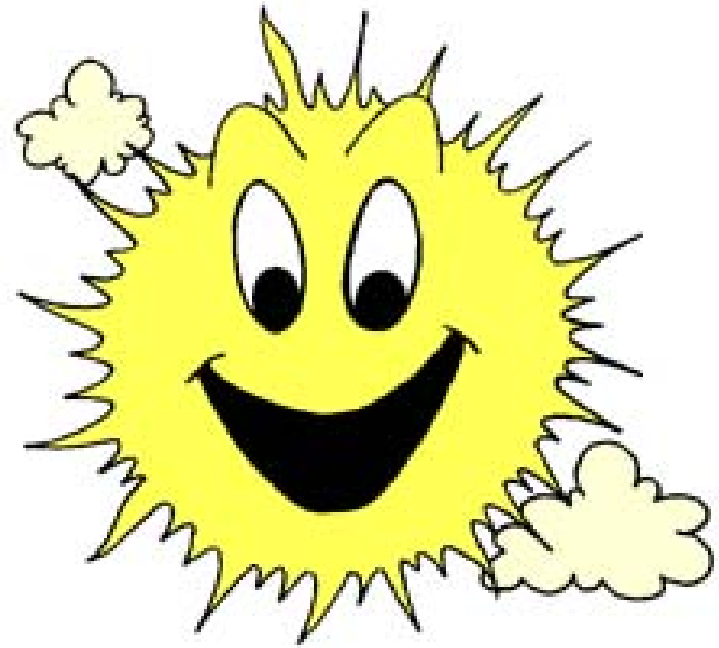
Wiederholung 11. Vorlesung



Farbkombinationen

- Vordergrund- und Textfarben
  - Weiß, Schwarz, mittel- bis langwellige Farben (Rot, Gelb, Grün)
  - kein Blau

- Hintergrundfarben
  - hell (Positivdarstellung), schwach gesättigt (Pastelltöne)
  - kurzwellige (Blau, Cyan) und unbunte Farben (helles Grau)
  - **kein** gesättigtes Rot oder Braun

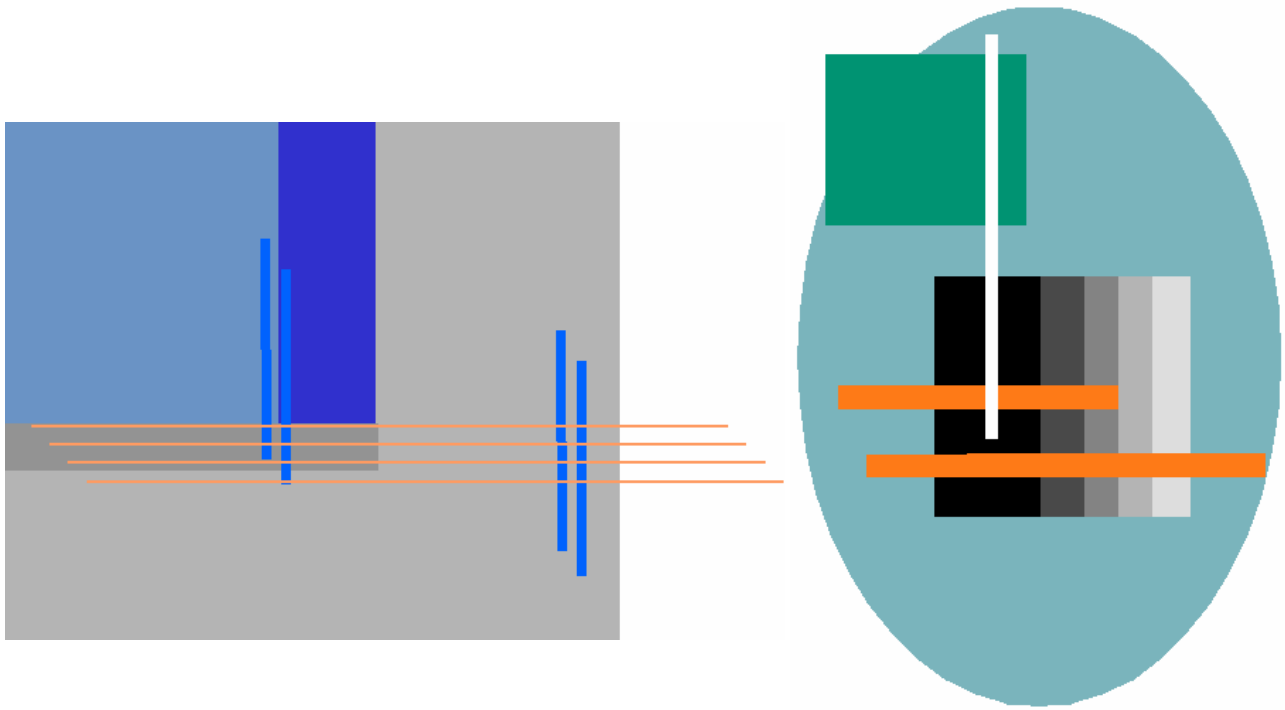
- kritische Farbkombinationen
  - vermeiden: Rot/Grün, Gelb/Blau, Rot/Blau
  - Gelb/Weiß wird leicht verwechselt

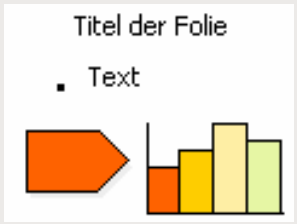
Geeignete Farbkombinationen

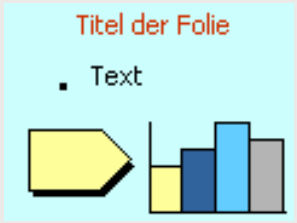
- Text auf verschiedenen Hintergründen muss gut und ermüdungsfrei lesbar sein.
- Text auf verschiedenen Hintergründen muss gut und ermüdungsfrei lesbar sein.
- Text auf verschiedenen Hintergründen muss gut und ermüdungsfrei lesbar sein.
- Text auf verschiedenen Hintergründen muss gut und ermüdungsfrei lesbar sein.
- Text auf verschiedenen Hintergründen muss gut und ermüdungsfrei lesbar sein.

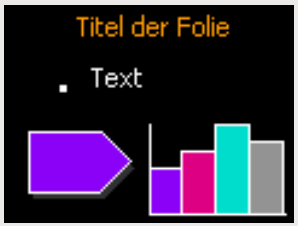
Farbschema / Farbklima

- ±5 verschiedene Farben
  + schwarz + weiß









Im Zweifelsfall

- eher weniger Farbe als mehr
- eher weniger Farbsättigung als mehr
- Codierung und Unterscheidung eher mittels Layout und Form als mittels Farbe
- Farben visuell überprüfen, nicht "berechnen"

Ausrichtung von Beschriftungen

- einzeilige Steuerelemente:
  Label links; horizontal zentriert

einzeilig

- mehrzeilige Steuerelemente:
  Label links oberhalb

mehrzeilig

- mehrere verschieden lange Labels:
  etwa gleich: linksbündig sehr unterschiedlich: rechtsbündig

minimal

mittel

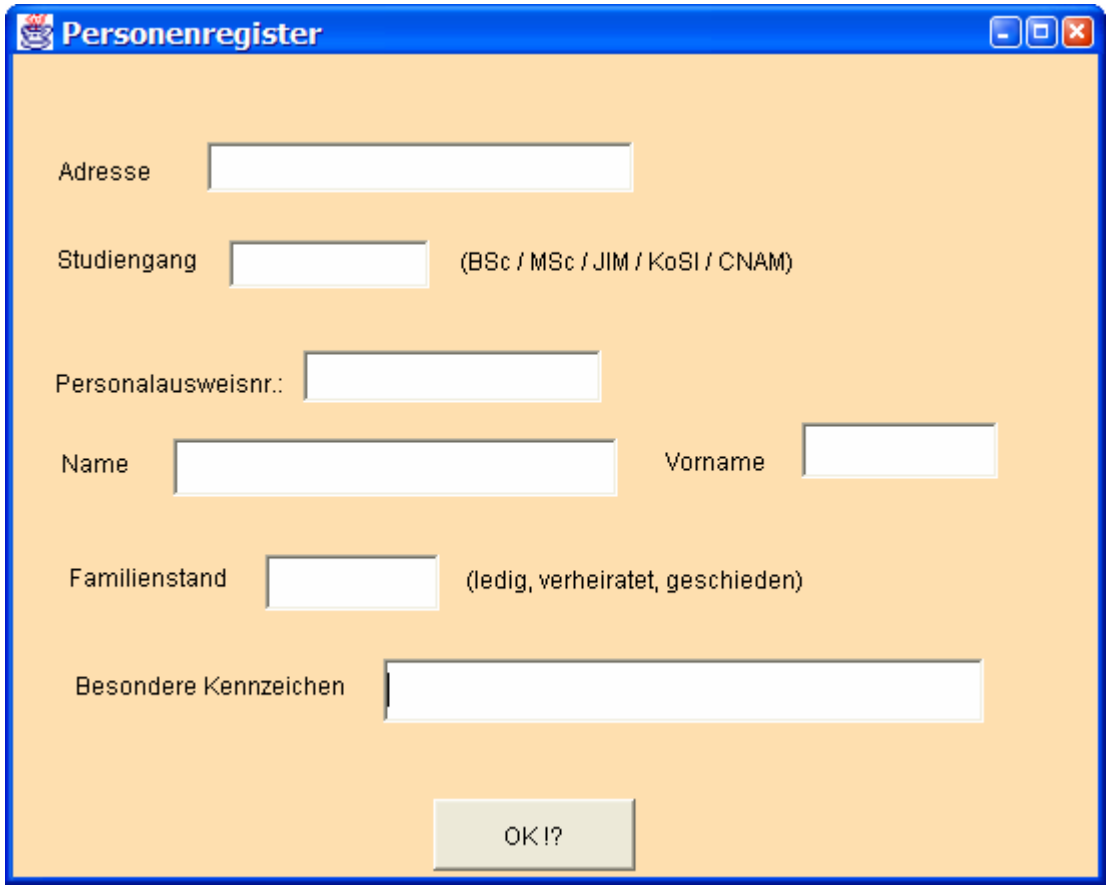
maximal

Tabulator

Makro

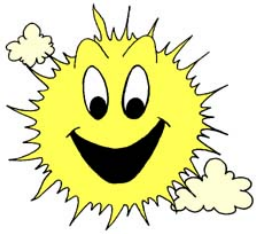
Einblendezeit

Gestalten eines Eingabefensters

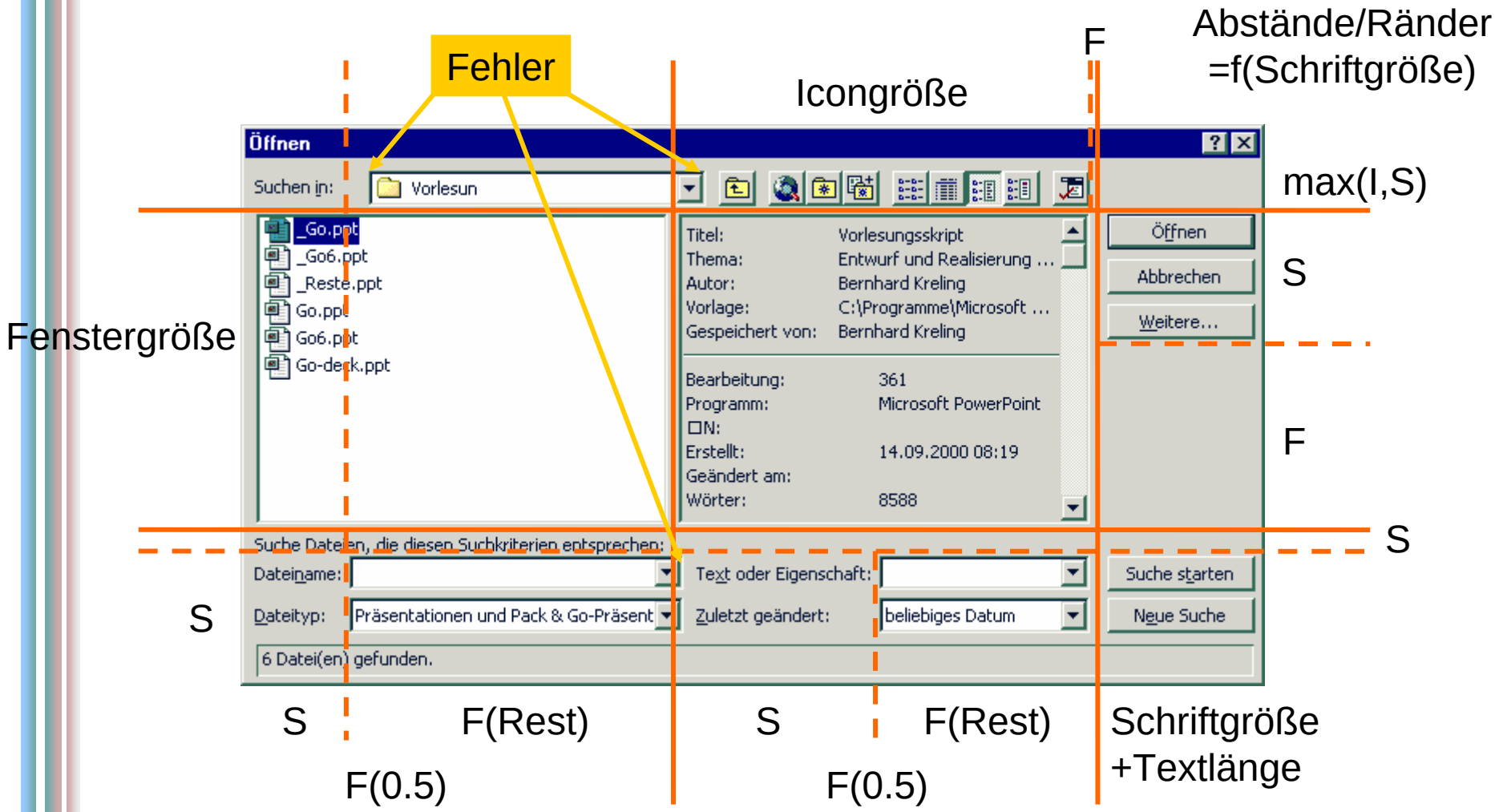


Was ist hier schlecht ? Verbessern Sie den Entwurf !

Ende der Wiederholung



Layout-Raster



Vorgehensweise beim Layout

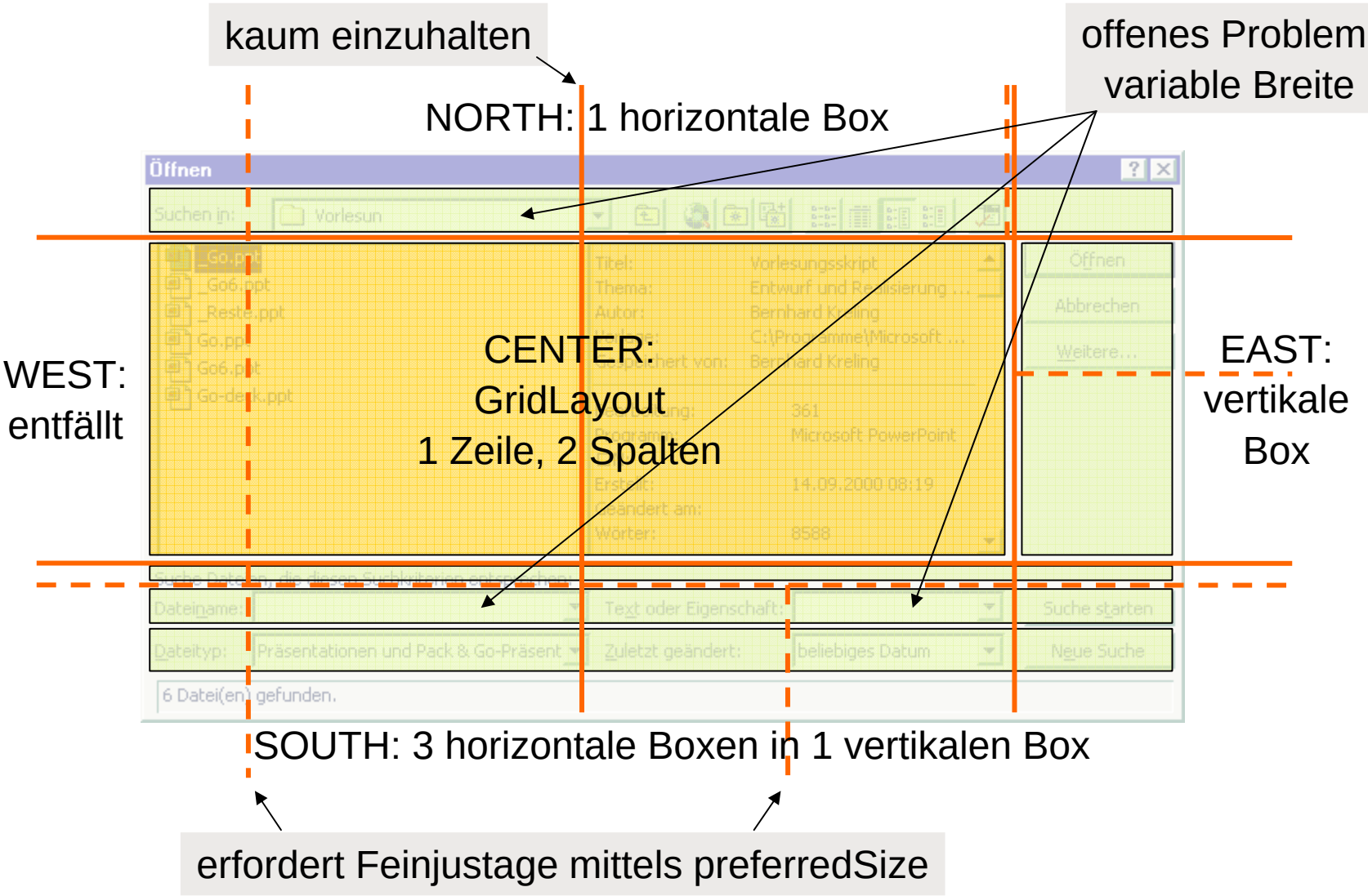
- notwendige Komponenten aus Use Cases ermitteln
- Komponenten freihändig gruppieren und positionieren
- Flucht- und Symmetrielinien identifizieren / definieren
  - Anzahl der Fluchtlinien klein halten
  - Fluchtlinien können gezeichnete Linien ersetzen: "gutes Design braucht keine (gezeichneten) Linien"

das Auge ist sehr empfindlich !

- Rasterzeilen und -spalten bemaßen
  - dyn. Layout: abhängig von Schrift-, Fenster- und Icongröße
    - bei mehrsprachigen Anwendungen auch abhängig von Textlänge
    - im allgemeinsten Fall sind alle Maße variabel!
  - (statisches Layout: absolute Zahlen in Pixel)

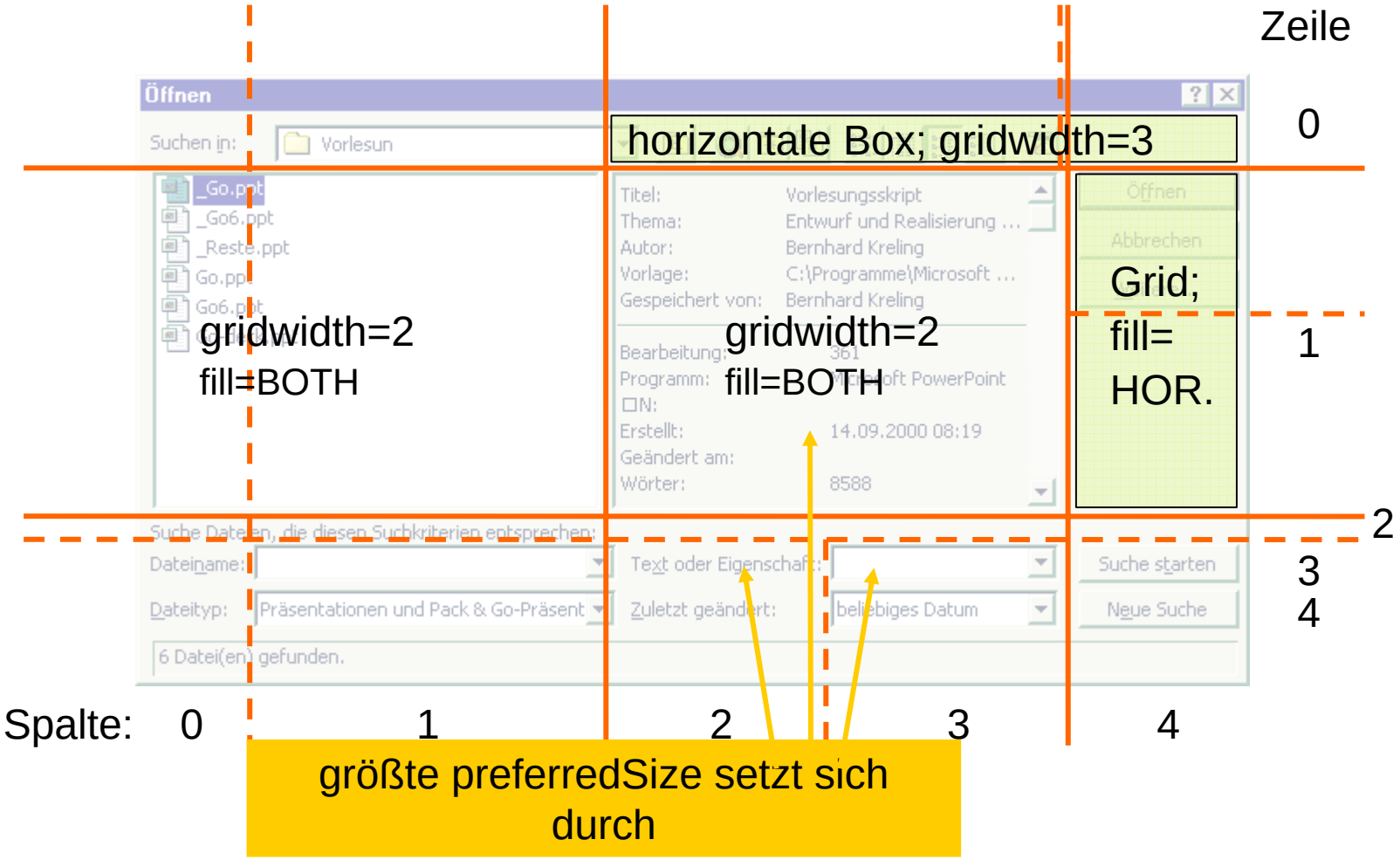
Analyse → Gestaltung → Konstruktion

Beispiel mit geschachtelten Layouts



Layout mit GridBagLayout

alle Zellen in Zeilen 0,2,3,4: fill=HORIZONTAL



Geeignete Auswahl treffen

- möglichst viel Platz für eine Komponente
  - BorderLayout: im CENTER
  - GridBagLayout:
    `constraint.fill = GridBagConstraints.BOTH`
- ein paar Komponenten kompakt in einer Zeile oder Spalte in deren jeweils eigener Standardgröße:
  - auf JPanel mit **`FlowLayout`** (nur Zeile) oder **`BoxLayout`**
- ein paar Komponenten gleicher Größe:
  - **`GridLayout`** (evtl. nur 1 Zeile oder 1 Spalte)
- komplexes Layout mit vielen Komponenten
  - **`GridBagLayout`** oder mehrere geschachtelte Panels
  - feste und variable Zellengrößen frei kombinierbar

Anmerkung 5. / 6. Praktikum

- 5/2. Überprüfen Sie, ob alle **Designkriterien** erfüllt sind. Führen Sie gegebenenfalls die notwendigen Verbesserungen durch.
- 5/4. Beschreiben Sie **schriftlich**, wie Sie Ihre **Anwendung testen** wollen (***Vorbereitung***)
  Führen Sie die Tests durch und protokollieren Sie den Testverlauf.
  - nummerierte Testfälle aufschreiben und die erwartete Antwort oder Verhalten des Systems zu notieren. z.B.
    1. Login
    1.1 richtiger Name und Passwort
    1.2 richtiger Name, falsches Passwort
    1.3 falscher Name, richtiges Passwort
    1.4 falscher Name, falsches Passwort
    - 1.1 System nimmt Benutzer an
    - 1.2-1.4 System lehnt Benutzer ab und gibt eine verständliche Rückmeldung
- Getestet wir gegen den Testplan, d.h. Vorgehen anhand des Plans

Debugging in JBuilder

statische Bilder: ImageIcon

- Buttons, Labels, Listen etc. können Icons darstellen
- Bilddateiformat GIF oder JPEG
  - am einfachsten mit Bildbearbeitungsprogramm erstellen und in Datei ablegen
  - GIF auch mit transparentem Hintergrund ("freigestellt")

```
ImageIcon Bild = new
ImageIcon("Verzeichnis/Bild.gif");
JLabel Label = new JLabel();
Label.setIcon(Bild);

ImageIcon Bilder[] = { new ImageIcon("Bild0.gif"),
                       new ImageIcon("Bild1.gif") };
JComboBox ComboBox = new JComboBox(Bilder);
```

Buttons und Bilder: Icons

```
JButton jButton1 = new JButton();
JButton jButton2 = new JButton();
...
ImageIcon ii = new ImageIcon ("contact.gif");
jButton1.setPreferredSize(
  new Dimension(ii.getIconWidth(),
  ii.getIconHeight() ));
jButton1.setIcon (ii);
...
jButton2.setText("Kontakt");
contentPane.add (jButton1, ...);
contentPane.add (jButton2, ...)
```

javadoc

- Hier eine Auswahl:
  - `@see name` Hyperlink auf eine Klasse
  - `@see name#methode` Hyperlink auf eine Methode
  - `@version text` steht (nur) vor Klassendefinition
  - `@author text` steht (nur) vor Klassendefinition
  - `@param name beschr.` Parameterbeschreibung vor Methoden
  - `@return beschr.` steht (nur) vor Methoden
  - `@exception Klassenname` Beschreibung steht vor Methode

- Erzeugt wird die Dokumentation mit dem Befehl

```
javadoc datei.java –version –author
  –windowtitle “ein Text”
```

- Nutzen Sie diese Dokumentationsmöglichkeiten in Ihrem Programmcode für die von Ihnen (weiter-)entwickelten Klassen im Praktikum!

Javadoc (2)

- der Befehl
  `javadoc datei.java ...`
  erzeugt die Dokumentation im **aktuellen** Verzeichnis
- Mit dem Parameter **`-d`** kann der **Ausgabepfad** angegeben werden:
  `javadoc datei.java -d c:\temp\meinedoku`
- Es gibt eine Unmenge von Parametern für javadoc.
  `javadoc -help`
  gibt Auskunft. Siehe auch Literatur dazu.

- Ein weiteres sehr nützliches Werkzeug zur Erstellung von Dokumentation, nicht nur für Java, ist **doxygen**
  - www.doxygen.org
- Mit doxygen lassen sich auch Text-Dokumentationen erzeugen, z.B *.rtf, LaTex
- In Verbindung mit graphviz (www.graphviz.org) sogar mit Grafik, z.B. Klassendiagrammen.

jar – Java Archive Tool

- `jar` ist ein Werkzeug, mit dem Java-Archivdateien erzeugt und verändert werden können. Eine jar-Datei ist eine komprimierte ZIP-Datei; sie enthält i.d.R. kompakt mehrere Dateien.
- Beispiel:
  - **`jar cmvf manf beispiel.jar *.class`**
    erzeugt eine jar-Datei mit allen *.class Dateien
    manf ist eine Datei, die Informationen enthält über Klasse mit der main-Methode (sog. Manifest)
  - **`jar cvf beispiel.jar mypackage/*.class`**
  - **`jar tvf beispiel.jar`** jar-Datei ansehen
  - **`java –jar beispiel.jar`** jar-Datei ausführen
  - **`java –classpath beispiel.jar mypackage.MyApp`**
  - Inhalt der Datei manf:

```
Main-Class: beispiel
Sealed: true
```