Topics:
1. Describing implementation of widgit toolkits in terms of MVC
2. How look-and-feel relates to widgits

Implementation of its widgits largely determines look-and-feel of a GUI:
- **The Look**: comes from visual presentation of widgits
- **The Feel**: comes from their interactive behavior

The implementation of a widgit class can be conceptualized in terms of a simple Model-View-Controller design

- **Model**: very simple for widgits; the state information maintained by the widgit
- **View**: the visual presentation of the widgit
- **Controller**: handles the widgit's input events

Advantage of implementing widgit with separate model: easier to develop new look-and-feel without changing model (*but rare for separate widgit View and Controller implementations*)
Grouping widgets according to what information is in their models:

I. Buttons:
   - Model is value of a single variable
   - Example: 3 radio buttons sharing a single model value (size variable):

   ![Radio Buttons Example](image)

II. Sliders and scroll bars
   - Model is upper and lower limit and current value in continuous range of values
   - Thumb (handle on slider) controls current value
   - Example: volume control

Sometimes model for scroll bar also includes page size (so user can click between handle and end to advance by one page)
III. Menus

- Model similar to button model, but must be able to handle large number of possible values

IV. Text Boxes

- Model is string of textual data

Design issues related to look-and-feel:

1. Ease of learning: must be obvious from appearance what widgit's purpose is and how to use it
   - consistency helps

2. Ease of use: once the user has learned how to use the widgit, is it easy to use?
   - Not the same as (1) above, e.g. widgits that are physically difficult to control

3. Attractiveness:
   - "Looks sell software"
   - as long as they do not interfere with usability

Issues to consider when designing the Look:
I. what information must be presented visually
II. screen space
III. consistency
IV. software architecture
Information that must be presented visually by widgit:
1. Affordances
   - Definition: a property of the visual appearance that makes objects appear as if they can be manipulated, and that provides cues as to what might happen if the object is manipulated
   - Sometimes affordances must be learned
   - Can exploit metaphor with real-world object
   - Example: Fig. below shows two sets of sliders. Set on right have "tracks" → visual cue that ovals can be slid up and down

![Diagram of sliders showing affordances](image)

Human perception issues in designing affordances:
- Lighter objects attract attention, e.g. ovals above
- Objects with detail/texture attract attention
- Objects with high contrast attract attention: ex. gray out disabled widgets
- Large takes precedence over small
- Varied takes precedence over regular
Information that must be presented visually by widgit (cont.)

2. Enable/disable status

- Whenever widgit does not accept input, need to visually show that it is disabled

- Confusing to make the disabled widgit just disappear

- Better to show its status, e.g., by reducing contrast

3. Active/inactive status

- Widgit should react to input to show that is active

- Example: animation of button when user performs action on it to show that it "heard" the action

4. Echo

- Purpose: visually display current value

- Example: showing settings in Font dialog box as checked or unchecked
5. **Structural grouping**
- Related widgits often grouped to support complex task, e.g. dialog box -- reasons:
  - Related to same object in model
  - Conserving screen space and simplifies finding related widgits
  - Grouping helps users remember

6. **Commercial product style**
- E.g. Macintosh vs. Windows

Issues to consider when designing the Look:
- what information must be presented visually
- screen space
- consistency
- software architecture

**Screen space** is scarce -- only 19x19 inches -- and lower resolution than a real desktop, so important to use it effectively! Tips:

- Physical design of widgits to provide needed functionality while minimizing space, ex. sliders instead of big circular dial

- But, size can be used to draw attention, ex. in flight simulator, artificial horizon needs to be very prominent (even though sliders could be used as control)
Issues to consider when designing the Look:
- what information must be presented visually
- screen space
- consistency
- software architecture

Consistent Look of widgits

1. Ease of learning
   - Users expect widgits that look alike to behave alike
   - Example (Fig. 6-26): Looks like you can type in a number:

   ![Example Image]

   But you change the font by pressing the 3D button (on the right) to get a menu of choices!
   - User needs to learn this

2. Ease of use
   - Important to make it easy to visually scan screen to find desired control

   - Ease depends on
     - consistency of spatial location

   - Type of scanning required:
     1. Fastest: image-matching
     2. Slower: concept-matching
Scanning using image-matching:
- User already has an image in mind ("template", "schema")
- User's vision system automatically searches for pattern matching the image
- Pre-wired into visual perception system
- Example: finding a paperclip in a drawer
- Consistent look of widgits exploits image matching
  - User learns an image of what a button should look like

Scanning using concept-matching:
- User has to check each item and think about whether it is what he is looking for
- Requires inference, not automatic
- Example: looking at menu items for first time

Issues to consider when designing the Look:
- what information must be presented visually
- screen space
- consistency
- software architecture: allow internationalization and allow artists to design look
Issues to consider when designing the Feel:

• feel is determined by input events needed to manipulate the widgit, i.e., "syntax"

• consistency of feel more important than consistent look!
  • Why? User needs to remember how to manipulate widgit (invisible syntax)

• Example: cut and paste
  • Select text (mouse-move-to-text-button-down-drag-release)
  • Select copy (mouse-move-to-button-click-release)
  • Select location (mouse-move-to-text-click)
  • Select paste (mouse-move-to-button-click-release)

• Experienced users: input behavior in "muscle memory"

• If behavior change then requires conscious thought and may clash with automatic behavior

• Principle of Safety: make it safe for users to explore GUI actions without damage to system or work! Examples:
  • Scroll bars can be returned to previous position
  • Button not activated until released so user can change his mind
  • Moving or clicking mouse outside of menu will not make a choice