# Benutzungsschnittstellen

**Andreas Zeller + Christian Lindig**
Lehrstuhl für Softwaretechnik
Universität des Saarlandes, Saarbrücken

2005-10-31

## Benutzungsschnittstellen

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

## Anforderungen

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.

## Anforderungen

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:[1]

- *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!

[1] http://www.gnome.org/

## 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.

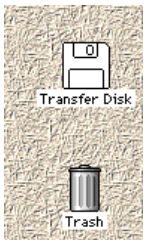
Manche Widget-Sets sehen auf allen Plattformen gleich aus – ein Verstoss gegen die Erwartungen des Benutzers.

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

## Inkonsistenzen - Beispiele [2]

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

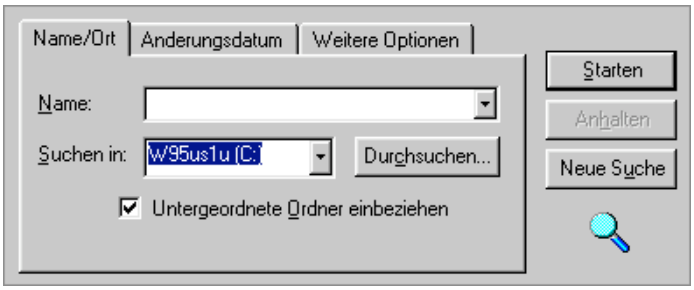
Auf einem *Macintosh*-Mülleimer kann man beliebige Objekte löschen, indem man sie auf den Mülleimer zieht.




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

Frage für Macintosh-Experten: Wie lösche ich eine Diskette?

[2] Alle Beispiele aus: *Interface Hall of Shame* (http://www.iarchitect.com/mshame.htm), dem *Museum für falsche Fehlermeldungen* (http://www.moffem.de/) sowie *Dialoge boxen* (http://www.dasding.de/dialoge/dialog2.htm)




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

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

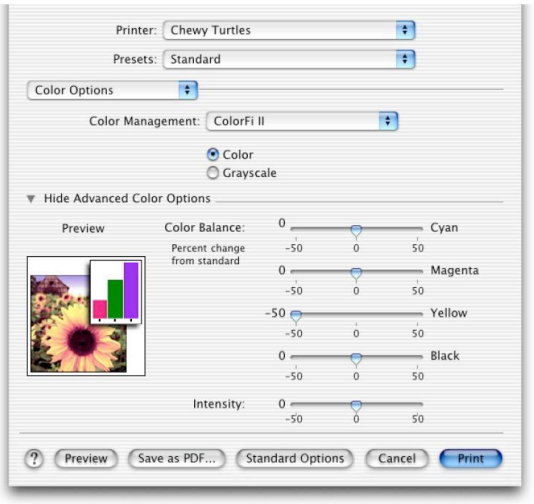

Hier ein paar einfache Aufgaben:

- Wie wähle ich eine Nummer über die Tastatur?
- Wie programmiere ich eine der zehn Kurzwahltasten?
- Was muss ich tun, um mehr als zehn Kurzwahltasten zu erhalten?
- Wie hebe ich den Hörer ab?
- Was macht eine Schublade in einem Telefon?
- Was passiert, wenn ich aus Versehen auf das *RealPhone* klicke?

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.

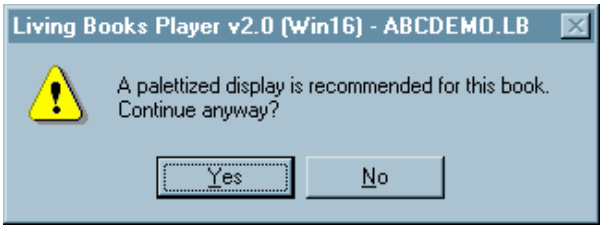
Ein Dialog ist selbsterklärend, wenn

- 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.

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

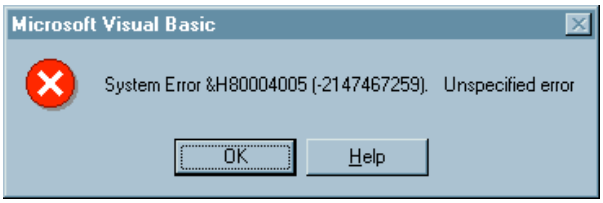



Was ist ein „palettized display“?

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.

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.*

Das bedeutet:

> Irgendetwas ist passiert. Wir wissen nicht, was hier passiert ist oder warum es passiert ist. Wir wissen nur, dass die dargestellte hexadezimale Zahl eine hexadezimale Zahl ist, aber die Zahl selbst hat keine Bedeutung.

Was nicht in der Erläuterung steht: Die Lösung des Problems. Neu booten.

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,

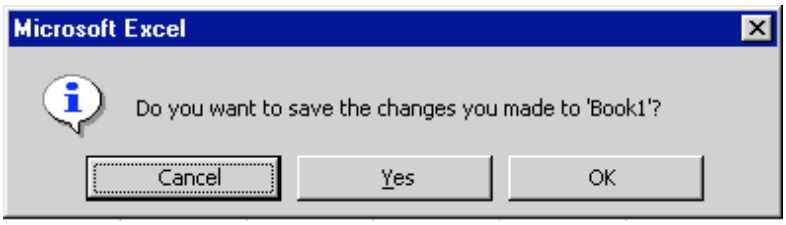
- die sie aus Erfahrungen mit bisherigen Arbeitsabläufen oder aus der Benutzerschulung mitbringen sowie
- den Erwartungen, die sie sich während der Benutzung des Dialogsystems und im Umgang mit dem Benutzerhandbuch bilden.

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*?

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

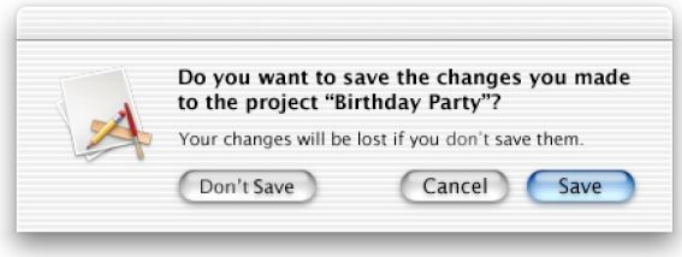

Der Benutzer mag denken, er könne auswählen, was denn nun tatsächlich deinstalliert werden soll. Weit gefehlt! Dies ist eine *Statusmeldung*, die anzeigt, was bereits deinstalliert wurde.

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)

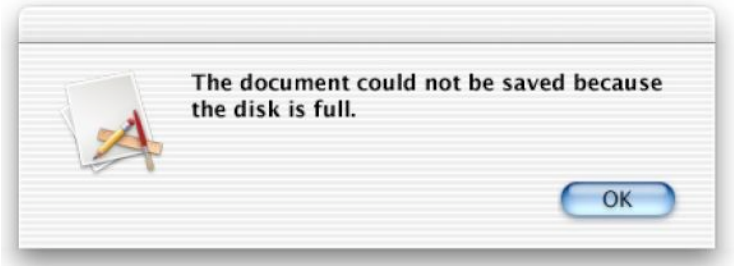
- Fehlermeldungen weisen auf den Ort des Fehlers hin. z.B. durch Markierung der Fehlerstelle.
- Fehlermeldungen sind
  - verständlich,
  - sachlich und
  - konstruktiv

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

## Eine schlechte Fehlermeldung

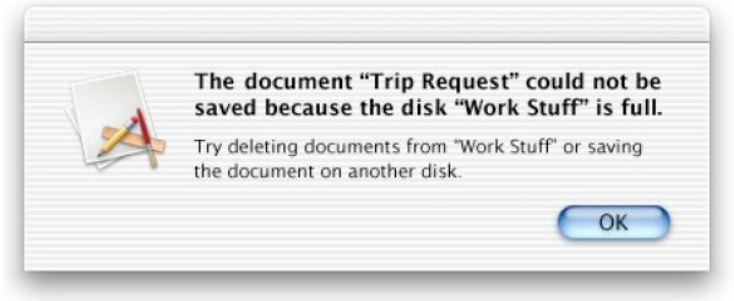

Eine Fehlermeldung, wie sie jeder von uns schon geschrieben hat. Das Problem: wie soll der Benutzer darauf reagieren?

## 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

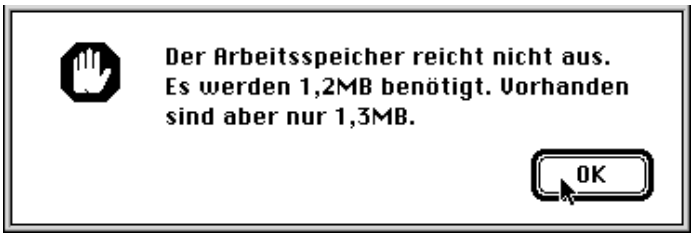
Diese Meldung erscheint in IBM's *Aptiva Communication Center*, wenn der Benutzer einen leeren Datensatz ausgewählt hat und auf den *Change*-Knopf klickt:


- 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:




## Anforderungen

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.

## 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)

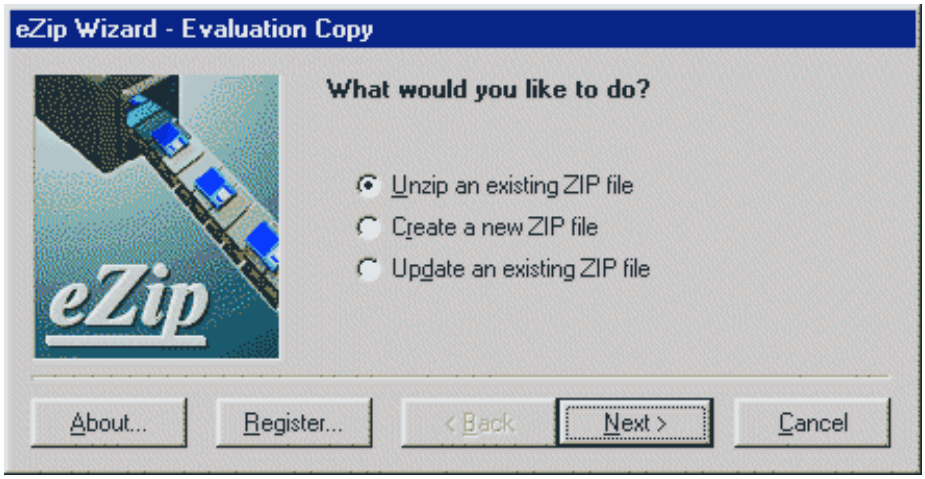
- *Makrobildung* ermöglichen, d.h. Operationen bei wiederkehrenden Abläufen können zu einer einzigen Operation zusammengefasst werden.
- *Mengenbildung* ermöglichen, d.h. Objekte, auf die die gleichen Operationen angewendet werden sollen, können zu größeren Einheiten zusammengefasst werden
- Soweit wie 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:

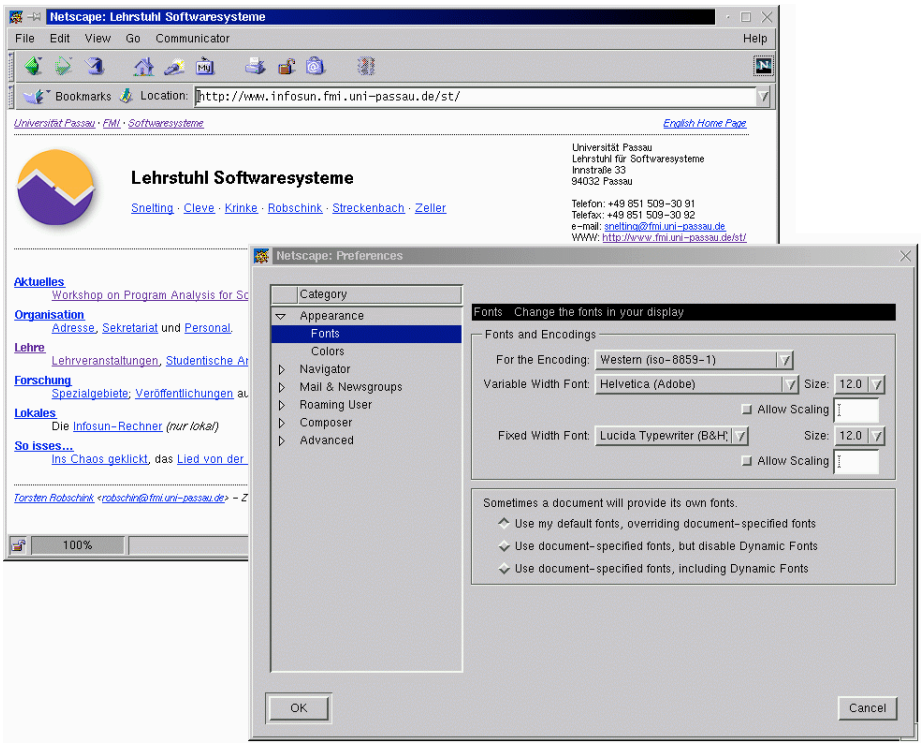



- Was möchten Sie tun?
- Welche Optionen wünschen Sie?
- Welchen Namen möchten Sie angeben?
- usw. usf.

wobei der Benutzer auf jede Frage antworten muss, bevor er zum nächsten Schritt gelangt. Dies mag für Erstbenutzer sinnvoll sein; erfahrene Benutzer sind schlichtweg genervt.

## *Netscape Navigator*

Einstellen von Benutzeroptionen, z.B. Schriftgrößen.

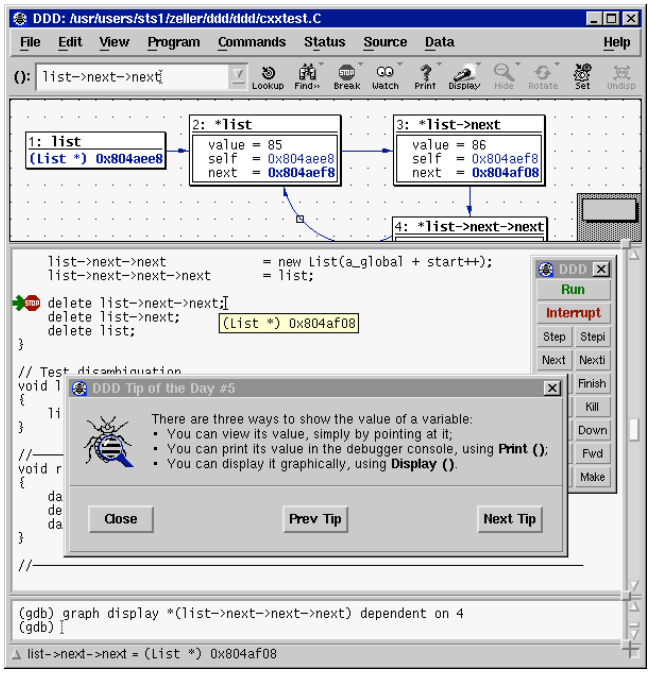



Unglücklicherweise ist der Einstellungs-Dialog *modal* – alle anderen Fenster sind inaktiv, während der Dialog geöffnet ist. Will der Benutzer häufig zwischen verschiedenen Schriftgrößen wechseln, muss er jedesmal den Dialog öffnen und wieder schließen.

Dieser Dialog muss nicht modal sein. Viel bequemer wäre es, nach Belieben zwischen den einzelnen Fenstern hin- und herzuschalten. Offensichtlich sind die Einstellungen nur deshalb modal, weil sich die Programmierer Arbeit ersparen wollten.

## Beispiele für große Flexibilität

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




## DDD: Haltepunkt setzen

So kann der Benutzer in DDD einen Haltepunkt (*Breakpoint*) 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)

## Anforderungen

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.

## 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*).

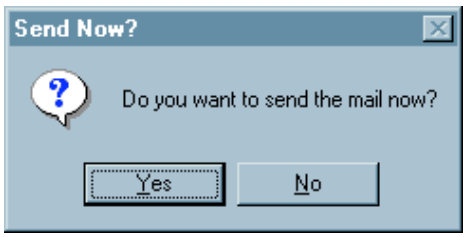
## Effizienz steigern

- *Minimierung der Interaktionsschritte* zur Ausführung einer Aufgabe oder einer einzelnen Operation, z.B. durch
  - Mnemonische Auswahl von Menüoptionen über die Tastatur
  - Auswahl über Tastaturkürzel (z.B. *Strg+X* statt *Bearbeiten→Ausschneiden*)
  - 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* (s. Abschnitt )
- *Syntaktische Fehler* verhindern oder abfangen. Überflüssige Systemmeldungen, die der Benutzer quittieren muss, vermeiden.

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

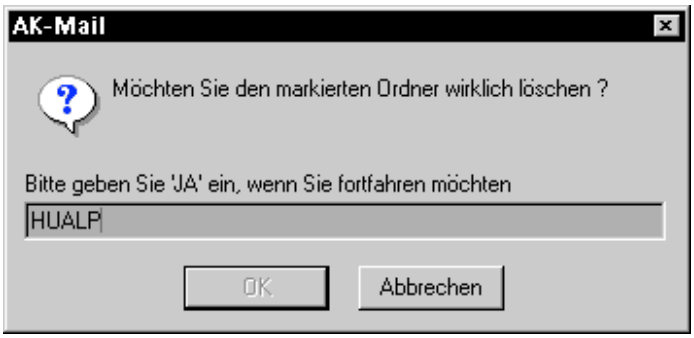
## Beispiele für mangelnde Effizienz

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



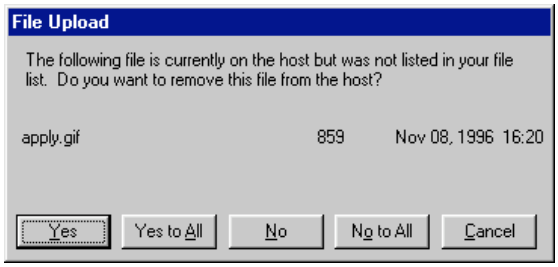

Wenn der Benutzer schon einmal auf *Absenden* gedrückt hat, warum nicht einfach anerkennen, dass er wohl *Absenden* meint?

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:

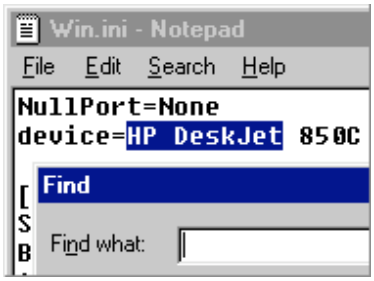



Wenn man etwa die Datei *zeller.gif* löschen wollte, müßte man zunächst 400mal auf *No* klicken – für jede der Dateien, die im Alphabet davor stehen.

Sehr viel effizienter wäre es hier, das Programm würde einfach die komplette Liste der Dateien anzeigen und Löschen von Gruppen anbieten.

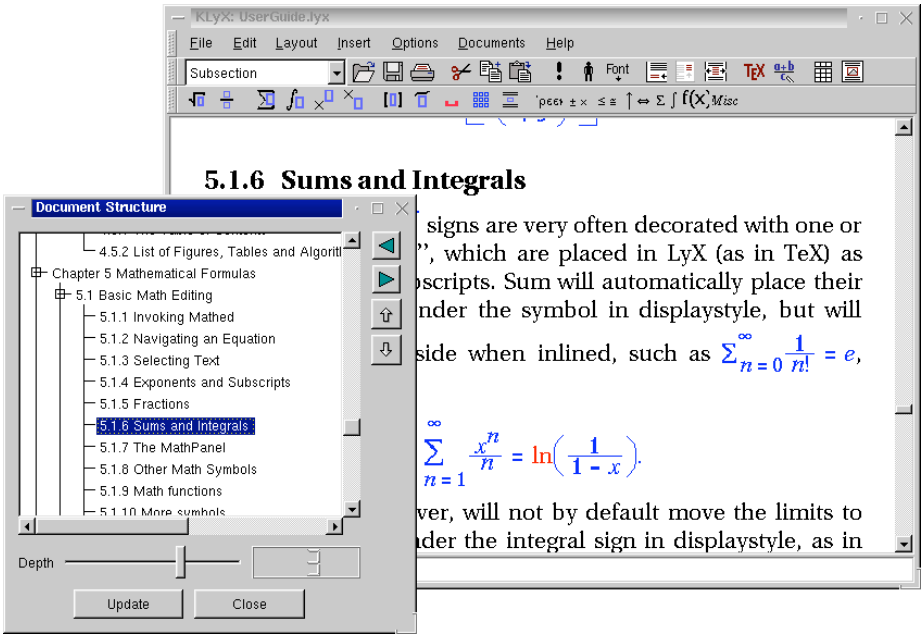
Wenn man in *Visual Basic* einen Text selektiert und dann *Search* aufruft, wird der selektierte Text zur Vorgabe. So sollte es sein.

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– von Jakob Nielsens Alertbox[3]:

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

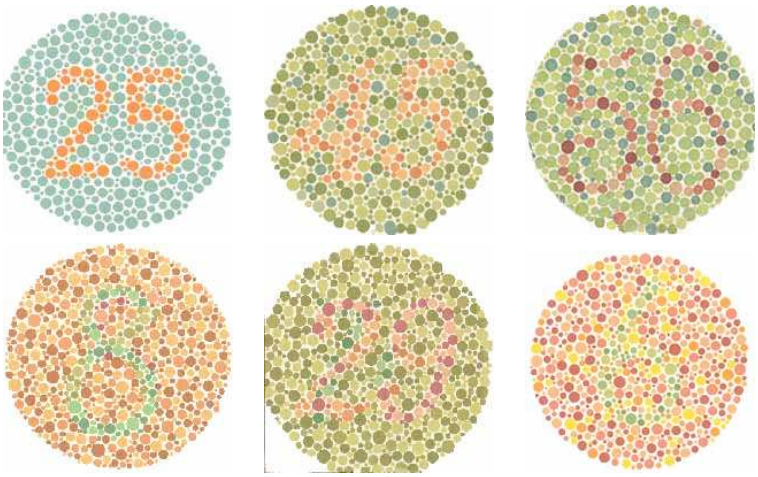
## ...Bottom 10

...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

[3] http://www.useit.com/alertbox/

## 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 ungeignet, 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

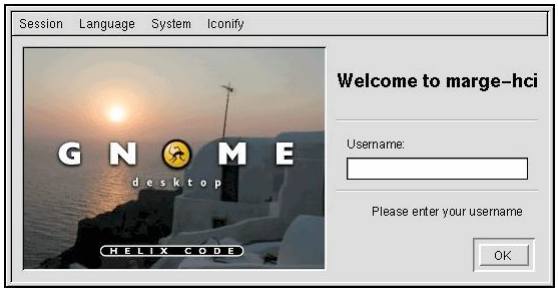
Beispiel: In Linux-Maschine einloggen (Studie, 2001)[4]


## 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)

[4] Suzanna Smith et al., *GNOME Usability Study Report*, http://developer.gnome.org/projects/gup/ut1_report/

## 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?*[5]

## Konzepte

Benutzeroberflächen müssen konsistent, selbsterklärend und erwartungskonform sein.

- Wird in der Regel durch Befolgen von *Standards* erreicht.

Die Anwendung sollte möglichst *flexibles* und *effizientes* Arbeiten ermöglichen.

- Lässt sich am besten durch *Benutzerstudien* prüfen.

[5] http://www.useit.com/alertbox/20010204.html