



# Benutzungsschnittstellen

Gregor Snelting, Andreas Zeller und Holger Cleve

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 Probleme angepaßt sind.





# 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 (II)***

- *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:<sup>1</sup>

- *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.*

---

<sup>1</sup><http://www.gnome.org/>





- *All dialogs should set the titlebar with an appropriate string.*
- *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 bis zu hundert Seiten umfassen!







# Bekannte Style Guides

---

Verbreitete Stile und Regelwerke der Dialoggestaltung sind

- *Windows* (Windows Interface Application Design Guide)
- *Motif* (OSF/Motif Style Guide)
- *MacOS* (Macintosh Interface Guidelines)

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.

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





## Inkonsistenzen - Beispiele <sup>2</sup>

---

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.

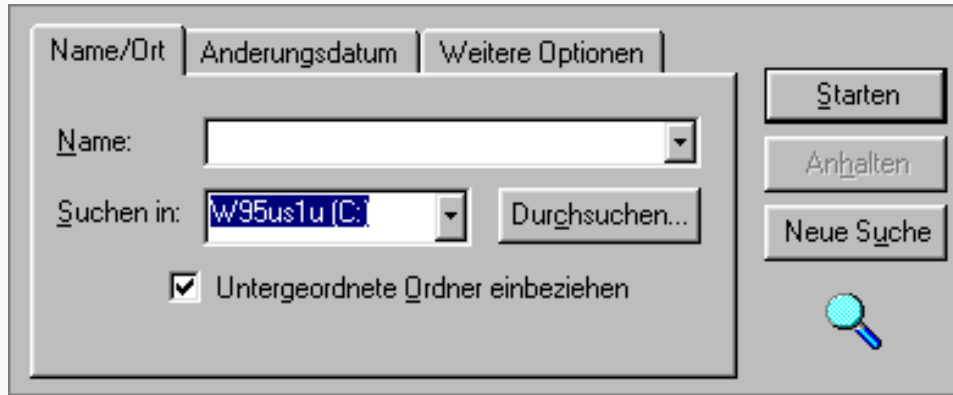
---

<sup>2</sup>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>)





## Suchen in Windows (seit WIN95)



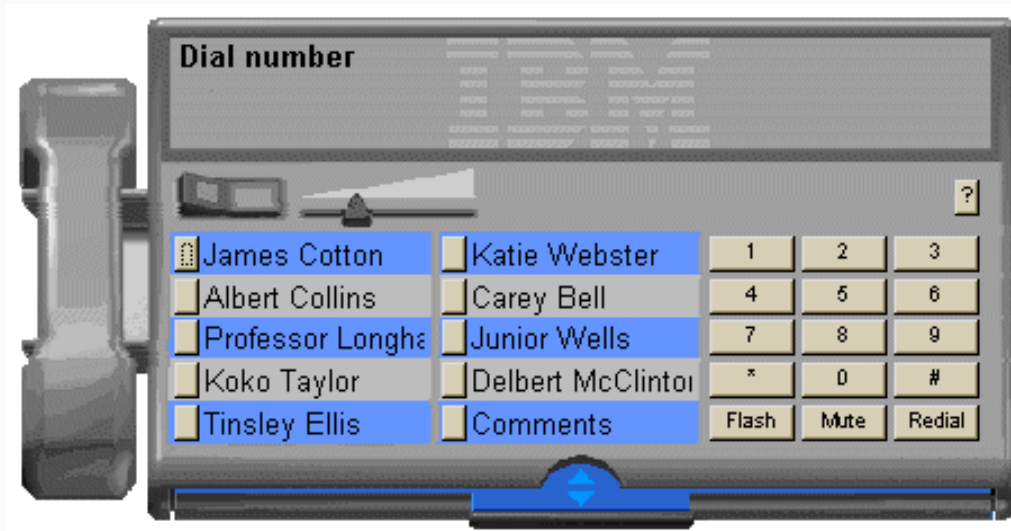
Originellerweise wird die Suche mit *Starten* aktiviert – der *Durchsuchen*-Knopf erlaubt nur die Auswahl eines Startverzeichnisses.





# 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.





# ***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 (II)*

---

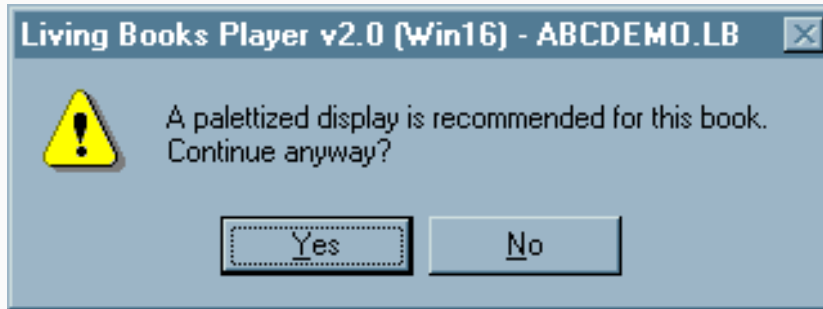
Konkrete Maßnahmen für selbsterklärende Dialoge:

- 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.



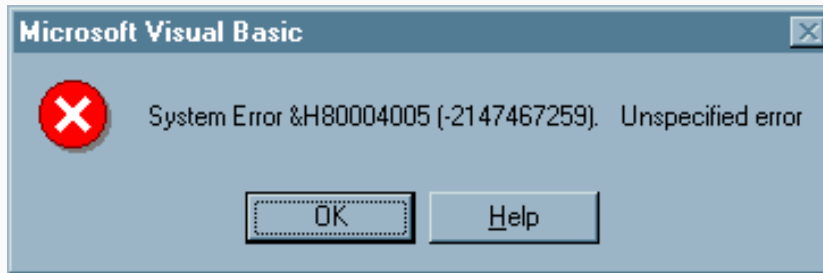




## Schlechte Erläuterungen (II)

---

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 (II)*

---

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.

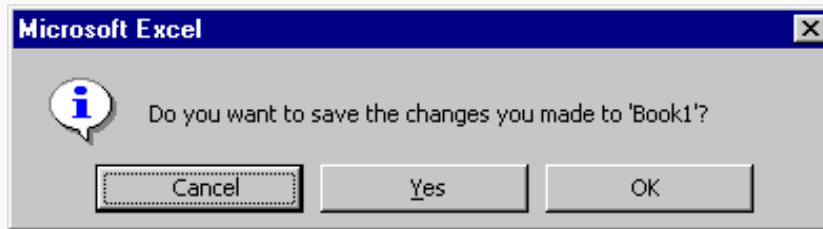




# *Unerwartetes Dialogverhalten*

---

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*?

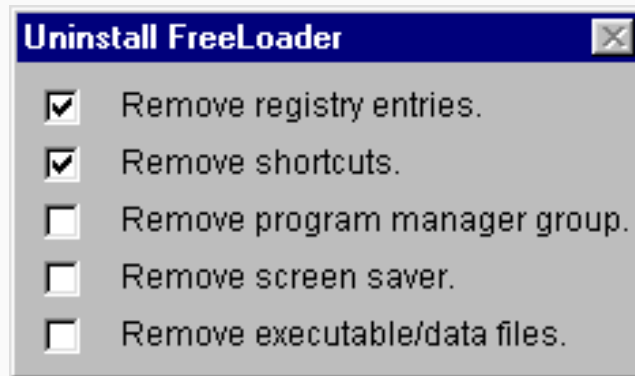




## ... bei FreeLoader

---

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 (II)***

---

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.
- Fehlermeldungen weisen auf den Ort des Fehlers hin. z.B. durch Markierung der Fehlerstelle.







## *Fehlerrobuste Dialoge(III)*

---

- Fehlermeldungen sind
  - verständlich,
  - sachlich und
  - konstruktiv

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



# Schlechte Fehlermeldungen

---

Dies ist eine Fehlermeldung von Eudora, einem E-Mail-Programm:



„503 höfliche Leute sagen erstmal Hallo.“ ■

Ein Programmierer, der mit dem SMTP-Protokoll vertraut ist, mag hiermit etwas anfangen können. Der gewöhnliche Benutzer aber wird sich fragen: „Häh? Was habe *ich* falsch gemacht?“

Was würden wohl 503 unhöfliche Leute sagen?



# Waddyamean?

---

Auch hier war der Programmierer zu faul, einen sinnvolle Meldung abzuliefern, geschweige denn, Hinweise, wie man den Fehler beheben kann:



Erschwerend kam hinzu, daß die arme Sekretärin, mit dieser Meldung konfrontiert, ein ums andere Mal „mismatch“ eintippte („type“ = tippen), ohne daß irgendetwas passierte.





## 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.

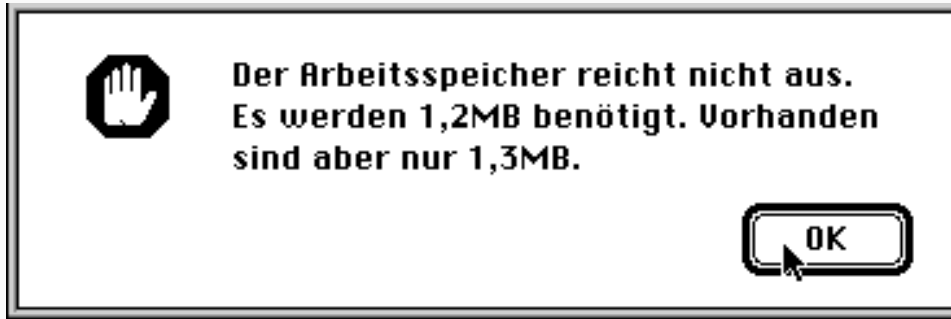
Denn Benutzer machen alle dummen Fehler die sie finden können!



# 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 (II)

---

- *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.

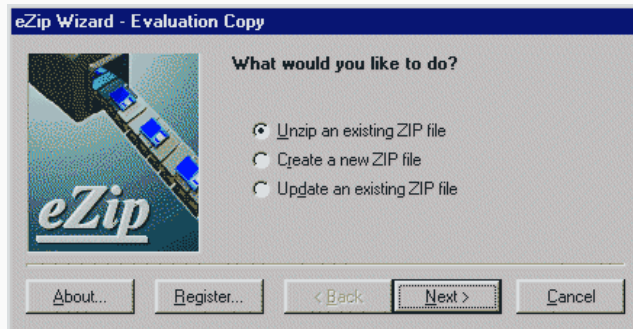
Generell: *Alternative Benutzeroperationen anbieten!*





# 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.

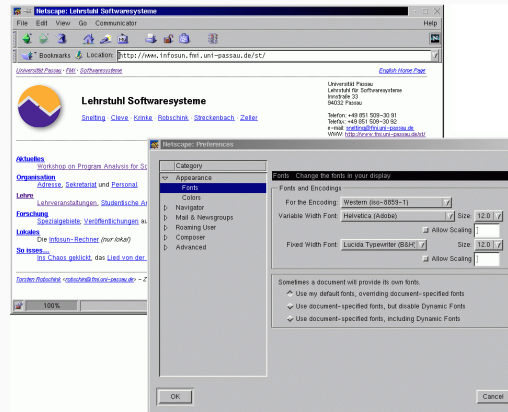






# Netscape Navigator

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.

The screenshot shows the DDD interface with a menu bar (File, Edit, View, Program, Commands, Status, Source, Data, Help) and a toolbar. The main window displays a graph of memory objects:

- 1: list (List \*) 0x804aee8
- 2: \*list value = 85 self = 0x804aee8 next = 0x804aef8
- 3: \*list->next value = 86 self = 0x804aef8 next = 0x804af08
- 4: \*list->next->next

Arrows indicate the pointers between these objects. Below the graph is a code editor showing the following code:

```
list->next->next = new List(a_global + start++);
list->next->next->next = list;
delete list->next->next;
delete list->next;
delete list;
}

// Test disambiguation
void l
{
  li
}

//
void r
{
  da
  da
}

//
```

A tooltip titled "DDD Tip of the Day #6" is displayed, containing a bug icon and the text: "There are three ways to show the value of a variable: • You can view its value, simply by pointing at it. • You can print its value in the debugger console, using Print (); • You can display it graphically, using Display ()."

At the bottom, the command `(gdb) graph display *(list->next->next->next) dependent on 4` is shown, along with the output `list->next->next = (List *) 0x804af08`.





## ***DDD: Breakpunkt 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!*

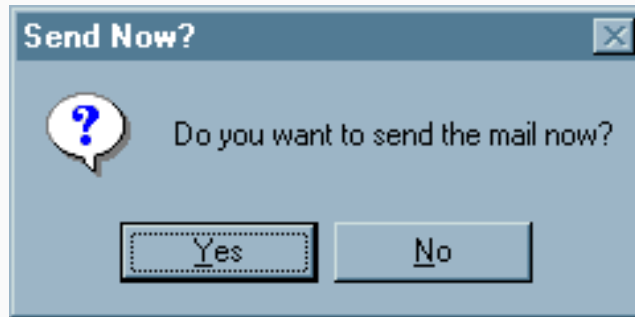




## Beispiele für mangelnde Effizienz

---

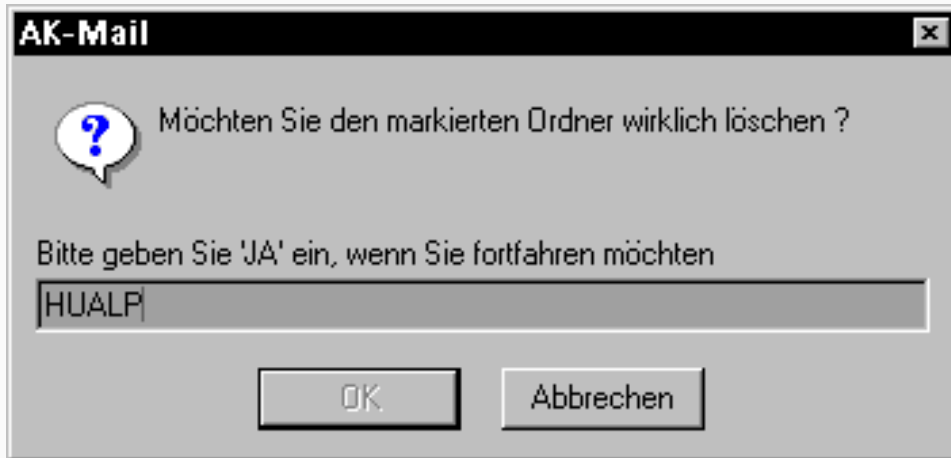
Jedesmal, wenn man in *Lotus Notes* eine e-mail absenden will, verlangt Notes eine Bestätigung. Das kann auf Dauer recht anstrengend werden:



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*:



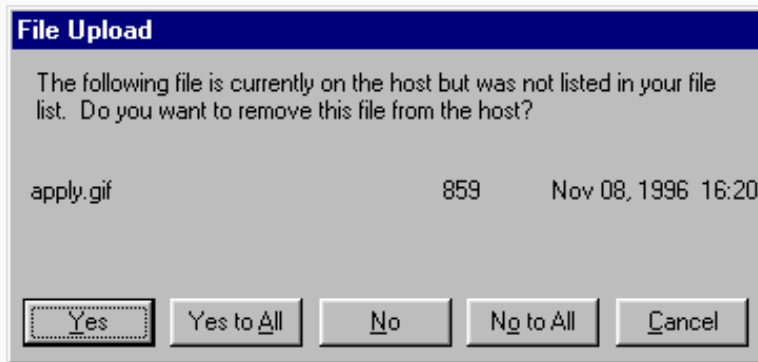
Warum eigentlich *JA*? Warum nicht gleich *JAAA, VERFLUCHT!?*







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:



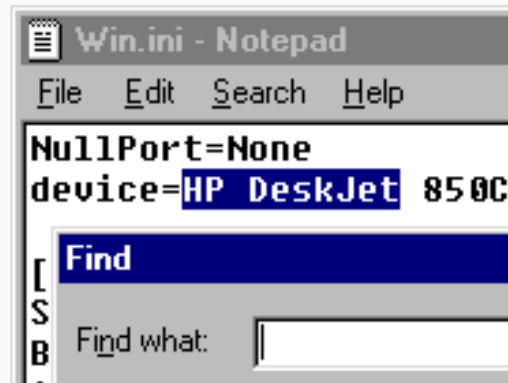
Wenn man etwa die Datei *zeller.gif* löschen wollte, müsste man zunächst 400mal auf *No* klicken – für jede der Dateien, die im Alphabet davor stehen.





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

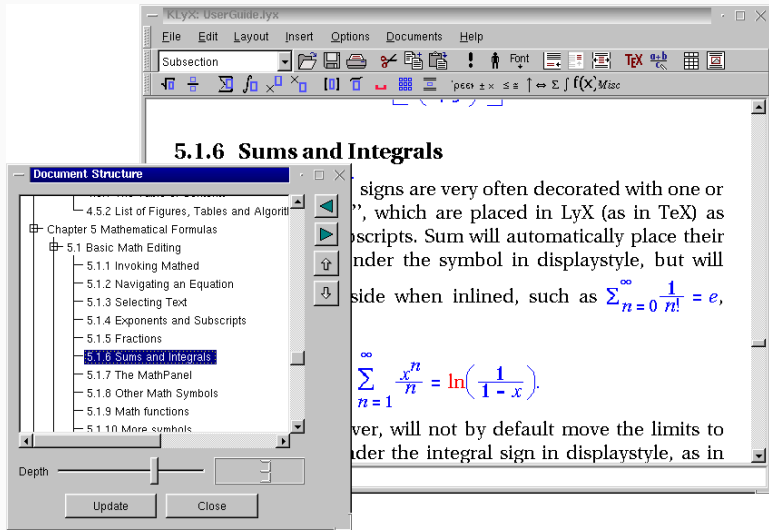
In anderen Programmen wie etwa *Notepad* muß der Benutzer stattdessen aufwendig den Text über die Zwischenablage kopieren und einfügen:





# Gute Effizienz

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



5.1.6 Sums and Integrals

signs are very often decorated with one or two primes, which are placed in LyX (as in TeX) as superscripts. Sum will automatically place their limits under the symbol in displaystyle, but will not do so on the side when inlined, such as  $\sum_{n=0}^{\infty} \frac{1}{n!} = e$ ,  $\sum_{n=1}^{\infty} \frac{x^n}{n} = \ln\left(\frac{1}{1-x}\right)$ . However, the sum command will not by default move the limits to the side when inlined, such as  $\sum_{n=0}^{\infty} \frac{1}{n!} = e$ , but will by default move the limits to the side when in displaystyle, as in  $\sum_{n=1}^{\infty} \frac{x^n}{n} = \ln\left(\frac{1}{1-x}\right)$ .





## ... mit der Tastatur

---

Der fortgeschrittene Benutzer kann aber auch direkt über die Tastatur  $\TeX$ -Kommandos eingeben - etwa

$$\backslash\text{sum}_{n = 1}^{\infty}\frac{x^n}{n} = \backslash\text{ln}\left(\frac{1}{1 - x}\right)$$

für

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

Bereits mit der ersten Eingabe von  $\backslash$  schaltet K $\text{L}\text{y}\text{X}$  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





# ***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)<sup>3</sup>



<sup>3</sup>Suzanna Smith et al., *GNOME Usability Study Report*,  
[http://developer.gnome.org/projects/gup/ut1\\_report/](http://developer.gnome.org/projects/gup/ut1_report/)







# *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 Kompetenz)
- 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?<sup>4</sup>*

<sup>4</sup><http://www.useit.com/alertbox/20010204.html>





# ***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.





## **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.

## **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.





# Grundprinzipien

---

- *Verwende Begriffe und Konzepte, die dem Benutzer vertraut sind*

Beispiele:

„Ablage“, „Papierkorb“, „Brief“, „öffnen“, „schließen“, „bearbeiten“  
etc. für ein Bürosystem

(und nicht „Datei“, „Datenbankrelation“, „Zugriffspfad“, . . .)

„Anweisung“, „Prozedur“, „Schleife“, „Variable“ in einem  
sprachspezifischen Editor

(und nicht „Syntaxbaumknoten“, „Environment“,  
„Continuation“)

- *Sei konsistent*

ähnliche Funktionen sollten ähnlich aussehen

Beispiele:

einheitliche Syntax für Kommandos

einheitliche Gestaltung von Fenstern, Scroll Bars, Menüs

einheitliche Mausknopfbelegung





Gegenbeispiel:

in einem System heißt es „quit“, im nächsten „end“, im dritten „bye“, im vierten „exit“, wobei im vierten System ebenfalls ein Kommando „quit“ existiert, das aber anders als im ersten die Daten nicht zurückschreibt

- *Vermeide Überraschungen*

ähnlich Aussehendes sollte ähnliche Funktionen auslösen

Gegenbeispiele:

„k“ steht in einem Subsystem für „keep transaction“, im nächsten für „kill transaction“

mittlerer Mausknopf steht in SunView für „Extend“, in X-Windows für „Paste“





- *Sei robust*  
z.B. durch

Bestätigung destruktiver Aktionen:

„Do you really want to delete this file?“

„undo“-Operation, die die letzte Operation vollständig rückgängig macht

Gegenbeispiel: UNIX-Kommando „rm“

„rm \*%“ vs. „rm \*“

- *Verwende Benutzerführung*  
z.B.

Menüsystem

Hilfesystem







# *Graphische Benutzerschnittstellen*

---

Charakteristische Merkmale:

- Mehrere Fenster erlauben simultane Darstellung verschiedener Prozesse
- „Icons“ zur Repräsentation von Objekten oder Prozessen
- Funktionsauswahl mit Menüs
- Maus zur Auswahl interessierender Objekte
- Sowohl textuelle als auch graphische Ausgabe

Vorteile:

- leicht zu erlernen  
man muß nicht 750 Kommandos auswendig lernen, sondern bekommt in jeder Situation ein Menü mit anwendbaren Optionen
- Wechsel von einer Aufgabe zur andern ohne Informationsverlust





z.B. Editieren in einem Fenster, Übersetzen in einem anderen Fenster

- schnelle Interaktion  
wer einmal mit einem zeilenorientierten Editor gearbeitet hat, weiß , was ein emacs mit Maus oder ein axe wert sind
- direkte Manipulation  
z.B. im Texteditor: setzen des Cursors mit der Maus; tippen; das getippte ist sofort zu sehen  
oder im Dateisystem: Auswahl einer Datei mit der Maus;  
Auswahl einer Operation aus einem Menü  
  
Benutzer haben Kontrolle über das was passiert und werden nicht eingeschüchtert  
sofortige Rückkopplung erlaubt sofortige Fehlerkorrektur





## *Schnittstellenmodelle*

Die Benutzerschnittstelle sollte sich analog zu einem dem Benutzer bekannten realen System verhalten

Paradebeispiel: „Desktop“ Metapher

Der Schreibtisch dient als *Modell* für das Verhalten der Schnittstelle

anderes Beispiel: Flugsimulator. Wehe, der sieht nicht so aus wie die Armaturen im echten Flieger!

Der Flieger ist Modell für den Simulator

Schnittstellen werden oft aus *Widgets* zusammengesetzt, die Konzepten in realen Systemen nachempfunden sind

Beispiele im OSF/Motif Toolkit: „Armaturenbrett“ als Modell

*Buttons* (beschriftete Knöpfe). Anklicken löst eine bestimmte





## Aktion aus

*Switches* (Schalter). Auswahl aus einer Menge von Alternativen. Die jeweils nächste wird durch Anklicken angewählt

*Menüs*. Kollektion von Knöpfen zu Auswahl aus einer Menge von Alternativen. Verschiedene Realisierungsmöglichkeiten (s.u.)

*Lights* (Kontrollampen). Zeigen, daß bestimmte Aktionen stattfinden

*Displays* (Anzeige). Dient zum Ausgeben/Manipulieren von Text oder Graphik

*Sliders* (Schieberegler). Zum Einstellen auf einer kontinuierlichen Skala

...





## *Menüsysteme*

Vorteile von Menüsystemen:

- kein Auswendiglernen von Kommandonamen
- wenig Tipparbeit
- Fehlerverhinderung: nur zulässige Aktionen sind auswählbar
- Benutzerführung: das System kennt den jeweiligen Kontext des Benutzers und kann entsprechende Hilfe anbieten

Nachteile:

- textuell kurze, aber strukturell komplexe Eingaben problematisch (Standardbeispiel: Eingabe von arithmetischen Ausdrücken in einem Struktureditor)





- Erfahrene Benutzer werden behindert. Deshalb: Funktionstasten als Alternative
- Bei großer Zahl von Alternativen Strukturierung erforderlich:
  - automatisches Scrollen der Items
  - „Walking menus“: bei Auswahl eines Items erscheint ein Untermenü
  - hierarchische Menüs: Auswahl eines Items führt zur Anzeige eines ganz neuen Menüs (evtl. Extrafenster zur Anzeige der Position im „Menübaum“)





# *Gestaltung von Displays*

---

**Anzeige von Text** Stelle statische Information anders dar als dynamische (invertiert, anderer Font, ...)

Aber: nicht mehr als ca. 4 verschiedene Fonts!

Wichtige Information sollte durch graphische Elemente hervorgehoben werden

*Im übrigen gelten die Grundsätze der Typographie*

**Anzeige von numerischen Werten** falls exakte numerische Werte verlangt sind, müssen diese textuell ausgegeben werden

falls Tendenzen (1. Ableitung) wichtig sind, Darstellung als Kurve oder Histogramm

Standardbeispiel: Verkaufszahlen

falls relative Werte wichtig sind, Darstellung als Tortendiagramm,



Thermometerskala o.ä.  
Standardbeispiel: Wahlergebnisse

Die Ergebnisse komplexer numerischer Simulationsmodelle sollten  
als dreidimensionale Graphik dargestellt werden



64/68







# *Verwendung von Farbe*

---

Farbe sollte man nur verwenden, wenn die Informationsflut anders nicht zu bewältigen ist  
z.B. VLSI-Entwurfssystem

Ein Texteditor mit 17 bunt blinkenden Anzeigen ist dagegen Overkill

Grundsatz: *Farbe muß sparsam verwendet werden*

Denn:

- etliche Männer sind farbenblind
- Farbempfinden ist individuell stark schwankend
- Farben sind stark emotional besetzt
- Farben sind schwach mit rationalen Konzepten besetzt
- ständige Farbschocks halten den Adrenalinpegel hoch, was die Lebenserwartung deutlich verkürzt



Deshalb:

- nicht mehr als 4 Farben pro Fenster, nicht mehr als 7 Farben insgesamt
- Farbe sollte entweder Anomalien oder Gemeinsamkeiten darstellen
- Farbe sollte abschaltbar/änderbar sein
- Zuerst sollte man eine gute monochrome Schnittstelle gestalten
- Farben müssen konsistent verwendet werden. Wenn rot für Fehlermeldungen verwendet wird, müssen das *alle* Systemteile tun
- Manche Farbkombinationen sind physiologisch schlecht; z.B. kann das Auge nicht gleichzeitig auf blau und rot fokussieren (→ Kopfweg)
- Verwende Farbänderungen, um *signifikante* Änderungen im Systemstatus anzuzeigen
- Wenn man die Wahl hat zwischen hochauflösendem Schwarzweiß und schlechtauflösender Farbe, nimm Schwarzweiß



66/68





# Entwurf von Systemmeldungen

---

Grundsätze für Fehlermeldungen und Hilfetexte:

- Meldungen müssen den momentanen Benutzerkontext berücksichtigen
- Meldungen müssen die Erfahrung des Benutzers berücksichtigen (lange Texte für Anfänger, kurze Texte für erfahrene Benutzer, Hexcodes für Hacker)
- Meldungen müssen die Fähigkeiten des Benutzers berücksichtigen (z.B. verschiedene Terminologie für Sekretärinnen und Programmierer)
- Meldungen müssen positiv sein. Keine Anklagen! Keine Beleidigungen! Keine Witze! Keine merkwürdigen Geräusche auf Workstations mit Lautsprecher!
- Fehlermeldungen müssen konstruktiv sein; sie sollten sagen, wie man den Fehler verbessern kann





- Stets sollte das System insinuieren, daß es zu dumm ist, den Benutzer zu verstehen, und nicht, daß der Benutzer zu dumm ist, das System zu bedienen
- Hilfesysteme sollten vielfache Einstiegsmöglichkeiten bieten (z.B. zu jedem Knopf gibt es eine „Help“ Option)
- Hilfesysteme sollten Querverweise bieten (→ *Hypertext*)
- Hilfstexte sollten nicht zu lang sein. Schließlich gibt es noch die gedruckte Dokumentation!

