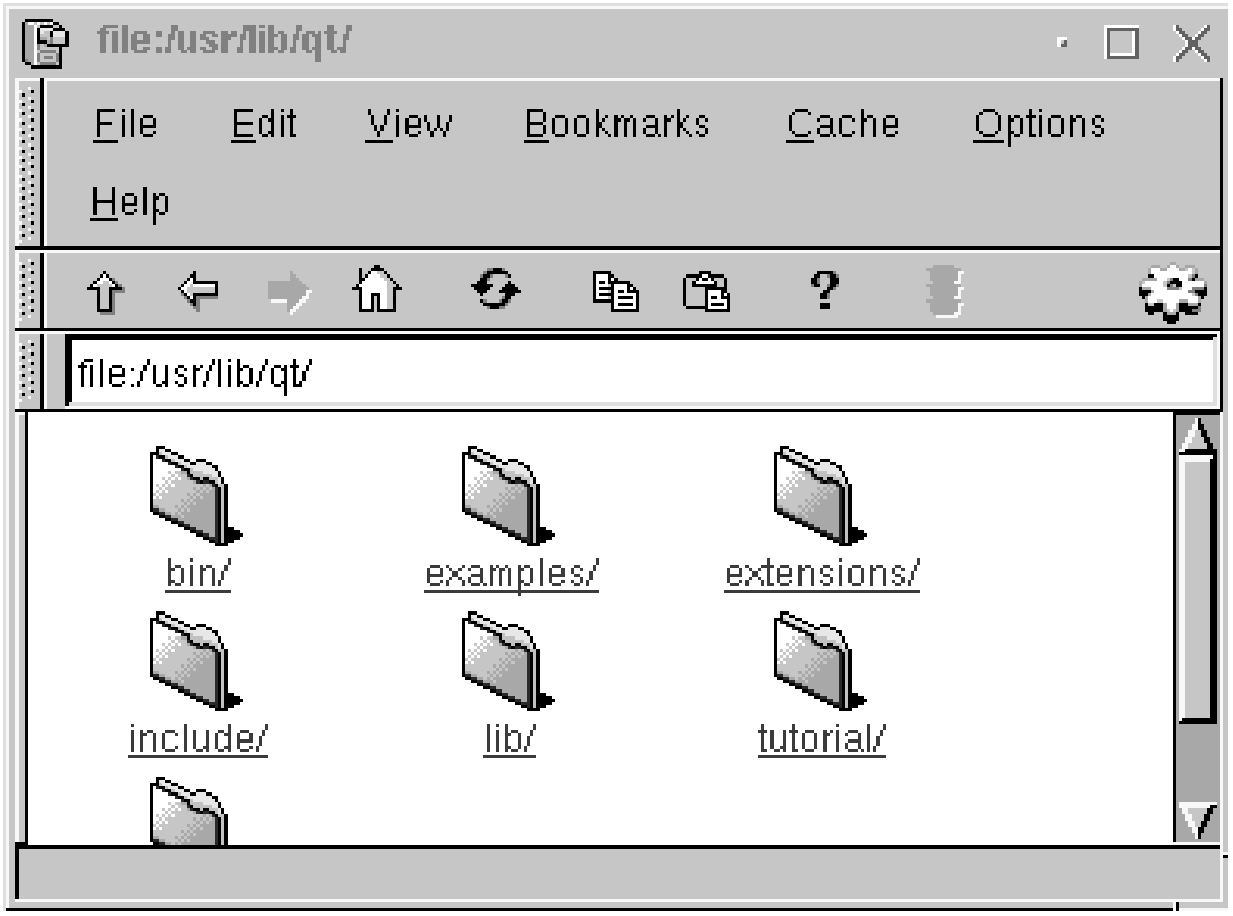
Fenstersysteme

Wozu Fenster-Systeme ?

- Gleichzeitige Darstellung von Ausgaben mehrerer Programme
- Graphische Ausgaben
- Einfache Kommunikation zwischen Benutzer und Programm (Menüs, Formulare ...)
- Quasi parallele Eingabe möglich
- Programmstatus leichter erkennbar
- Direkt manipulativ

Beispiel: KDE-Dateimanager



aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

Interaktionsmöglichkeiten des Beispiels (unvollständig):

- Schließen, Vergrößern und Iconifizieren des Fensters durch Icons
- Bewegen des Fensters durch Ziehen der Titelleiste
- Vergrößern/Verkleinern des Fensters
- Verschieben des Fensterinhaltes mittels Scrollbar
- Aktivieren des Menüs des Fenstermanagers mittels Icon
- Aktivieren der Menü-Spalten der Anwendung
- “Drag & Drop” der Dateiordner-Icons
- Aktivieren der Toolbar-Buttons
- Texteingabe

Geschichte der Fenster-Systeme

- Mitte der 70er Paradigma für Fenster-Systeme Xerox PARC
- Erstes System Xerox Alto (1972), später Xerox Star (1981)
- Apple Lisa (1983) und Apple Macintosh (1984)
- X-Windows System (1984)
- weitere Fenster-Systeme, wie MS Windows, NeWS, NeXTSTEP

Fenstermanagement

$\rightarrow$ bietet Software-Unterstützung für Applikationen

Das heißt:

- Generieren und Organisation der Fenster
- Grundlagen der interaktiven Komponenten
  (interaktive Objekte $Widgets$, z.B. Buttons, Scrollbars)

außerdem:

eine Benutzerschnittstelle

$\rightarrow$ Kontrolle über Größe und Plazierung der Fenster

Organisation der Fenster und Interaktionsobjekte

- Verwalten der Fenster mittels eines Fenster-Baums
- Hierarchie ergibt sich aus der Bildschirm-Geometrie (in der Regel rechteckige Aufteilung)

$\Rightarrow$ einfaches und schnelles Modell für die Aufteilung des Bildschirms und für die Ereignis-Verwaltung (event handling)

Beispiel:

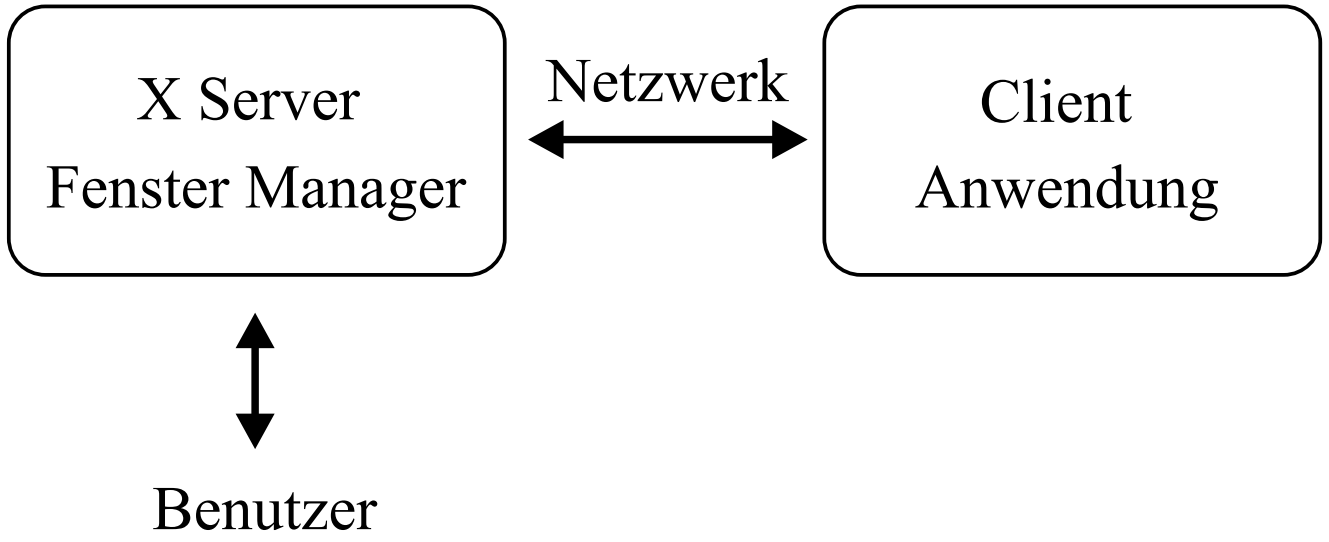
Bildschirm
- Filemanager window
  - - Title bar
    - - Menu icon window
    - - Icon sticky
    - - Title bar
    - - Icon iconify
    ...
  - - Menubar
    – Popup menu file
    ...
  - - Toolbar
    ...

Fenster-Systeme

X Windows

→ Client/Server Architektur

→ hohe Flexibilität, aber hohe Netzauslastung



aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

NeWS

- PostScript anstelle des X-Protokolls
- PostScript basiert auf FORTH
- Einige Benutzeraktionen können beim Server ausgeführt werden
  $\longrightarrow$ weniger Netzauslastung

Java

- komplette Anwendungen können auf dem Server laufen (Applets)

MsWindows

- alles liegt auf einer Maschine

Ereignisse (window events)

$\longrightarrow$ Fensterbasierte Applikationen können eine Vielzahl verschiedener Eingaben (messages, events) registrieren

Input events:

- Mouse button events (button down, button up, button move ...)
- Mouse enter und mouse exit events
- Keyboard events (key down, key up)

Windowing events:

- Create, destroy events, Iconify, deiconify events
- Raise, lower, resize, move events

Redraw event

Ereignis-Verarbeitung (event handling)

Hauptprogramm (main event loop)

```
Initialization                  /* define windows, set up things ... */
while (not time to quit)
{
    Get next event E            /* from queue */
    Dispatch event E
}
```

Filtern der Ereignisse vereinfacht die Abarbeitung

```
Initialization                  /* define windows, set up things ... */
while (not time to quit)
{
    Get next event E            /* from queue */
    if (not FilteredEvent(E))   /* interested in this event ? */
        Dispatch event E
```

Beispiel (Event handling Xlib):

```
Window window = XCreateWindow(display, ...);
XMapWindow(display, window);
app -> running = true;
while (app -> running)
{  XEvent event;
    XNextEvent(display, &event);        /* get next event */
    switch(event.type)                  /* filter and dispatch */
    {
        case Expose:                    /* redraw event */
            ...                         /* draw window content */
            break;
        case ButtonPress:               /* mouse button down event */
            app -> running = false;     /* quit application */
            break;
        default:
            break;
```

Ereignis-Abarbeitung (dispatching)

Welchem Fenster bzw. Interaktionsobjekt wird Ereignis gemeldet?

Zwei Arten der Entscheidung (anhand Fensterbaum):

- **Bottom-up:**
  das vorderste Fenster in der dargestellten Fenster-Hierarchie das im Fenster-Baum an unterster Stelle ist, erhält das Ereignis
- **Top-down:**
  das voreste Fenster in der dargestellten Fenster-Hierarchie das im Fenster-Baum an oberster Stelle ist, erhält das Ereignis

andere Möglichkeit der Zuordnung:

- **Click-to-focus:**
  bestimmtes Fenster bekommt den Einagbe-Focus zugewiesen und damit das Ereignis

$\longrightarrow$ Unterschiede können hierbei in der Handhabung von Maus- und Keyboard-Ereignissen bestehen

Programmiermodelle

- Low-level (siehe Xlib Beispiel)
- Xt/Motif Callback Modell
- WindowProc-Based (MS-Windows)
- Signal/Slot Modell (Qt)

$\rightarrow$ die Modelle bieten neben der unterschiedlichen Ereignis/ Fensterzuordnung dem Programmierer Hilfe zum Layout der Fenster

Callback-Funktionen (Xt/Motif)

→ einem Widget wird eine Funktion (Callback) zugeordnet,
die bei einer bestimmten Interaktion aufgerufen wird

```
void Callback(Widget widget, caddr_t data, caddr_t motifdata)
{
...                                          /* do something if button pushed */
}

Widget button = XmCreatePushButton(parent, ...)
XtManageChild(button)                        /* create and show button */
XtAddCallback(button,                        /* connect callback */
        XmNactivateCallback, Callback, data);
XtMainLoop();                                /* jump in main loop */
```

WindowProc (MS Windows)

→ jedes Fenster besitzt eine WindowProc Funktion, die jedes Event erhält, eine Selektion der gewünschten Ereignisse findet dort statt

WindowProc Funktion:

```
WndProc(HWND Window, UINT MsgID, ...)
{ switch(MsgID)
    {
        case WM_PAINT:
        ...                 /* redraw */
        case WM_LBUTTONDOWN:
        ... }
}
```

```
int WindowMain(...)
{ ...                          /* initialize */
    HWND window = CreateWindow(...);
    ShowWindow(window)
    MSG Message;
    while (GetMessage(&Message, ...))
        DispatchMessage(&Message)
}
```

Signal/Slots (Qt)

→ jedes Widget (definiert durch Klassen) kann Signals und Slots (Klassenfunktionen) besitzen, die verbunden werden können

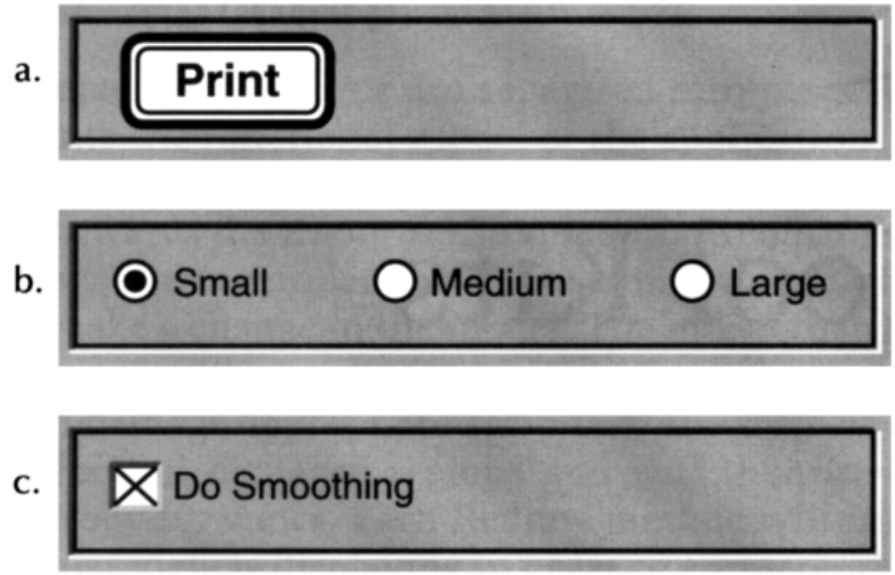
→ wird Signal verschickt, werden die angebundenen Slot-Funktionen aufgerufen

```
int main(int argc, char **argv)
{
    QApplication a(argc, argv);          // init application
    QPushButton quit("Quit");            // create new button
    quit.resize(75, 30);
    QObject::connect(&quit,              // connect: if button sends signal clicked,
            SIGNAL(clicked()),
            &a, SLOT(quit()));           // application function quit is called
```

```
a.setMainWidget(&quit);
quit.show();                    // show button on screen
return a.exec();                // start main loop
```

Dialogbausteine

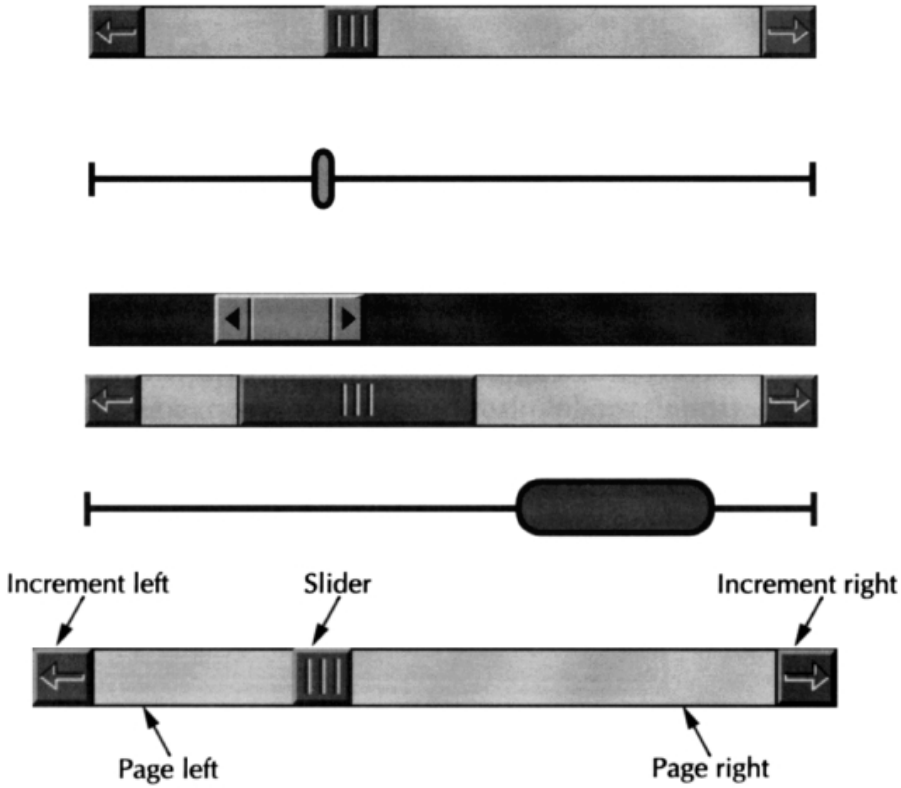
- Button
  $\longrightarrow$ eine Variable wird auf einen oder mehrere diskrete Werte gesetzt



aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

- Slider

  $\longrightarrow$ Variable kann auf kontinuierliche Werte gesetzt werden



aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

- Menü

  → Auswahl aus vielen Möglichkeiten

aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

- Textfeld (Graphikfeld)
  → Text / Graphikeingabe

auch hier:

Unabhängigkeit von Funktionalität und Aussehen

Look and Feel:

$\longrightarrow$ durch geeignetes Programmieren (Programmbibliotheken) können verschiedene visuelle Repräsentationen eines Benutzerdialogs erzeugt werden

Beispiel: QT bildet auch auf Motif und Windows ab

Abstrakte Geräte

→ fassen verschiedene Eingabegeräte bzw. -dialoge zusammen
(→ Standardisierung)

Physikalische Geräte

→ Maus, MIDI, Mischpult, Schalter

Virtuelle Eingabegeräte

→ werden auf Bildschirm simuliert (Selektion im Menü)

Logische Eingabegeräte

→ abstraktes Modell der Eingabe

→ Sicht der Software

Beispiel:

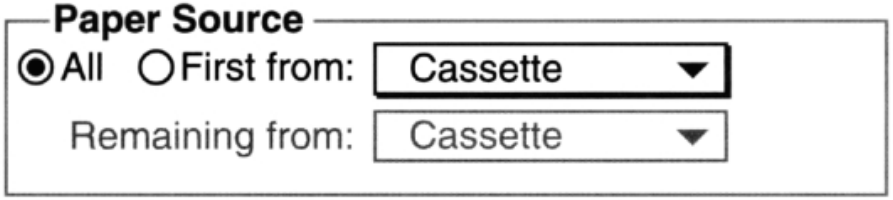
Beenden eines Programms mit logischem “Quit”-Button

entweder: Button, realer Knopf

oder: Ctrl-Q, Quit menu item

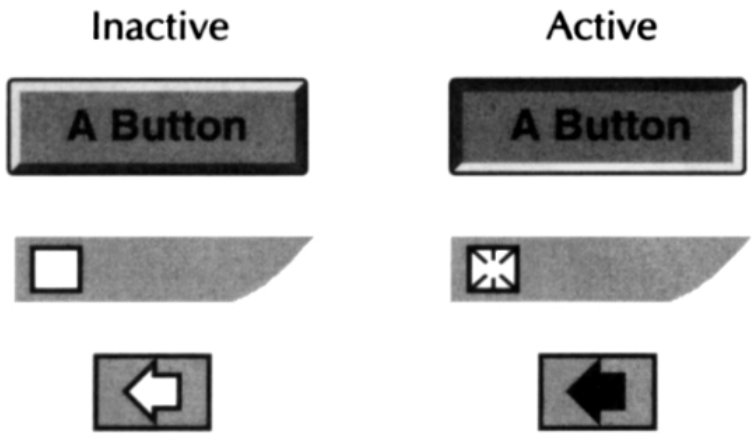
Eigenschaften logischer Eingabegeräte

- **Angefordert / Freigegeben**
  logisches Eingabegerät ist angefordert, wenn es seine Resourcen (Speicherplatz, Bildschirmplatz) alloziert hat, ansonsten freigegeben (Menü: nur beim Aufpoppen wird angefordert)
- **Enable / Disable**
  oftmals ist es besser, Dialogelemente nicht verschwinden zu lassen, sondern sie nur anzudeuten, wenn Eingabe nicht erwünscht (Position aller Dialogelemente bleibt immer gleich)



aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

- **Aktiv / Inaktiv**
  $\longrightarrow$ visuelles Feedback, wenn Dialogelement selektiert



aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

- **Echo**
  $\longrightarrow$ Mechanismus zum Darstellen des aktuellen Zustandes
  (Bsp.: Position des Sliders in einer Scrollbar)

Look and Feel

→ bestimmen visuelle Repräsentation (Look) und Interaktionsschema (Feel) von Dialogbausteinen

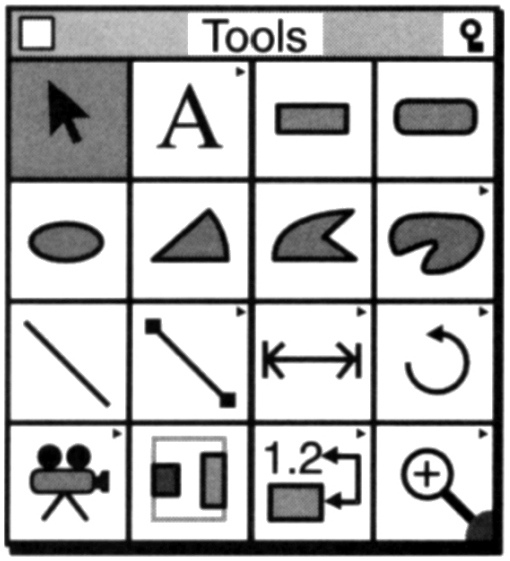
→ Designaufgabe

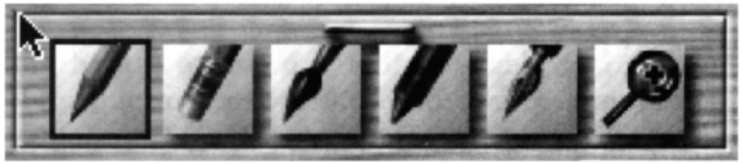
→ wichtig: geeignete Interaktionsmechanismen (lassen sich schwer umlernen)

→ Mittel zur Kundenbindung

Look

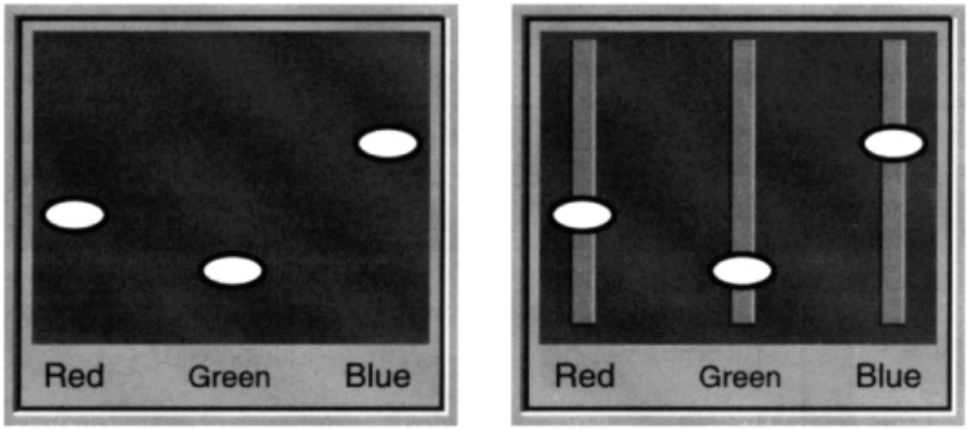
→ visuelle Repräsentation von Dialogelementen





aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

$\longrightarrow$ beinhaltet geeignete Darstellung, sodaß Benutzer weiß, daß er manipulieren kann



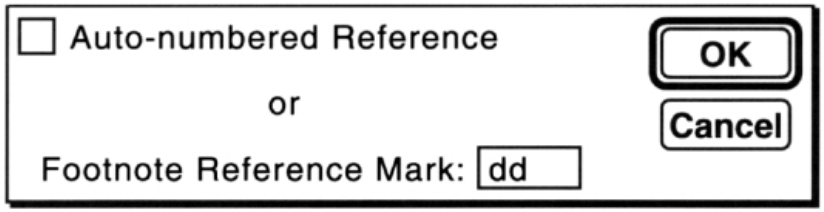
aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

$\Rightarrow$ Visualisierung aller Zustände wichtig

Hinweise zur Gestaltung von Dialogbausteinen:

a) Geeignete Darstellung

$\longrightarrow$ manipulierbare Stellen müssen hervorgehoben sein



MacIntosh

Microsoft Windows

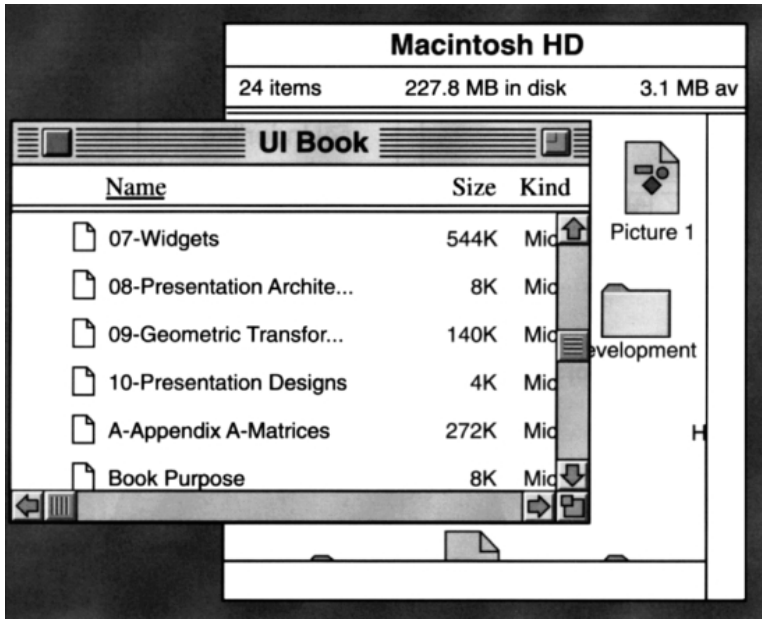
aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

Methoden zur Hervorhebung:

1. Helle Objekte $\Leftrightarrow$ dunkle Objekte
2. Objekte mit Detail und Textur $\Leftrightarrow$ einheitliche, flache Darstellung
3. hoher Kontrast $\Leftrightarrow$ geringer Kontrast
4. große Objekte $\Leftrightarrow$ kleine Objekte
5. unterschiedlich geformt $\Leftrightarrow$ uniform, regelmäßig

b) Visualisierung von Enable / Disable

→ oft: Kontrast-Verminderung, Detail-Verminderung



aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

c) Visualisierung von Active / Inactive

→ 3D-Effekte: Schatten unten ⇔ oben, um Position vor/hinter Bildebene zu simulieren

→ "Ankreuzen" (aktives Feld erscheint markiert)

d) Gruppieren von Widgets

→ funktionale Gruppen visuell repräsentieren (pull-down menü)

→ spart Platz und Zeit beim Suchen

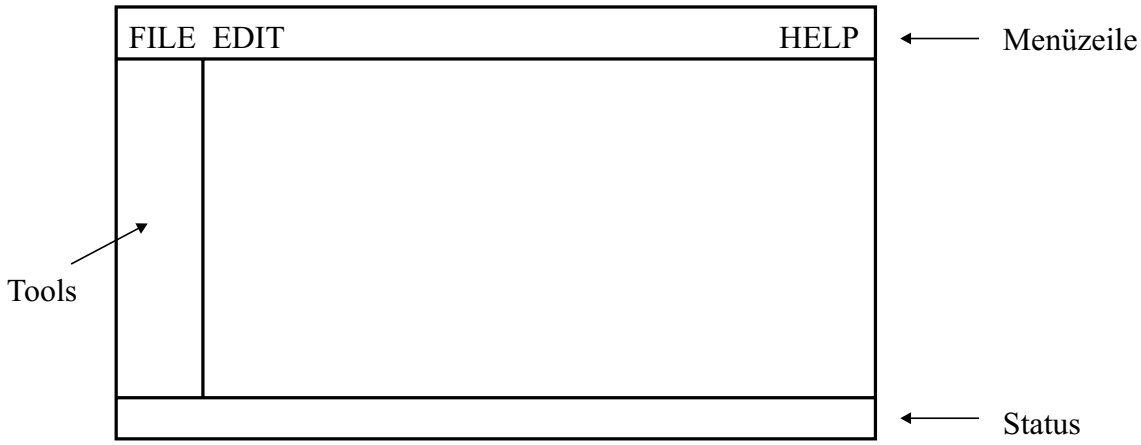
aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

e) Konsistente Darstellung

→ Programme haben dasselbe Gesamtschema

→ spezifiziert Produktdesign

→ erleichtert das Erlernen



aus: R. Olsen: Developing User Interfaces, Morgan Kauffman

Feel

$\longrightarrow$ wird durch Eingabe-Events definiert, die der Benutzer erzeugen muß, um Widget zu manipulieren

$\longrightarrow$ Konsistenz hier noch wichtiger als beim Aussehen

(Beispiel: Kopieren mit Maus $\longrightarrow$ Unix $\Leftrightarrow$ Windows)

Eingabeaktionen:

Die meisten Systeme sind begrenzt auf:

- "drag" (Maus drücken, bewegen, loslassen)
- "click"
- "doppel-click"
- "drag select" (Maus drücken am Anfang, Maus loslassen am Ende)

- “click” für select
  - einmal “clicken”: Buchstabe selektieren
  - zweimal “clicken”: Wort selektieren
  - dreimal “clicken”: Zeile selektieren
- “shift click” (Mehrfach-Selektion)
- Keyboard-Eingabe