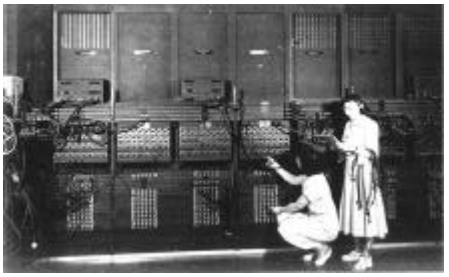
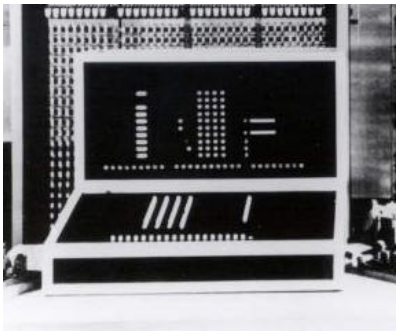
GUI Programmierung

Teil 1
Kurzgeschichte des Benutzerinterfaces

Geschichte: Die ersten Rechenmaschinen



ENIAC, 1946



Zuse Z3,1941

Geschichte: Computer zweiter Generation

Eingabe: Lochkartenleser

Ausgabe: Drucker

1955-1965

Geschichte: Die ersten Terminals

IBM 2741, 1975

Amdahl 470/U6,1977

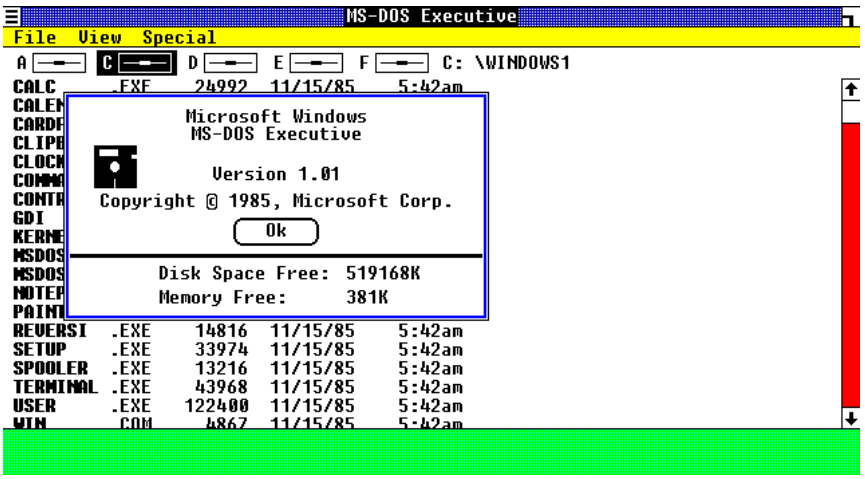
Geschichte: Das Textterminal

Commodore 64, 1982

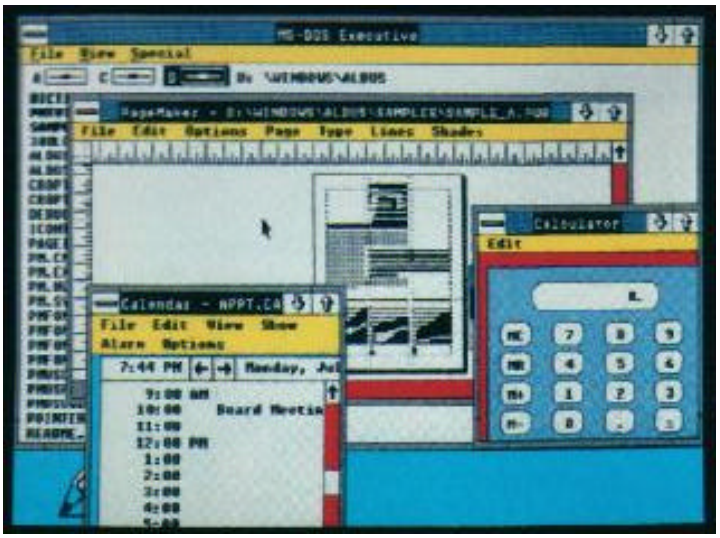
IBM SAA Standard, 1987

Hier: Turbo Pascal

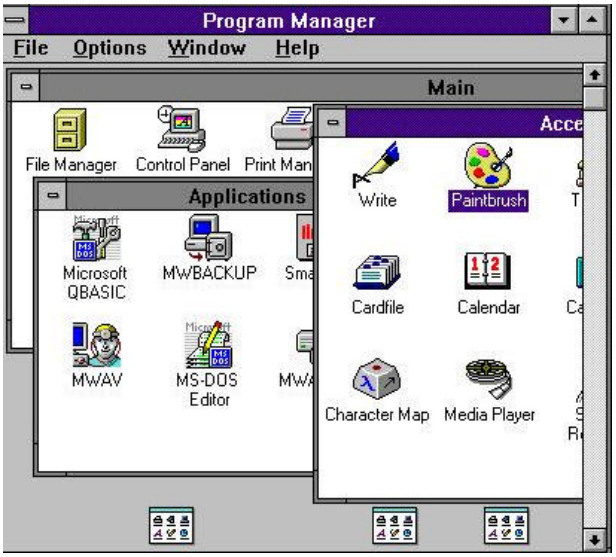
Geschichte: Windows 1.0 (1985)



Geschichte Windows 2.0 (1987)



Geschichte: Windows 3.0 (1990), 3.1 (1993)



Geschichte: Windows 9x, XP

Windows 95: 1995
Windows 98: 1998
Windows ME: 2000

Windows NT: Abkömmling von OS/2
Windows 2000: 2000
Windows XP: 2001

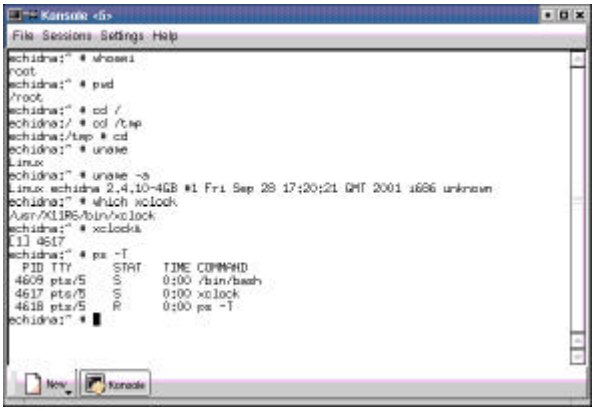
Teil 2
Grundlagen

Benutzbarkeit (Usability)

Usability = Effectiveness + Efficiency + Satisfaction

(Dix et al.)

Die Kommandozeile



- Historisch entstanden
- Schnell zu bedienen
- Leicht zu automatisieren

Die Kommandozeile

- Hauptvorteil:
  Maximal große “Eingabedomain” => Mächtig, maximaler Freiheitsgrad.

- Hauptnachteil:
  Maximal große “Eingabedomain” => Nicht ohne erhebliches Vorwissen bedienbar.

=> Problem: Freiheitsgrade einschränken, aber universelle Bedienbarkeit behalten.

Die GUI

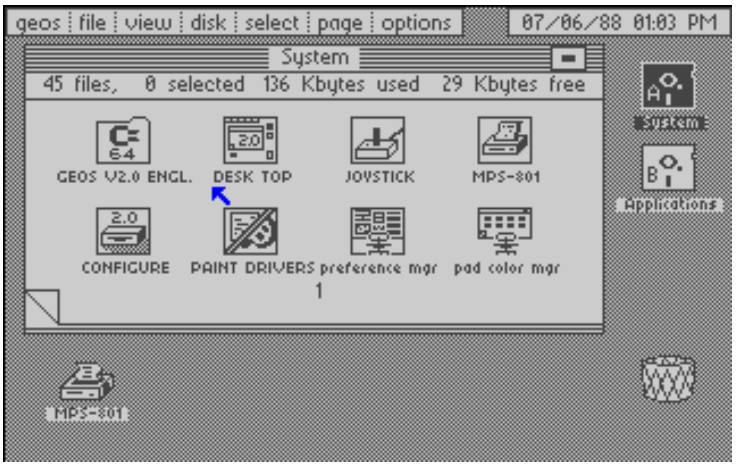
Erster kommerzieller Rechner mit GUI: Apple Lisa, 1983.

Vorher Forschung am Xerox Parc (z.B. Erfindung der Maus durch Douglas Engelbart 1968).

Entscheidende Idee: Die Desktop-Metapher

Die Desktop Metapher

Grundidee: Der Computer erweitert den Schreibtisch



Nichtanwendbarkeit der Desktopmetapher

- Die Desktopmetapher hilft die Zeit zu minimieren, die man braucht, einen Brief zu schreiben, zu drucken, zu speichern und zu löschen. Aber ist sie z.B. für einen Netzwerkadministrator hilfreich?

- Sehr wichtiges Anwendungssystem für das die Desktopmetapher im Allgemeinen ungeignet ist: Computerspiele.

- Wahl der Metapher hängt von der Hardware ab: PDAs, Integrierte Systeme, Wandtafeln (E-Chalk), etc... brauchen eine andere Metapher.

Aktuelle Forschungsdisziplin: Human Computer Interface Design (HCI)

GUI Standards

- Microsoft: The Windows Interface Guidelines for Software Design: An Application Design Guide, Microsoft Press, Redmond, WA, 1995.

- Apple: Macintosh Human Interface Guidelines, Addison-Wesley, Reading, MA, 1993.

- Apple: Aqua Human Interface Guidelines, Apple Inc, Cupertino, CA, 2001.

- Sun: Java Look and Feel Design Guidelines, second edition, Boston, MA, 2001 und Java Look and Feel Design Guidelines: Advanced Topics, Addison-Wesley, Boston, MA, 2002.

Teil 3
Designprinzipien

Prinzip 0

Do not think „inside-out“

- Häufigstes Problem nicht nur bei der Entwicklung von Software: „Betriebsblindheit“

Andere Beispiele:

- Vorlesungen

- Aufsätze (vor allem technische Beschreibungen)

- Soziale Phänomene („von sich auf andere schliessen“, „sich in eine Lage nicht hineinversetzen können“)

Prinzip 1

Orientiere Dich nicht an dem, was durch die Technik vorgegeben wird, sondern konzentriere Dich auf die Benutzer und Ihre Aufgaben

- Welche Fähigkeiten hat der Benutzer?
- Bei welchen Aufgaben soll ihm das Programm helfen?
- Wie kann das Programm die wichtigsten Aufgaben des Benutzers schnell erledigen und die selteneren überhaupt?

„Common things should be easy, advanced things should be at least possible“ (Larry Wall)

Prinzip 2

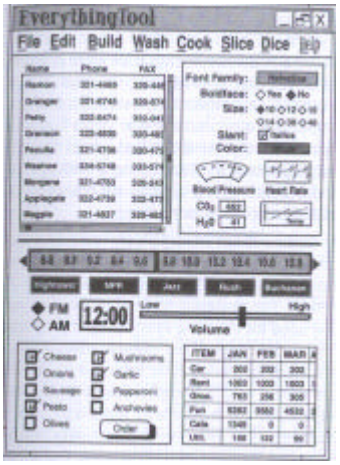
Denke erst an die Funktion, dann an die Präsentation

- Weniger ist mehr
- Halte Dich an Standards, sei nicht an der falschen Stelle kreativ.
- Halte es so einfach wie möglich, aber nicht zu einfach (Negativbeispiel: mv in Unix)
- Benutze vorgegebene GUI Komponenten nur so, wie sie gedacht sind

Prinzip 3

Denke daran, wie der Benutzer seine Aufgaben sieht

- Lass Deinen Benutzer keine unnatürlichen Handlungen ausführen (Beispiel: Schachprogramm)
- Führe keine willkürlichen Beschränkungen ein (typischer Fehler: konstante Arraylängen)
- Benutze das Vokabular des Benutzers
- Behalte Programminterna im Programminnern
- Finde den richtigen Kompromiss zwischen Mächtigkeit und Komplexität des Programms



Prinzip 4

Fördere den Lernprozess des Benutzers

- Strukturiere logisch, vermeide Widersprüche (gutes Beispiel: Unixphilosophie)
- Sei konsistent (Vorsicht: Konsistenz bringt auch Probleme mit sich)
- Schaffe eine risikolose Umgebung (Beispiel: Undo, Redo)
- Gib intelligente Standardwerte vor

Prinzip 5

Vermittle Inhalt, nicht nur Daten

- Dein Programm muss sich sinnvoll präsentieren: Es muss zu jeder Zeit klar sein, was der nächste Schritt ist.

- Das Programm muss dem Anzeigemedium angepasst sein (Beispiel: 1280x1024 auf einem PDA?)

- Bildschirmmeldungen müssen erklärend, verständlich und angebracht sein (Negativbeispiel: Geldautomat).

- Der Bildschirm gehört dem Benutzer (Negativbeispiel: Officeassistenten)

Prinzip 6

Garantiere die Ansprechbarkeit des Programms

- Das Programm sollte möglichst immer ansprechbar sein. (Negativbeispiel: Internet Explorer)

- Wenn das Programm ausnahmsweise nicht ansprechbar ist, sollte der Benutzer darüber informiert werden, was das Programm gerade tut und wie lange noch. (Beispiel: Busy Cursor, Progress Bar)

- Alle längeren Aufgaben sollten grundsätzlich abbrechbar sein.

Responsiveness = Perceived Performance

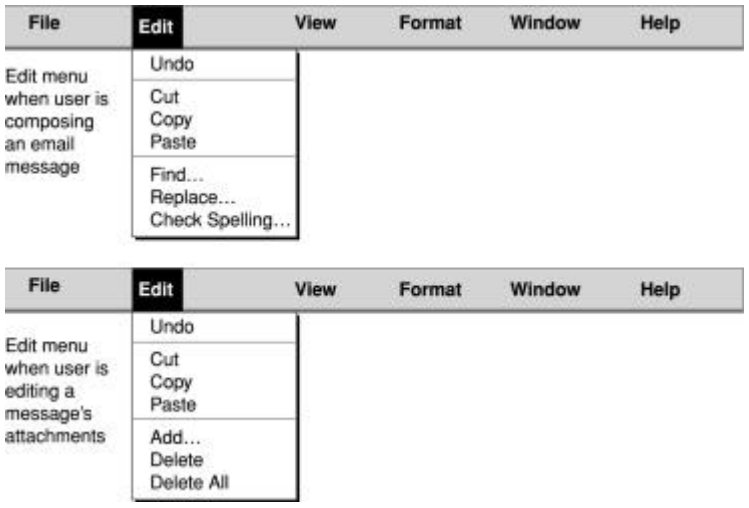
Prinzip 7

Probiere Dein Programm an Benutzern aus

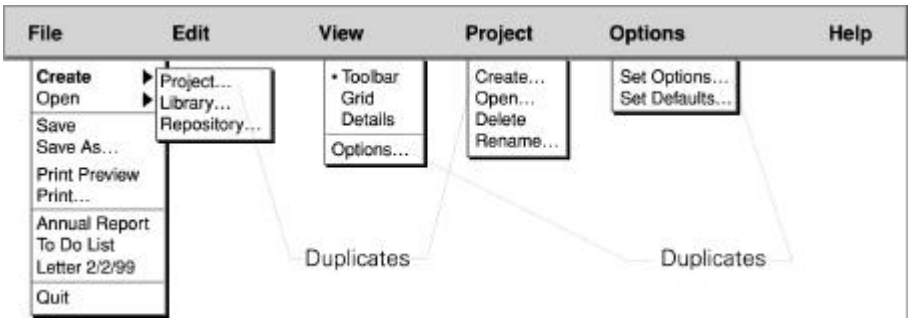
- Zwei Ziele: Sozial, Inhaltlich

Teil 4
Exemplarische Designfehler

Schlecht: Variable Menüeinträge

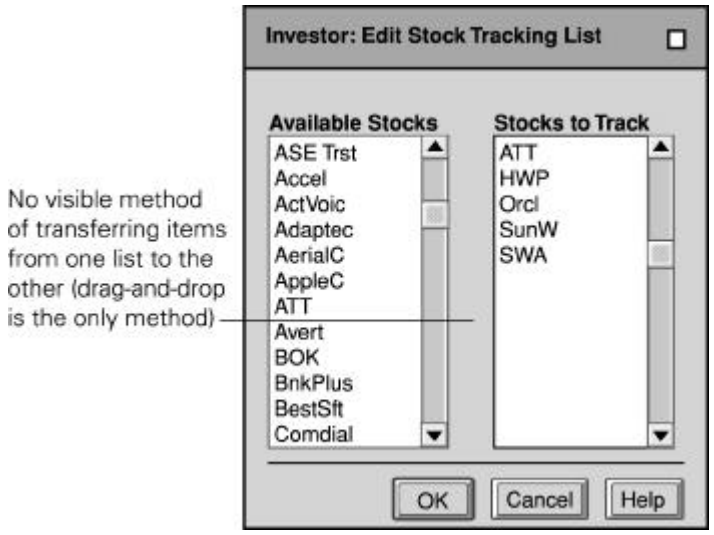


Schlecht: Doppelte Menüeintrage

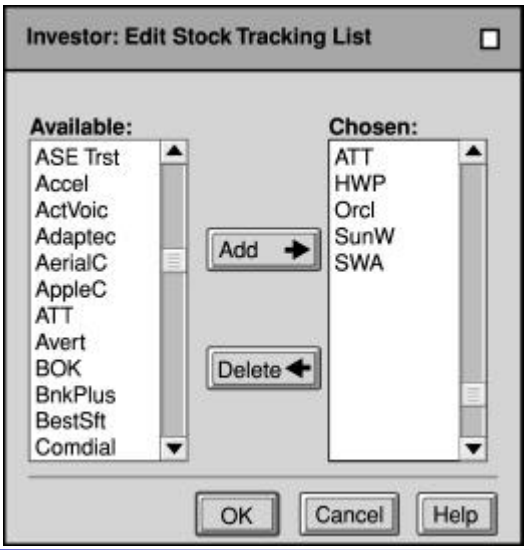


Ausserdem schlecht: Keine Tastaturunterstützung

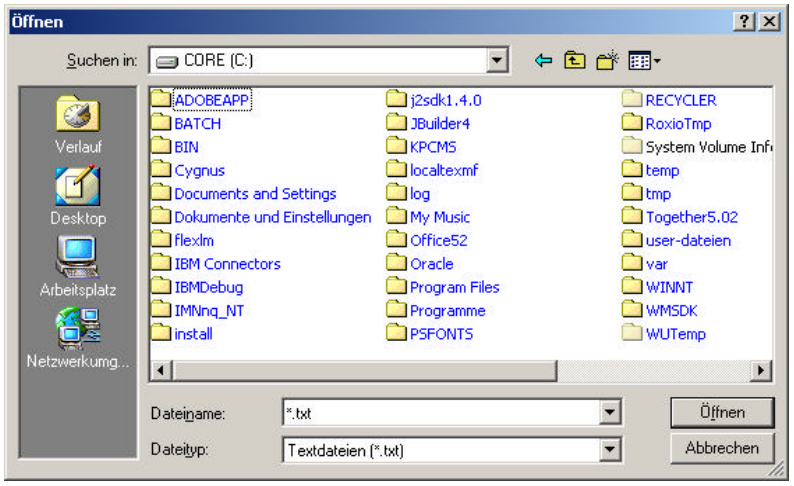
Schlecht: Handlungen implizit verlangen



Besser: Handlungen explizit vorgeben

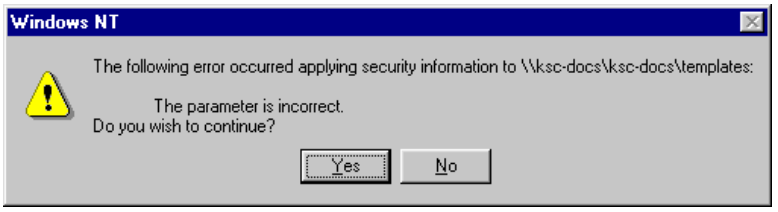


Schlecht: Unintuitive Anordnung

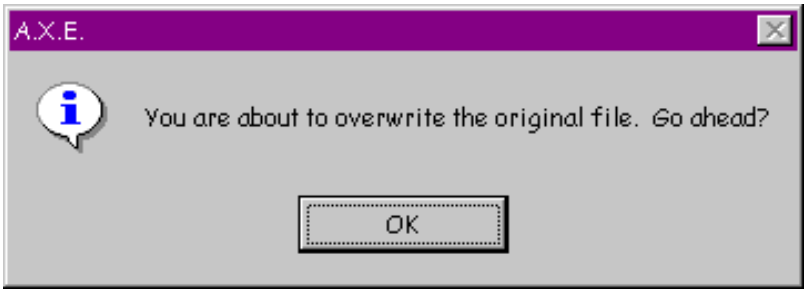


Schlecht: Entscheidungen verlangen, ohne hinreichende Hinweise

Schlecht: Entscheidungen verlangen, ohne hinreichende Hinweise (2)



Schlecht: Information geben, aber keine Wahl lassen

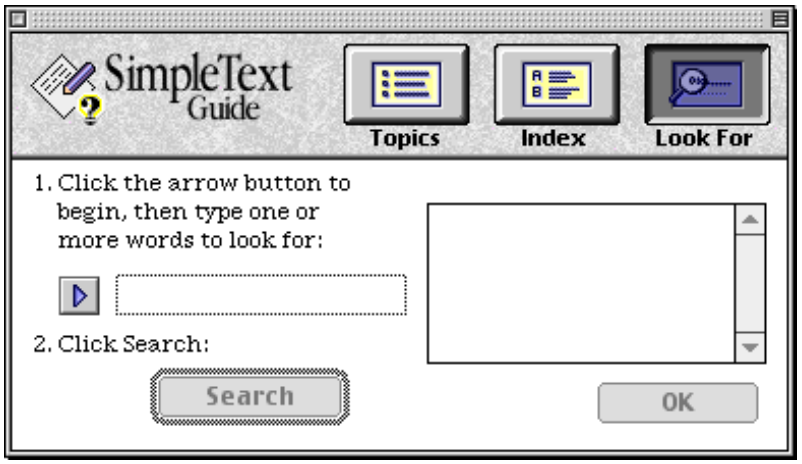


Schlecht: Dialoge mit Nullaussage

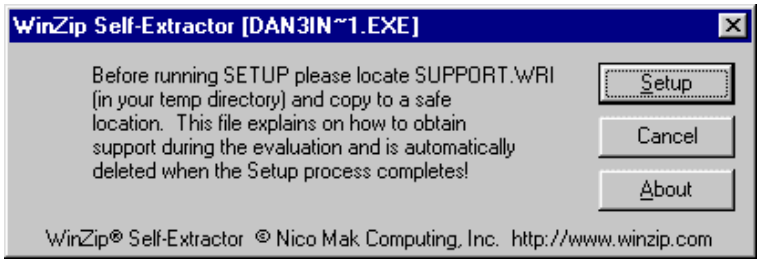
Schlecht: Künstlich Stress erzeugen

Noch schlechter: Künstlich Stress erzeugen und die Zeit des Benutzers verschwenden (Endlosschleife)

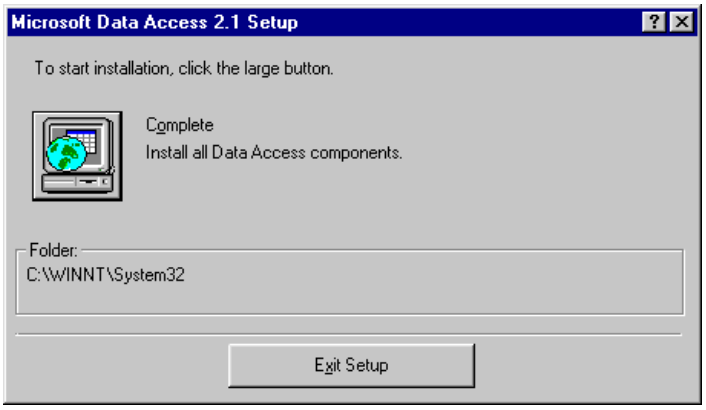
Schlecht: GUI Elemente verkomplizieren



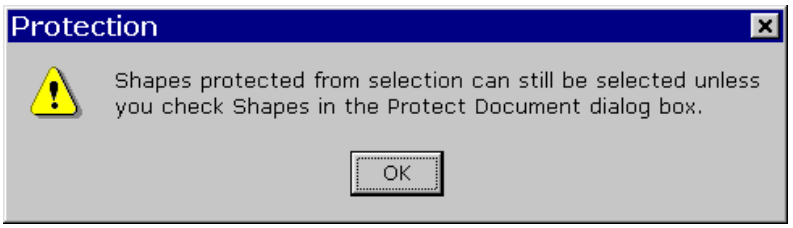
Schlecht: Aufgaben an Benutzer verteilen



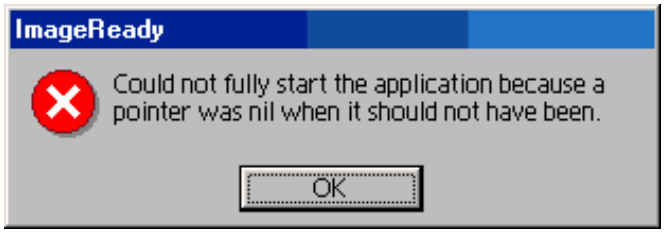
Noch schlechter: Unerfüllbare Aufgaben verteilen



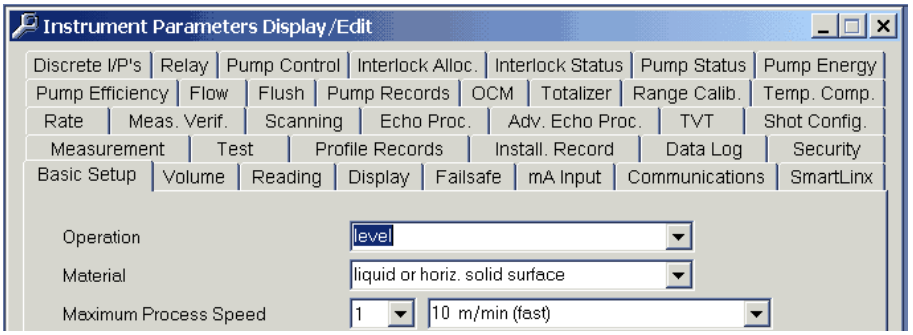
Schlecht: Komplizierte Formulierungen



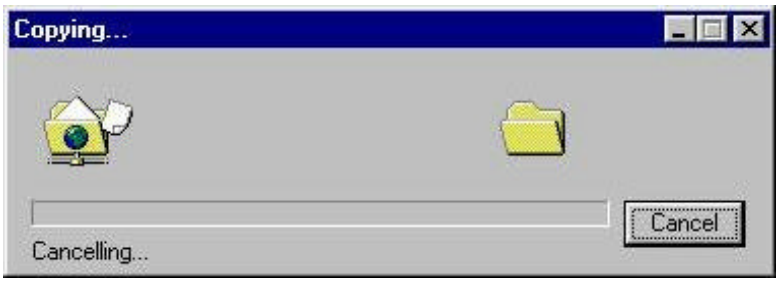
Schlecht: Geek Speak



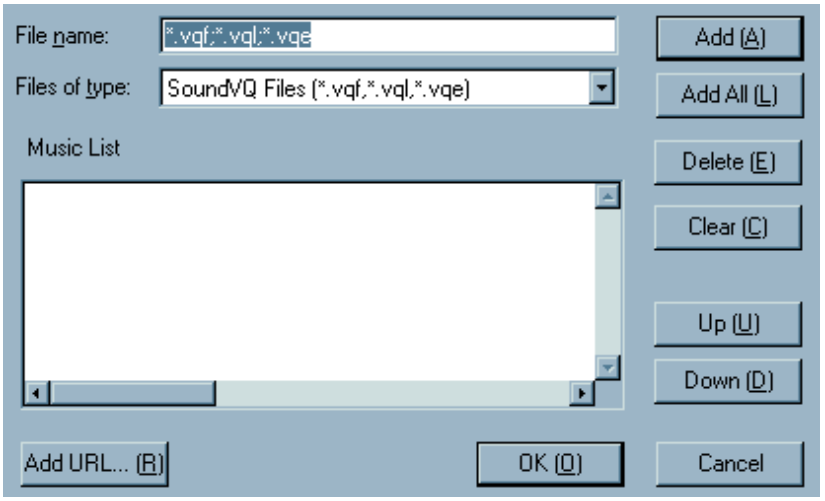
Schlecht: Benutzer mit Komplexität erschlagen



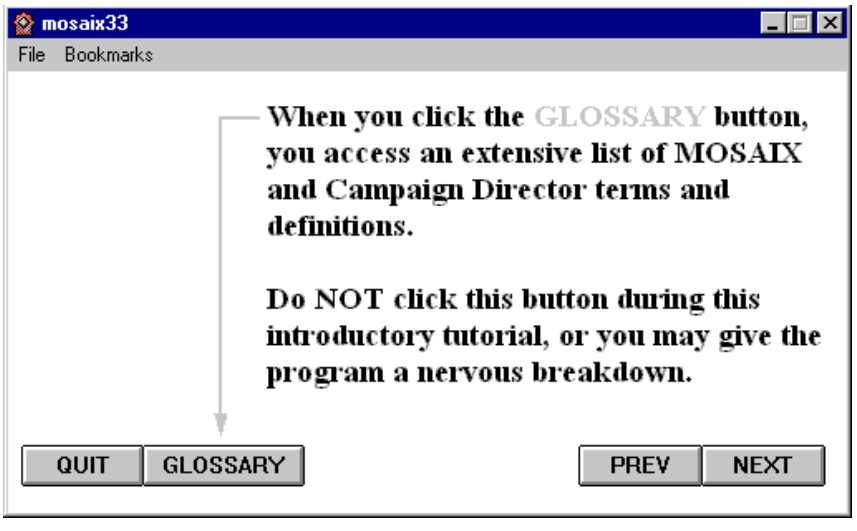
Schlecht: Programm nicht wirklich ansprechbar



Schlecht: Kreativität an der falschen Stelle



Schlecht: Die wichtigsten Features der GUI nicht zu kennen



Literatur

- Jeff Johnson, „GUI Bloopers“, Morgan Kaufmann, 2000.
- Jakob Nielsen, „Designing Web Usability“, New Riders, 2000.
- Jakob Nielsen, „Usability Engineering“, Academic Press, San Diego, 1993.
- Alan Dix, Janet Finlay, Gregory Abowd, „Human Computer Interaction“, Prentince Hall, 1998.
- Jeff Raskin, „The Humane Interface“, Addison-Wesley 2000.
- Donald R. Norman, „The Design of Everyday Things“, Currency/Doubleday, 1990.