was Bedienung angeht, ist das Befolgen von Standards der richtige Weg.

Standardelemente

Benutzer erwarten standardisierte Bedienelemente, auch wenn es für sie keine physikalische Entsprechung gibt.



Kompetenzfördernde Dialoge

Die DIN-Norm 66324, Teil 8 führt zum Thema *Kompetenzförderlichkeit* besonders auf:

Selbstbeschreibungsfähigkeit Jeder Dialogschritt muß verständlich sein.

Erwartungskonformität Dialoge sollen den Erwartungen der Benutzer entsprechen

Fehlerrobustheit Dialoge sollen robust mit Fehleingaben umgehen.







Selbsterklärende Dialoge

Selbsterklärende Dialoge steigern die Handlungskompetenz, indem sie das Wissen über das Software-System fördern.

Ein Dialog ist selbsterklärend, wenn

- dem Benutzer auf Verlangen Einsatzzweck sowie Leistungsumfang des Dialogsystems erläutert werden können und wenn
- jeder einzelne Dialogschritt
  - unmittelbar verständlich ist oder
  - der Benutzer auf Verlangen dem jeweiligen Dialogschritt entsprechende Erläuterungen erhalten kann.







Selbsterklärende Dialoge (2)

Konkrete Maßnahmen für selbsterklärende Dialoge:

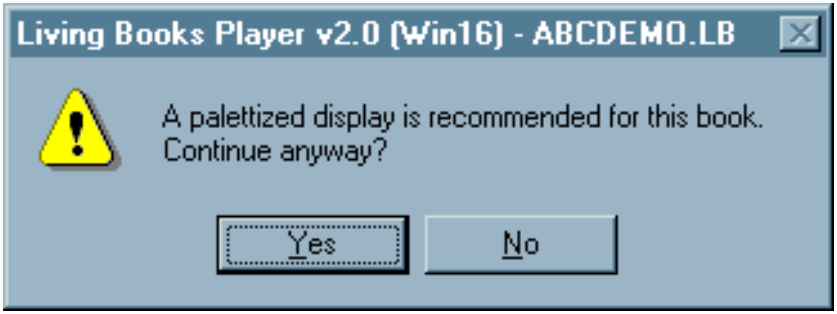
- Der Benutzer muß sich zweckmäßige Vorstellungen von den Systemzusammenhängen machen können
- Erläuterungen sind an allgemein übliche Kenntnisse der zu erwartenden Benutzer angepaßt (deutsche Sprache, berufliche Fachausdrücke)
- Wahl zwischen kurzen und ausführlichen Erläuterungen (Art, Umfang)
- Kontextabhängige Erläuterungen







Schlechte Erläuterungen



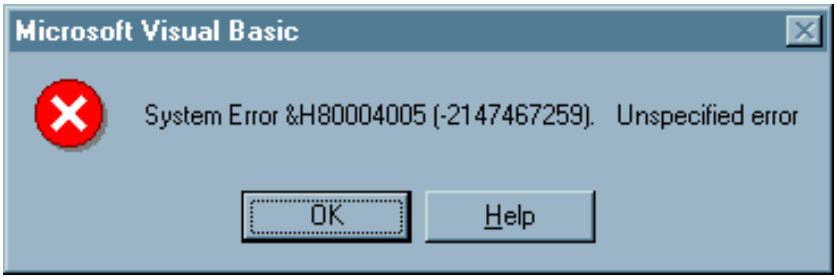
Was ist ein „palettized display“?

Ein PC-Experte mag diese Meldung vielleicht verstehen. Diese Warnung entstammt aber *Dr. Zeuss's ABC*, ein Alphabet-Lern-Programm für *3- bis 5jährige Kinder*.

Noch bemerkenswerter: Die Warnung ist völlig überflüssig, denn auch ohne „palettized display“ funktioniert das Programm einwandfrei.

Schlechte Erläuterungen (2)

Diese höchst aussagekräftige Fehlermeldung ist Microsoft *Visual Basic 5.0* zu entnehmen:



Nach dem Klicken auf *Help* erhalten wir:

> *Visual Basic encountered an error that was generated by the system or an external component and no other useful information was returned.*
>
> *The specified error number is returned by the system or external component (usually from an Application Interface call) and is displayed in hexadecimal and decimal format.*

Lösung des Problems: Neu booten?

Unix kann das auch...

Zum Schluß eine unfreundliche Fehlermeldung der *Secure Shell*:

```
$ ssh somehost.foo.com
You don't exist, go away!
$ _
```

Diese Fehlermeldung erscheint etwa, wenn der NIS-Server gerade nicht erreichbar ist. Nicht, daß man den Benutzer darüber aufklären würde...







Erwartungskonforme Dialoge

Die Handlungskompetenz wird durch *erwartungskonforme Dialoge* unterstützt.

Ein Dialog ist erwartungskonform, wenn er den Erwartungen der Benutzer entspricht,

- die sie aus Erfahrungen mit bisherigen Arbeitsabläufen oder aus der Benutzerschulung mitbringen sowie
- den Erwartungen, die sie sich während der Benutzung des Dialogsystems und im Umgang mit dem Benutzerhandbuch bilden.







Erwartungskonforme Dialoge (2)

Konkrete Maßnahmen für erwartungskonforme Dialoge:

- Das Dialogverhalten ist einheitlich (z.B. konsistente Anordnung der Bedienungselemente).
- Bei ähnlichen Arbeitsaufgaben ist der Dialog einheitlich gestaltet (z.B. Standard-Dialoge zum Öffnen oder Drucken von Dateien)
- Zustandsänderungen des Systems, die für die Dialogführung relevant sind, werden dem Benutzer mitgeteilt.
- Eingaben in Kurzform werden im Klartext bestätigt.
- Systemantwortzeiten sind den Erwartungen des Benutzers angepaßt, sonst erfolgt eine Meldung.
- Der Benutzer wird über den Stand der Bearbeitung informiert.

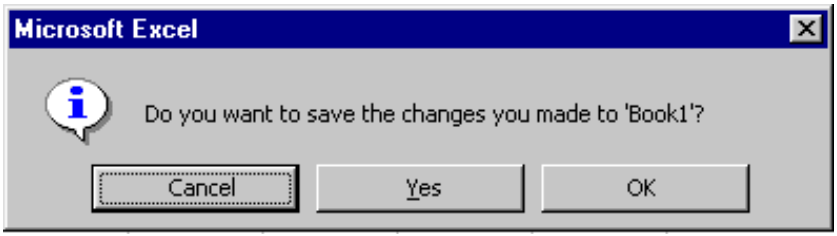






Unerwartetes Dialogverhalten

Gewöhnlich schreiben Standards vor, wie Bedienungselemente anzuordnen sind – etwa *Bestätigen* vor *Abbrechen*, oder *Ja* vor *Nein*, oder *Hilfe* ganz rechts. *Microsoft Excel* macht alles anders:



Wo ist der Unterschied zwischen *Yes* und *OK*?

. . . bei Freeloader

Wenn man *Freeloader*, einen WWW-Browser deinstalliert, erscheint dieses Fenster:

Der Benutzer mag denken, er könne auswählen, was denn nun tatsächlich deinstalliert werden soll. Weit gefehlt! Dies ist eine *Statusmeldung*, die anzeigt, was bereits deinstalliert wurde.

Originellerweise ändern die Knöpfe ihren Zustand, wenn man auf sie klickt – aber diese Änderung hat keine Auswirkungen.

Fehlerrobuste Dialoge

Ein Dialog ist *fehlerrobust*, wenn trotz erkennbar fehlerhafter Eingaben das beabsichtigte Arbeitsergebnis mit minimalem oder ohne Korrekturaufwand erreicht wird.

Dem Benutzer müssen Fehler verständlich gemacht werden, damit er sie beheben kann.

Fehlerrobuste Dialoge (2)

Konkrete Maßnahmen für fehlerrobuste Dialoge:

- Benutzereingaben dürfen nicht zu Systemabstürzen oder undefinierten Systemzuständen führen.

  Aus der GCC-Dokumentation:

  If the compiler gets a fatal signal, for any input whatever, that is a compiler bug. Reliable compilers never crash.

- Automatisch korrigierbare Fehler können korrigiert werden. Der Benutzer muß hierüber informiert werden.
- Die automatische Korrektur ist abschaltbar.
- Korrekturalternativen für Fehler werden dem Benutzer angezeigt.







Fehlerrobuste Dialoge (3)

- Fehlermeldungen weisen auf den Ort des Fehlers hin. z.B. durch Markierung der Fehlerstelle.
- Fehlermeldungen sind
  - verständlich,
  - sachlich und
  - konstruktiv

  zu formulieren und sind einheitlich zu strukturieren (z.B. Fehlerart, Fehlerursache, Fehlerbehebung).







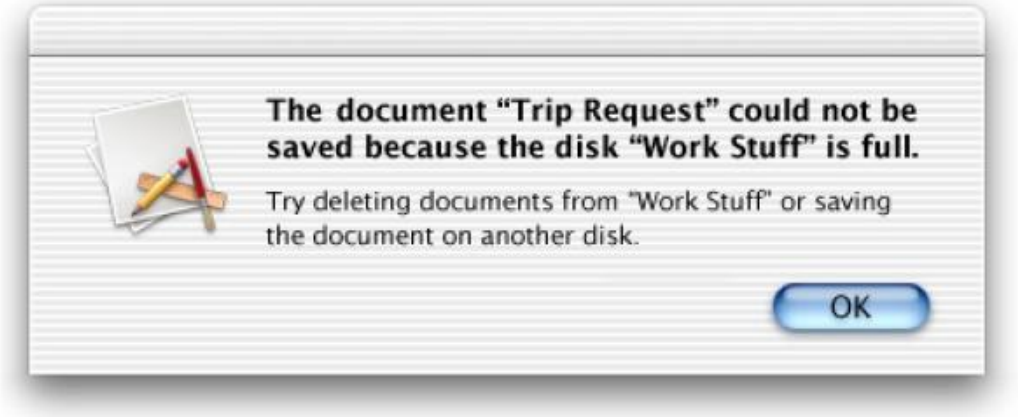
Eine schlechte Fehlermeldung

Eine Fehlermeldung, wie sie jeder von uns schon geschrieben hat.
Das Problem: wie soll der Benutzer darauf reagieren?

Eine bessere Fehlermeldung

Diese Fehlermeldung verrät die Ursache des Fehlers und gibt damit zumindest indirekt einen Hinweis, wie man ihn vermeiden könnte.

Eine gute Fehlermeldung



Eine Fehlermeldung, die keine Fragen offen lässt.

Auch schlecht

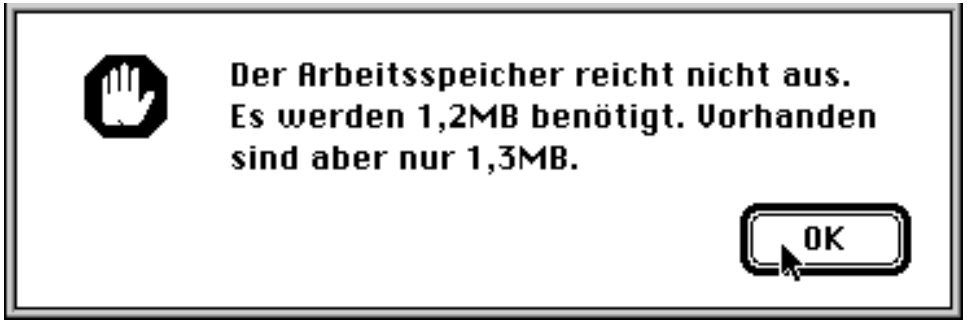
Diese Meldung erscheint in IBM's *Aptiva Communication Center*, wenn der Benutzer einen leeren Datensatz ausgewählt hat und auf den *Change*-Knopf klickt:

- Niemals eine Funktion anbieten, die in einer Fehlermeldung endet
- Stattdessen Funktionen *ausblenden*, die nicht angewandt werden können.

Denn Benutzer machen alle dummen Fehler die sie finden können!

Hä?

Zum Abschluß eine freundliche und aussagekräftige *Macintosh*-Meldung:



Flexibilität

Eine Anwendung ist dann *flexibel*, wenn

- der Benutzer mit einer *geänderten Aufgabenstellung* seine Arbeit noch effizient mit demselben System erledigen kann,
- eine Aufgabe auf alternativen Wegen ausgeführt werden kann, die dem Benutzer entsprechend seinem *wechselnden Kenntnisstand* und seiner aktuellen Leistungsfähigkeit wählen kann,
- *unterschiedliche Benutzer* mit unterschiedlichem Erfahrungshintergrund ihre Aufgaben auf alternativen Wegen erledigen können.

Flexibilität (2)

- *Makrobildung* ermöglichen, d.h. Operationen bei wiederkehrenden Abläufen können zu einer einzigen Operation zusammengefaßt werden.
- *Mengenbildung* ermöglichen, d.h. Objekte, auf die die gleichen Operationen angewendet werden sollen, können zu größeren Einheiten zusammengefaßt werden
- Soweit wie möglich *nicht-modale Dialoge* verwenden.

  Ein *modaler Dialog* schränkt im Rahmen einer Ausnahmesituation die Handlungsflexibilität ein (z.B. zur Fehlerbehebung, wenn erst danach weitergearbeitet werden kann).
- *Parallele Bearbeitung* mehrerer Anwendungen mit gegenseitigem Informationsaustausch vorsehen.

Generell: *Alternative Benutzeroperationen anbieten!*

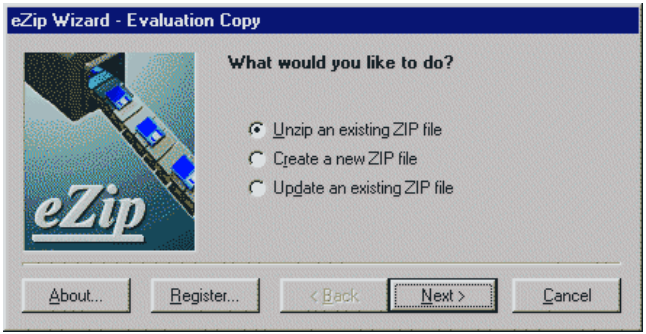






Beispiele für geringe Flexibilität

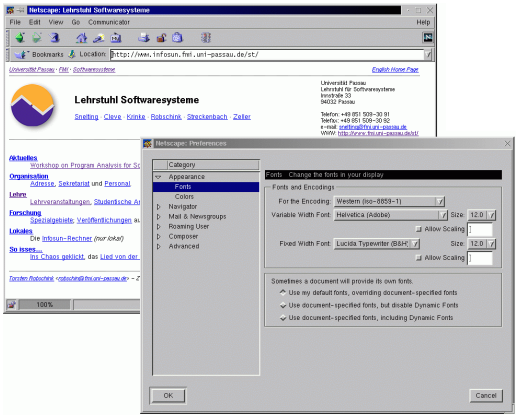
eZip ist ein Programm zum Entkomprimieren von Dateien. Die einzelnen Schritte sind genau vorgegeben:



- Was möchten Sie tun?
- Welche Optionen wünschen Sie?
- Welchen Namen möchten Sie angeben?
- usw. usf.

Netscape Navigator

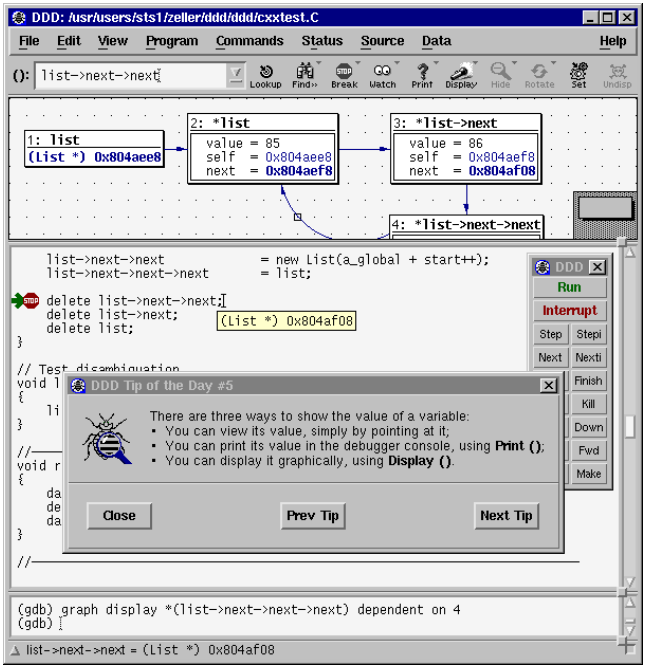
Einstellen von Benutzeroptionen, z.B. Schriftgrößen.



Unglücklicherweise ist der Einstellungs-Dialog *modal* – alle anderen Fenster sind inaktiv, während der Dialog geöffnet ist. Will der Benutzer häufig zwischen verschiedenen Schriftgrößen wechseln, muß er jedesmal den Dialog öffnen und wieder schließen.

Beispiele für große Flexibilität

DDD ist ein graphischer *Debugger*, mit dem der Ablauf eines Programms Schritt für Schritt untersucht werden kann.



DDD: Haltepunkt setzen

1. eine Zeile auswählen und auf *Break* klicken (Anfänger)
2. einen Funktionsnamen auswählen und auf *Break* klicken (Anfänger)
3. auf eine Zeile mit der rechten Maustaste klicken und *Set Breakpoint* aus einem Kontext-Menü auswählen (Fortgeschrittener)
4. einen Funktionsnamen mit der rechten Maustaste auswählen und *Set Breakpoint at* aus einem Kontext-Menü auswählen (Fortgeschrittener)
5. auf eine Zeile doppelklicken (Experte)
6. ein *break*-Kommando über die Tastatur eingeben (Experte) – oder
7. das Kommando zu *b* abkürzen. (Guru)







Effizienz und Angemessenheit

Der Benutzer soll seine Arbeitsaufgabe mit Hilfe der Anwendungen in einer Weise bearbeiten, die der Aufgabe angemessen ist:

- Kann der Benutzer die Zielsetzung seiner Aufgabe überhaupt mit dem System erreichen, oder muß er zusätzlich andere Systeme oder Medien einsetzen (z.B. Speicherung von Zwischenergebnissen auf Papier)
- Mit welchem Planungs- und Zeitaufwand (einschließlich des Aufwandes zur Korrektur von Fehlern) sowie mit welcher Qualität des Arbeitsergebnisses kann dieses Ziel erreicht werden?







Gegenanzeigen

Ein System ist immer dann *nicht* aufgabenangemessen, wenn

- bestimmte erforderliche Funktionalitäten nicht vorhanden sind (*fehlende Funktionalität*) oder
- „der Benutzer zur Ausführung eines in seinem Verständnis zusammenhängenden und einheitlichen Arbeitsschrittes eine größere Anzahl von Interaktionsschritten benötigt“ (*geringe Effizienz der Mensch-Rechner-Interaktion*).

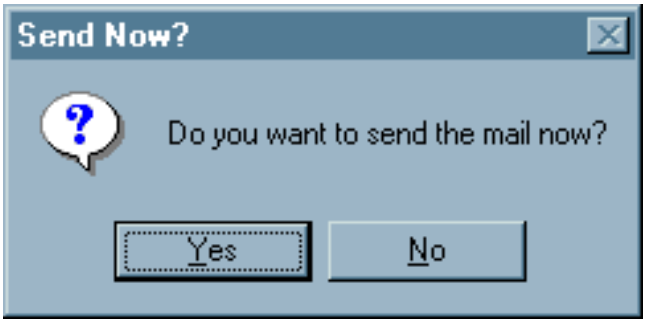
Effizienz steigern

- *Minimierung der Interaktionsschritte* zur Ausführung einer Aufgabe oder einer einzelnen Operation, z.B. durch
  - Mnemonische Auswahl von Menüoptionen über die Tastatur
  - Auswahl über Tastaturkürzel
  - Symbolbalken (Toolbar) mit häufig benutzten Funktionen
  - Aufführung der zuletzt benutzten Objekte / Einstellungen
  - Kommandosprache (ggf. mit Kontrollstrukturen)
- *Planungsaufwand reduzieren*, z.B. durch syntaktisch einfache Aufgaben oder Reduktion vieler Einzelschritte
- *Makro- und Mengenbildung*
- *Syntaktische Fehler* verhindern oder abfangen. Überflüssige Systemmeldungen, die der Benutzer quittieren muß, vermeiden.

Am besten ist die volle Funktionalität eines Menüsystems und einer Kommandosprache!

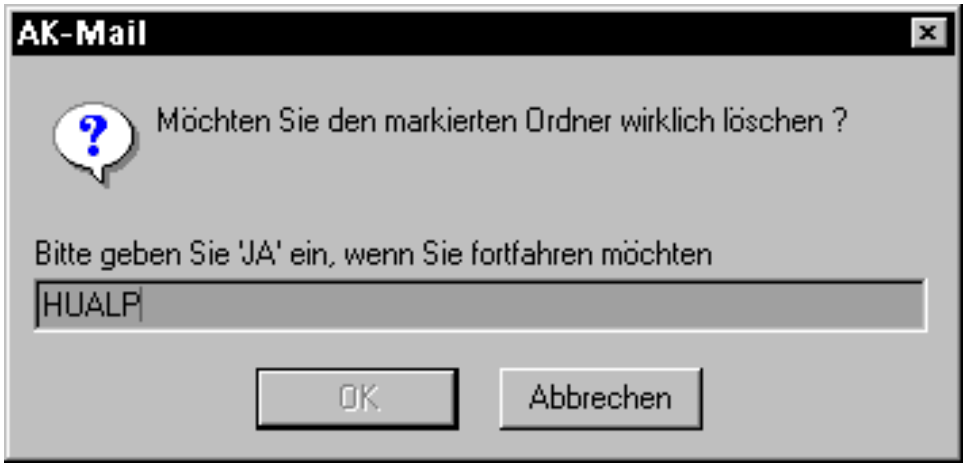
Beispiele für mangelnde Effizienz

Jedesmal, wenn man in *Lotus Notes* eine e-mail absenden will, verlangt Notes eine Bestätigung. Das kann auf Dauer recht anstrengend werden:



Wenn der Benutzer schon einmal auf *Absenden* gedrückt hat, warum nicht einfach anerkennen, daß er wohl *Absenden* meint?

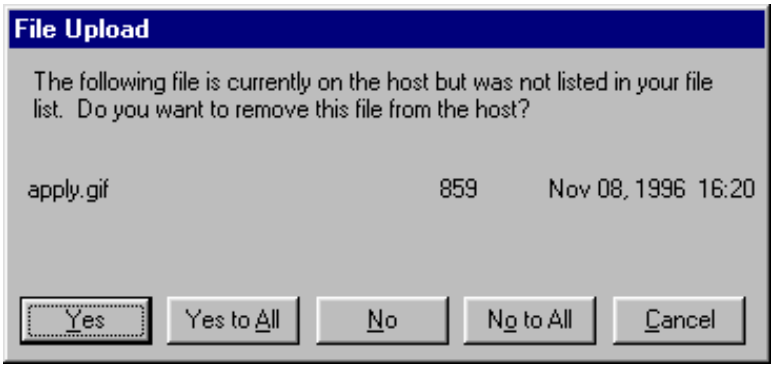
Noch schlimmer treibt es da *AK-Mail*:



Warum eigentlich *JA*? Warum nicht gleich *JAAA, VERFLUCHT!*?

. . .

Microsoft's *Web Wizard* ist ein Programm zum Verwalten von Dateien auf WWW-Servern. Während des Ablaufs erstellt Web Wizard eine Liste der Dateien auf dem Server und fragt ab, für jede Datei einzeln, ob sie gelöscht werden soll:

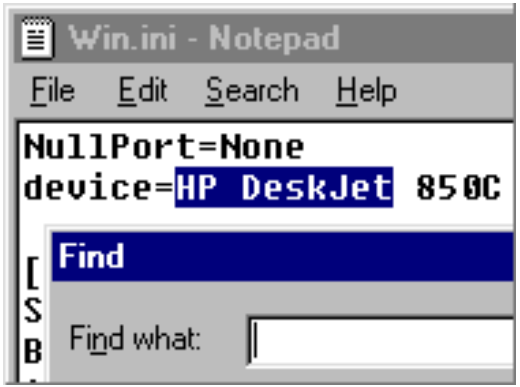


Wenn man etwa die Datei *zeller.gif* löschen wollte, müßte man zunächst 400mal auf *No* klicken – für jede der Dateien, die im Alphabet davor stehen.

. . .

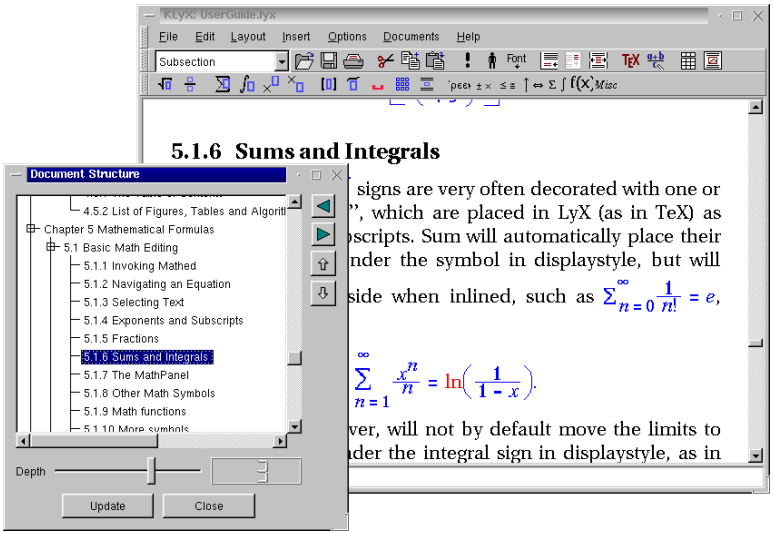
Wenn man in *Visual Basic* einen Text selektiert und dann *Search* aufruft, wird der selektierte Text zur Vorgabe. So sollte es sein.

In anderen Programmen wie etwa *Notepad* muß der Benutzer stattdessen aufwendig den Text über die Zwischenablage kopieren und einfügen:



Gute Effizienz

In den Textverarbeitungsprogrammen LyX und KLyX gibt es einen *Formeleditor*, mit dem Anfänger Formeln über ein Menü zusammensetzen können:



. . . mit der Tastatur

Der fortgeschrittene Benutzer kann aber auch direkt über die Tastatur TEX-Kommandos eingeben – etwa

```
\sum_{n = 1}^{\infty}\frac{x^n}{n} =
    \ln\left(\frac{1}{1 - x}\right)
```

für

$$\sum_{n=1}^{\infty} \frac{x^n}{n} = \ln\left(\frac{1}{1-x}\right) \quad .$$

Bereits mit der ersten Eingabe von \ schaltet KLyX in den TEX-Eingabemodus.

Flexible Programme sind meist auch effizient!

Benutzerfreundliche Web-Seiten

Die Top 10, um die Benutzungsfreundlichkeit zu erhöhen:

1. Name und Logo auf alle Seiten
2. Suchfunktion (bei > 100 Seiten)
3. Aussagekräftige Titelzeilen
4. Vertikales Scannen ermöglichen
5. Information per Hypertext strukturieren (statt Scrollen)
6. Kleine Photos benutzen
7. Übertragungsgeschwindigkeit maximieren
8. Links mit Titeln versehen
9. Seiten für Behinderte zugänglich machen
10. Konventionen von anderen Seiten übernehmen







. . . Bottom 10

. . . und umgekehrt die Top 10, um die Benutzungsfreundlichkeit zu *verringern:*

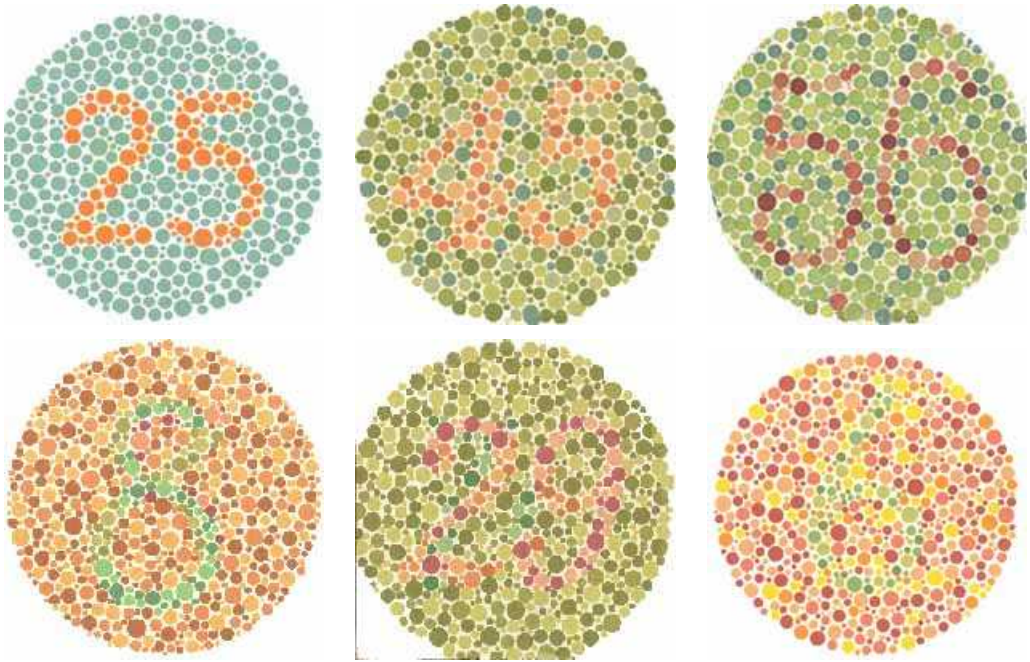
1. Frames
2. Neueste Plug-Ins
3. Scrollender Text
4. Lange URLs
5. Waisenseiten (= Error 404)
6. Scrollende Navigations-Seiten
7. Keine Unterstützung für Navigation
8. Eigene Farben für Links
9. Obsoleter Inhalt
10. Lange Ladezeiten







Farbe



Unfähr 8% aller Männer und 0.4% aller Frauen sind farbenblind;
häufig: Rot/Grün-Blindheit, selten: Blau/Grün-Blindheit.

Farbe

Grundsatz: *Farbe muss sparsam, gezielt und konsistent verwendet werden.*

- Farben sind emotional besetzt, nicht rational.
- Je reiner eine Farbe, desto kleiner ihre Fläche in einer GUI. Deshalb: neutrale Grundfarben verwenden.
- Farben sollen harmonieren (Trick: aus einem Landschaftsfoto herausgreifen).
- Eine Farbe wirkt nicht für sich, sondern in Kombination mit anderen. Viele Kombinationen sind ungeeignet, da sie zu wenig Kontrast aufweisen (Zum Beispiel: Blau/Rot).

Literatur: Edward Tufte, *Envisioning Information*, Graphics Press 1990.

Gegenbeispiele: http://www.themes.org/.

Benutzerfreundlichkeit prüfen

Um die Benutzerfreundlichkeit zu prüfen, gibt es nur ein Mittel: *Das System mit echten Benutzern testen!*

Typische Vorgehensweise: Benutzer sollen mit dem System eine bestimmte Aufgabe erledigen – und halten anschließend fest, was sie gestört hat.







Benutzerfreundlichkeit prüfen

Beispiel: In Linux-Maschine einloggen (Studie, 2001)

Probleme bei der Benutzung dieses Dialogs

- Was ist ein „Login“? Der Benutzername? Oder das Paßwort? (Konsistenz und Kompetenz)
- Username und Paßwort werden nicht gemeinsam abgefragt (Konsistenz und Kompentenz)
- Was soll der Benutzer nach der Eingabe des Namens tun? (Erwartungskonforme Dialoge)
- Bei falschem Benutzernamen/Paßwort erscheint kein Dialog (Erwartungskonforme Dialoge)
- Ist Klein-/Großschreibung relevant? Was passiert, wenn die Caps-Lock-Taste gedrückt ist? (selbsterklärende Dialoge)
- Was ist „marge-hci“? (selbsterklärende Dialoge)







Vorschlag für (leicht) verbesserten Dialog

Typically, when project managers observe their design undergoing a usability test, their initial reaction is:

Where did you find such stupid users?

Checkliste: Benutzungsoberfläche

Ist die Oberfläche konsistent gestaltet?

Dies wird in der Regel durch das Verwenden von Standard-Dialogen erreicht. Wo immer möglich, sollte man sich an Standards orientieren – Dialoggestaltung, Menüstruktur, Tastaturbelegung usw.

Sind die Dialoge selbsterklärend?

Sind die Dialoge erwartungskonform?

Auch dies wird durch Standard-Dialoge erreicht.

Checkliste: Benutzungsoberfläche (2)

Sind die Dialoge fehlerrobust?

Fehleingaben dürfen in keinem Fall zu Abstürzen oder undefiniertem Verhalten führen.

Meldungen müssen positiv sein. Keine Anklagen! Keine Beleidigungen! Keine Witze! Keine merkwürdigen Geräusche auf dem Lautsprecher!

Stets sollte das System vermitteln, daß es zu dumm ist, den Benutzer zu verstehen, und nicht, daß der Benutzer zu dumm ist, das System zu bedienen.







Checkliste: Benutzungsoberfläche (3)

Ist die Anwendung flexibel?

Nach Absprache mit dem Auftraggeber sollte die Anwendung möglichst so gestaltet werden, daß Aufgaben auf alternativen Wegen ausgeführt werden können.

Modale Dialoge sollten, wo immer möglich, vermieden werden.

Unterstützt die Oberfläche effizientes Arbeiten?

Das entscheidet der Auftraggeber.







Gestaltung von Benutzer-Handbüchern

Das *Benutzer-Handbuch* ist in der Regel der erste Kontakt eines Benutzers mit einem neuen Software-System.

Das Handbuch ist sozusagen die Visitenkarte des Systems; entsprechend sorgfältig sollte es gestaltet sein.

Das Benutzer-Handbuch soll die Handhabung und das Verhalten des Produkts möglichst vollständig und fehlerfrei beschreiben.

Es muß möglich sein, das Produkt nur anhand des Handbuchs zu bedienen.







Ökonomie von Benutzer-Handbüchern

Warum lohnt sich die Investition in gute *gedruckte* Handbücher?

- Die Benutzbarkeit von gedruckten Büchern übersteigt die von Online-Handbüchern.
- Gute Handbücher sind ein guter Kopierschutz.
- Klare Handbücher ersparen Folgekosten bei der Benutzerberatung.
- Handbücher vermitteln die Konzepte hinter einer Software, die sich aus der GUI nicht erschliessen lassen. Beispiele: Adobe Photoshop, Quark XPress, Programmiersprachen.

Lesenswert:
http://www.asktog.com/columns/017ManualWriting.html







Benutzer-Kategorien

Anordnung und Aufbau eines Handbuchs werden im wesentlichen durch die *Adressaten* bestimmt. Man unterscheidet drei Kategorien:

- *Anfänger* haben keine oder wenig Erfahrungen mit Computersystemen im allgemeinen und dem Produkt im speziellen.
- *Experten* haben viel Erfahrungen mit Computersystemen im allgemeinen, aber wenig Erfahrungen mit dem speziellen Produkt.
- *Fortgeschrittene* liegen irgendwo zwischen Anfängern und Experten; sie haben vielleicht schon ähnliche Produkte benutzt.







Handbuchtypen

Für jede Benutzer-Kategorie gibt es eigene Handbuch-Typen:

- Trainings-Handbuch (*Tutorial*)
- Referenz-Handbuch
- Referenzkarte
- Benutzer-Leitfaden (*user guide*)







Trainings-Handbuch (Tutorial)

Zielgruppe: *Anfänger*

Ein Trainings-Handbuch

- ist als Kurs oder Trainingsprogramm organisiert
- ist nach *Aufgaben* gegliedert:
  - Es beschreibt *Wege zur Erreichung von Arbeitszielen.* Beispiel: *So buchen Sie eine Anmeldung*
  - Die nötigen Arbeitsabläufe bestimmen die Gliederung.
- muß von Anfang bis Ende vollständig durchgearbeitet werden
- erfordert direkte Arbeit mit dem Produkt, d.h. der Leser führt die Anweisungen des Trainings-Handbuchs aus
- vermittelt rasch Erfolgserlebnisse
- ist für fortgeschrittene Benutzer und Experten zu elementar







Referenz-Handbuch

Zielgruppe: *Experten*

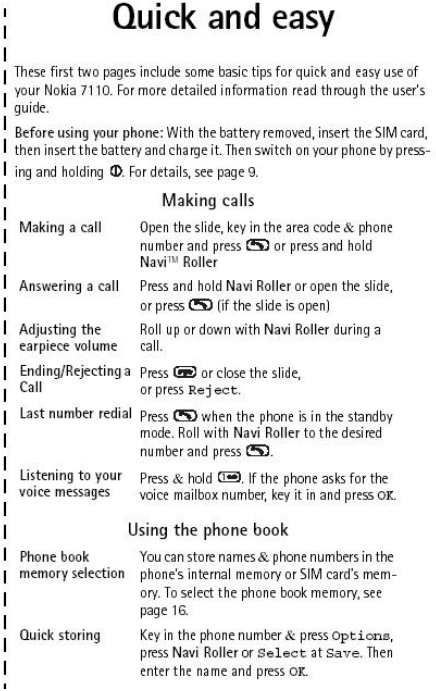
Ein Referenz-Handbuch

- ermöglicht schnellen Zugriff auf spezifische Information
- ist nach *Produktaufbau* gegliedert:
  - Alle Funktionen und Bestandteile werden in der Reihenfolge beschrieben, die sich aus der Struktur des Produktes ergibt. Beispiel: *Die Funktion „Anmeldungen bearbeiten“*
  - Es wird beschrieben, was das Produkt kann. . .
  - . . . und nicht, wie man Arbeitsziele mit dem Produkt erreicht
- bietet vollständige Informationen zu allen Produktfunktionen
- unterstellt, daß der Benutzer mehr weiß, als er tun will

Referenzkarte

Zielgruppe: Ebenfalls *Experten*
Eine Referenzkarte

- faßt die wesentlichen Informationen der wichtigsten oder häufigsten Funktionen zusammen.
- ist möglichst kompakt (nicht mehr als eine A4-Seite)
- Beispiel rechts aus Handbuch für Nokia 7110 Mobiltelefon.



Quick and easy

These first two pages include some basic tips for quick and easy use of your Nokia 7110. For more detailed information read through the user's guide.

Before using your phone: With the battery removed, insert the SIM card, then insert the battery and charge it. Then switch on your phone by pressing and holding ⏻. For details, see page 9.

Making calls

Making a call	Open the slide, key in the area code & phone number and press 📞 or press and hold Navi™ Roller
Answering a call	Press and hold **Navi Roller** or open the slide, or press 📞 (if the slide is open)
Adjusting the earpiece volume	Roll up or down with **Navi Roller** during a call.
Ending/Rejecting a Call	Press 📵 or close the slide, or press `Reject`.
Last number redial	Press 📞 when the phone is in the standby mode. Roll with **Navi Roller** to the desired number and press 📞.
Listening to your voice messages	Press & hold 1. If the phone asks for the voice mailbox number, key it in and press `OK`.

Using the phone book

Phone book memory selection	You can store names & phone numbers in the phone's internal memory or SIM card's memory. To select the phone book memory, see page 16.
Quick storing	Key in the phone number & press `Options`, press **Navi Roller** or `Select` at `Save`. Then enter the name and press `OK`.

Benutzer-Leitfaden (user guide)

Zielgruppe: *fortgeschrittene Benutzer*

Ein Leitfaden

- ist eine Mischform aus Trainings- und Referenzhandbuch
- muß simultan die Bedürfnisse beider Benutzergruppen, Anfänger und Experten, erfüllen.
- besteht typischerweise aus zwei Teilen:
  - Die aufgabenorientierte *Arbeitsanleitung* dient zum Einarbeiten.
    * nicht vollständig
    * erlaubt das Arbeiten mit Grundfunktionen.
  - Der *Referenzteil* dient zum späteren Nachschlagen.
    * erlaubt es, sich die weiteren Funktionen zu erschließen.







Benutzer-Leitfaden (2)

Ein Leitfaden

- erlaubt es, bereits bekannte Informationen zu überschlagen
- darf aber nicht davon ausgehen, daß der Leser Systemdetails bereits kennt
- ist bei einfachen Systemen das einzige Handbuch

Bei umfangreichen Systemen gibt es oft alle vier Handbuch-Typen!







Beispiel: TEXbook

Aufgaben vertiefen das Verständnis, Passagen für Experten sind besonders gekennzeichent.

take up the visual slack. The standard rule of thumb is to use \/ just before switching from slanted or italic to roman or bold, unless the next character is a period or comma. For example, type

```
{\it italics\/} for {\it emphasis}.
```

Old manuals of style say that the punctuation after a word should be in the *same* font as that *word;* but an italic semicolon often looks wrong, so this convention is changing. When an italicized word occurs just before a semicolon, the author recommends typing '`{\it word\/};`'.

▸ **EXERCISE 4.2**

Explain how to typeset a roman *word in the midst of an italicized sentence.*

Every letter of every font has an italic correction, which you can bring to life by typing \/. The correction is usually zero in unslanted styles, but there are exceptions: To typeset a bold '**f**' in quotes, you should say `a bold '{\bf f\/}'`, lest you get a bold '**f**'.

▸ **EXERCISE 4.3**

Define a control sequence `\ic` such that '`\ic c`' puts the italic correction of character *c* into TEX's register `\dimen0`.

Aufbau eines Benutzer-Handbuchs

Für Benutzer-Handbücher läßt sich kein generelles Gliederungsschema wie etwa Pflichtenhefte vorgeben. Neben dem sachlichen Inhalt gehören aber bestimmte Teile in jedes Handbuch:

Vorwort

Enthält Adressatenkreis, Anwendungsbereich, Änderungen gegenüber Vorversionen.

Ziel: der Leser muß nach den ersten Seiten wissen, ob Produkt und Handbuch für ihn und seine Anwendung bestimmt sind.

Manche Leser überblättern das Vorwort; deshalb gehören unverzichtbare Informationen in die *Einleitung*.

Inhaltsverzeichnis







Handbuch-Aufbau (2)

Einführung

Dient in der Regel als *Gebrauchsanleitung*, die über den Aufbau und den Umgang mit dem Handbuch informiert.

Beschreibt oft auch

- die Konfiguration, die für das Produkt erforderlich ist (wenn nicht in eigenem Kapitel).
- Zielgruppe
- Vorkenntnisse zum Verstehen des Handbuchs

Insgesamt soll die Einführung kurz sein.

Installation

Beschreibt, was der Benutzer tun muß, um das Produkt in Betrieb zu nehmen.

Wird nur selten gelesen, kann deshalb in den Anhang (dann muß aber in der Einführung darauf verwiesen werden).





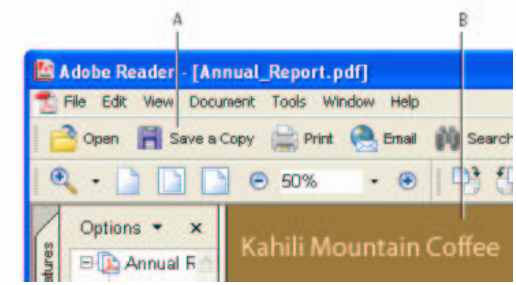


Handbuch-Aufbau (3)

Benutzungsoberfläche

Verdeutlicht den prinzipiellen Aufbau der Benutzungsoberfläche.
Insbesondere

- Tasten- und Mausbelegung
- grundsätzlicher Bildschirmaufbau (z.B. Fenstertypen)
- Dialogstrategie (z.B. erst Objekt, dann Funktion auswählen)



Produktstruktur

Gibt einen Überblick über die einzelnen Bestandteile des Produkts.

Handbuch-Aufbau (4)

Trainingsteil

(im Benutzer-Leitfaden oder Trainings-Handbuch)

Beschreibt *typische Arbeitsabläufe* mit Beispielen und Übungen, mit besonderem Wert auf *Routinearbeiten*.

Anordnung:

- häufigste Arbeitsabläufe und Routinearbeiten
- seltene Arbeitsabläufe
- Initialisieren und Löschen von Systemen

Beispiele müssen aus dem geplanten Anwendungsbereich stammen

Abstraktionen (also *var-1*, *var-2* statt *zins* und *kapital*) sind zu vermeiden

Übungen sind so zu gestalten, daß der Benutzer mit dem Produkt arbeitet – und dabei Erfolgserlebnisse hat.







Handbuch-Aufbau (5)

Referenzteil

(im Benutzer-Leitfaden oder Referenz-Handbuch)

Enthält eine vollständige Beschreibung der einzelnen Objekte und Funktionen.

Die Anordnung orientiert sich meistens an der Menüstruktur.

Kommandos werden alphabetisch angeordnet.

Behandlung von Problemen

Enthält eine Liste von Fehlermeldungen einschließlich detaillierter Erklärungen und Vorschläge







Handbuch-Aufbau (6)

Literaturverzeichnis

Abkürzungsverzeichnis

Glossar

Stichwortverzeichnis / Index / Register

Wichtigstes Hilfsmittel für einen schnellen Zugriff

Enthält Benutzerziele (z.B. *Anmeldung vornehmen*) und Funktionsnamen (z.B. *Buchung erfassen*)

Erstellt der Handbuch-Autor ein gutes Handbuch, dann sparen hunderte oder tausende Benutzer eine Menge Zeit und Nerven!

Hilfesysteme

Ein *Hilfesystem* unterstützt den Benutzer während der Benutzung durch explizite Erklärungen und Auskünfte.

Schnelle und vom Kontext abhängige Ergänzung zum Benutzer-Handbuch

Hilfesysteme erläutern typischerweise folgende Punkte:

- Aktuell wählbare Objekte, Funktionen, und Kommandos (und deren Optionen)
- Bedeutung von Funktionstasten und Mausknöpfen
- Hinweise zu Eingaben
- Erläuterungen zu Funktionsergebnissen
- Spezifische Fehlererklärungen (einschließlich Hilfen zur Korrektur)







Hilfesysteme (2)

Vorteile gegenüber dem Handbuch:

+ Texte in Hilfesystemen sind schneller und leichter zu aktualisieren.
+ Die Information in Hilfesystemen kann nicht verlorengehen.

Nachteile gegenüber dem Handbuch:

– Der Text auf dem Bildschirm wird *langsamer gelesen* als in einem Handbuch (kürzer fassen!)
– Notizen und Markierungen wie auf Papier sind nicht möglich.

Hilfesysteme (3)

Es gibt auch *Mischformen* aus Hilfesystemen und Handbüchern:

- Es gibt Programme, die *ausschließlich per Hilfesystem* dokumentiert sind; lediglich die Installationsanleitung kommt in gedruckter Form.
- Bei manchen Programmen (z.B. *Netscape*) ist die gesamte Dokumentation als WWW-Hypertext organisiert, auf den über das Netz zugegriffen wird.







Dynamische Hilfesysteme

Ein *statisches Hilfesystem* liefert stets dieselbe Information, unabhängig vom Kontext, z.B.:

- in einem Fenster wird immer die gleiche Erklärung gegeben.
- der UNIX-Editor ed gibt bei Fehlern aller Art die Meldung "*?*" aus.

Ein *dynamisches Hilfesystem* berücksichtigt den Kontext zum Zeitpunkt der Hilfeanforderung, z.B.:

- in jedem Eingabefeld eines Fensters wird eine spezifische Hilfe gegeben.
- eine Fehlermeldung lautet *Der eingegebene Wert 0.005 ist zu klein*

Dynamische Hilfesysteme sind stets vorzuziehen.







Individuelle Hilfesysteme

Ein *uniformes Hilfesystem* gibt jedem Benutzer dieselbe Hilfe unabhängig von seinem Kenntnisstand.

Ein *individuelles Hilfesystem* unterscheidet nach verschiedenen Benutzergruppen (Anfänger, Experte).

Die Einstufung kann auch automatisch erfolgen.

> Beispiel: Nachdem Sachbearbeiter Müller nur noch selten das Hilfesystem aufruft, wird er als vom Hilfesystem als *Experte* eingestuft und erhält nur noch eine Kurzform der Erklärungstexte.

Individuelle Hilfesysteme sind stets vorzuziehen.

Passive und aktive Hilfesysteme

Nur 40% der Funktionalität komplexer Systeme werden benutzt.

Manche Funktionen werden nur selten benutzt, da der Benutzer

- die Funktionen nicht im Detail kennt
- sich unsicher über ihre Details und Wirkungen ist

Das ist der Einsatzbereich von *passiven Hilfesystemen*.

Ein passives Hilfesystem erwartet, daß der Benutzer von sich aus eine Hilfeleistung anfordert, z.B.

- durch Drücken einer Taste (*F1* oder *Help*)
- durch Eingabe eines Kommandonamens ("*man ls*")
- durch Anfragen in natürlicher Sprache (*„Warum kann hier kein Text eingefügt werden?*")
- durch Anwahl eines Bedienelements (*balloon help*)

Passive und aktive Hilfesysteme (2)

Problem: Oft sind Konzepte realisiert, die der Benutzer nicht vermutet.

Ein *aktives Hilfesystem* beobachtet das Benutzungsverhalten und wird von sich aus aktiv, um dem Benutzer eine Hilfe zu geben.

> Beispiel: Um sich von einem Wort zum anderen zu bewegen, wurden einzeln Pfeiltasten benutzt. Der Word-Assistent erkennt diesen Vorgang und weist den Benutzer darauf hin, daß mit Ctrl+Pfeiltasten Wörter übersprungen werden können.

In der Regel werden heute passive und uniforme Hilfssysteme eingesetzt, die statische und dynamische Hilfeleistungen mischen.







Handbücher erstellen mit Texinfo

Texinfo ist ein einfaches System, das aus einer Quelle drei Dokumente erzeugt:

1. Ein *Handbuch* zum Ausdrucken.
2. Eine *HTML-Datei* als Online-Dokument im WWW.
3. Eine *Text-Datei* als Online-Hilfe.







Handbücher erstellen mit Texinfo (2)

Texinfo ergänzt einen Text um abstrakte Kontrollmechanismen, die die *Semantik von Textabschnitten* beschreiben:

```
@chapter So funktioniert Texinfo          % Kapitelüberschrift

@uref{http://www.gnu.org/software/texinfo/, @emph{Texinfo}}
erzeugt aus einer Quelle drei Dokumente:

@enumerate                          % Aufzählung
@item                               % Aufzählungspunkt
Ein Handbuch zum Ausdrucken.
@cindex Handbuch                    % "Handbuch" in Index aufnehmen

@item
Eine HTML-Datei als Online-Dokument im WWW.
@cindex HTML-Datei

@item
Eine Text-Datei als Online-Hilfe.
@cindex Text-Datei
@end enumerate
```







Handbücher erstellen mit Texinfo (3)

Das gedruckte Handbuch enthält einen WWW-Verweis:

> *Texinfo* (http://www.gnu.org/software/texinfo/) erzeugt aus einer Quelle drei Dokumente ...

Die WWW-Fassung sieht so aus:

> *Texinfo* erzeugt aus einer Quelle drei Dokumente ...

Die Text-Datei benutzt Asteriske zum Hervorheben:

```
*Texinfo* (http://www.gnu.org/software/texinfo/)
erzeugt ...
```

Jedes der Dokumente enthält ein Inhaltsverzeichnis und einen Index.







Handbücher erstellen mit Perl-POD

Dokumentationssystem von Perl. Ähnlich wie Texinfo, aber einfacher. Aus einer Quelldatei (`*.pod`) können verschiedene Formate erzeugt werden: HTML, Unix Manual Pages, LaTeX.

```
=head1 EXAMPLE

Here is a short example which shows how C<IO::Select> could
be used to write a server which communicates with several
sockets while also listening for more connections on a
listen socket

    use IO::Select;
    use IO::Socket;

    $lsn = new IO::Socket::INET(Listen => 1, LocalPort => 8080);
    $sel = new IO::Select( $lsn );
```

POD ≡ Plain Old Document Format.







Checkliste: Benutzer-Handbuch

Kann man leicht durch das Handbuch navigieren?

- Sind im Benutzer-Leitfaden und Trainings-Handbuch Kapitel und Abschnitte nach Benutzerzielen organisiert und benannt?
- Ist das Inhaltsverzeichnis gut gegliedert?
- Enthält das Stichwortverzeichnis Benutzerziele und Funktionsnamen?

Unterstützt das Handbuch leichtes Erlernen?

- Richtet sich die Sprache an den Endanwender?
- Wird die Information in kleinen Einheiten dargeboten?
- Wird die Information in einer logischen Sequenz dargeboten?
- Gibt es Beispiele? Zeichnungen? Grafiken?
- Werden visuelle Kennzeichen (Typographie) konsistent verwendet?
- Wird eine „Vermenschlichung“ des Computers vermieden?







Checkliste: Benutzer-Handbuch (2)

Ist das Handbuch gut lesbar?

Lesbarkeit wird insbesondere erreicht

- durch großzügigen Gebrauch von Leerraum
- durch Vermeiden von unnötigem Jargon

Sind die Pflichtteile enthalten?

Gibt es geeignete Szenarien?

Benutzer-Leitfaden und Trainings-Handbuch müssen typische Szenarien im Umgang mit dem System enthalten, und beschreiben, wie sich das System verhält.

Diese Szenarien bilden die Grundlage für spätere Tests (und sollten deshalb einzeln gekennzeichnet sein).

Ist eine Hilfefunktion vorhanden?

Das System sollte zumindest eine statische, passive, uniforme Hilfe anbieten.