CSE 403
Lecture 6

User Interface Prototyping

Reading:
Paper Prototyping, C. Snyder

slides created by Marty Stepp
http://www.cs.washington.edu/403/

Big questions

- What's the point of prototyping? Should I do it?
  - If so, when in the overall process or "lifecycle" should I?
- Should I make my prototype on paper or digitally?
- How do I know whether my UI is good or bad?
  - What are the ways in which a UI's "quality" can be quantified?
  - What are some examples of software you use that have especially good/bad UIs? What do you think makes them good/bad?

Usability and SW design

- **usability**: The effectiveness with which users can achieve tasks in a software environment.
  - studying and improving usability is part of Human-Computer Interaction (HCI)
  - usability and good UI design are closely related
  - a bad UI can have unfortunate results...

Achieving usability

- some methods to achieve good usability:
  - user testing / field studies
    - having users use the product and gathering data
  - evaluations and reviews by UI experts
  - card sorting
    - Show users various UI menus and ask them to group the ones that are similar, to see what UI tasks are seen as being related by users.
  - prototyping
    - paper prototyping
    - code prototyping
- Good UI design focuses on the *user,* not developer or system.

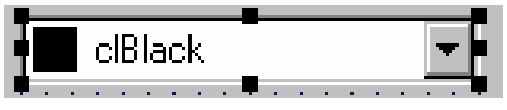
Prototyping

- **prototyping**: Creating a scaled-down or incomplete version of a system to demonstrate or test aspects of it.

- What are some possible benefits of prototyping?
  - aids UI design
  - help discover requirements
  - help discover test cases and provide a basis for testing
  - allows interaction with user and customer to ensure satisfaction
  - team-building

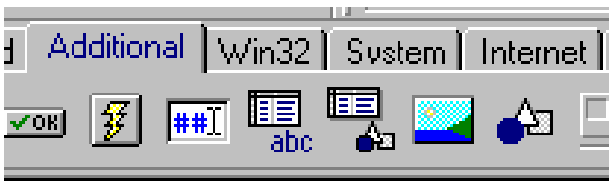
Some prototyping methods

- UI builders (Visual Studio, etc.)



  - draw a GUI visually by dragging/dropping UI controls on screen

- implementation by hand
  - writing a "rough" version of your code



- **paper prototyping**: a paper version of a UI

  Question: Why not just code up a working code prototype?
  - much faster to create than code
  - can change faster than code
  - more visual bandwidth (can see more at once)
  - more conducive to working in teams
  - can be done by non-technical people

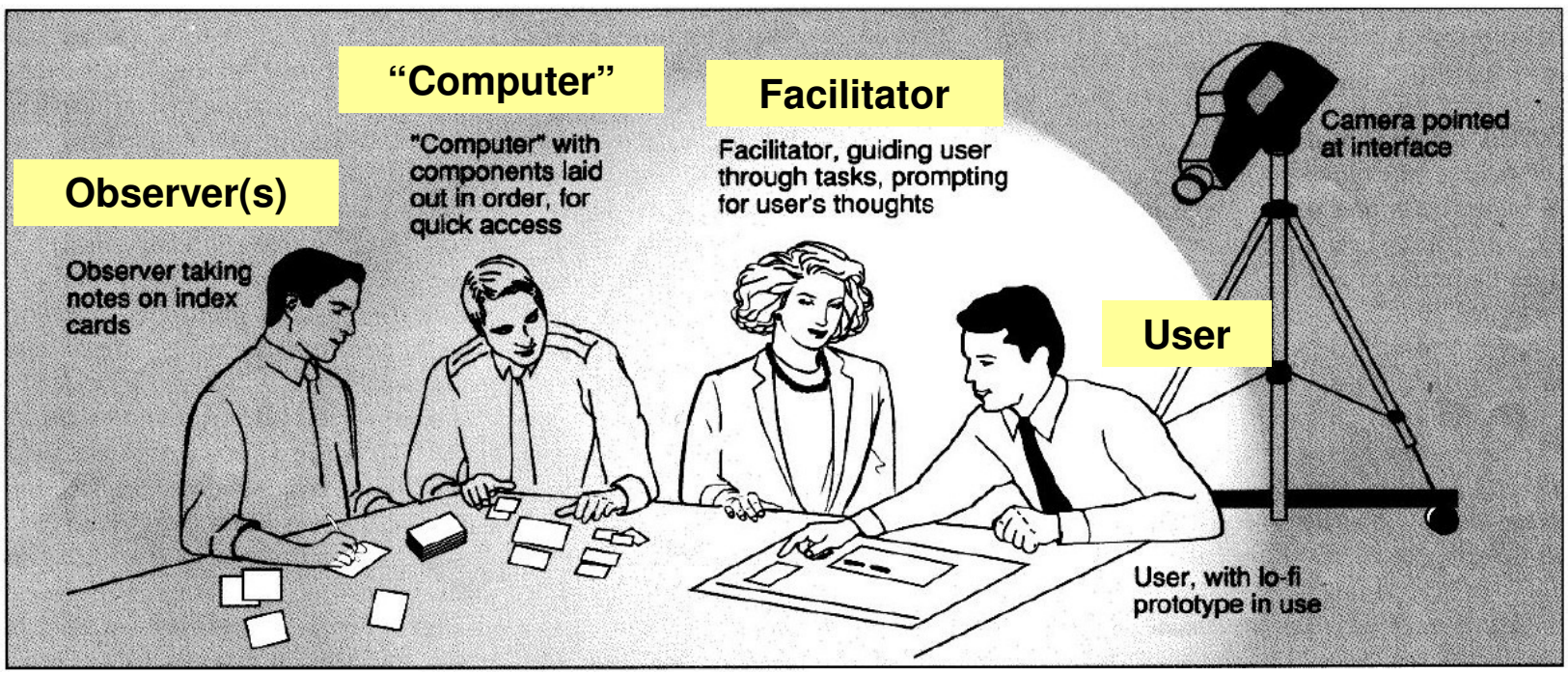
Where does it fit in?

- At what point in the software lifecycle should we do (paper) prototyping? When would it be most useful to do it? Why?

- We talk about requirements being about "what" and design being about "how." Which is paper prototyping?
  - PP helps uncover requirements and also upcoming design issues
  - do PP during or after requirements; before design
  - "what" vs. "how": PP shows us "what" is in the UI, but it also shows us details of "how" the user can achieve goals in the UI

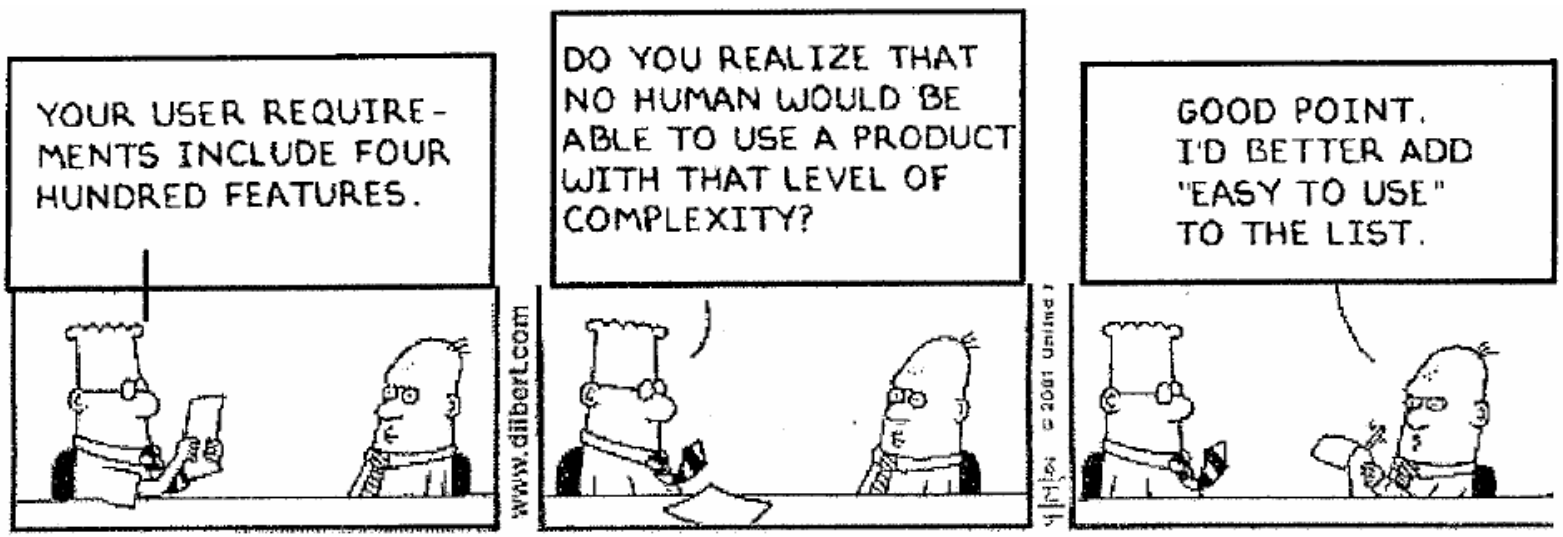
P.P. usability session

- user is given tasks to perform using paper prototype
- session can be observed by people or camera
- one developer can "play computer"



Schneiderman's 8 Golden Rules

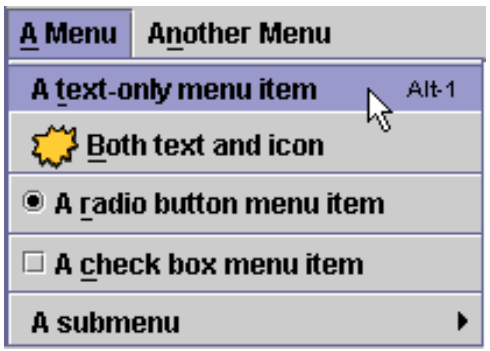
- Strive for consistency.
- Give shortcuts to the user.
- Offer informative feedback.
- Make each interaction with the user yield a result.
- Offer simple error handling.
- Permit easy undo of actions.
- Let the user be in control.
- Reduce short-term memory load on the user.

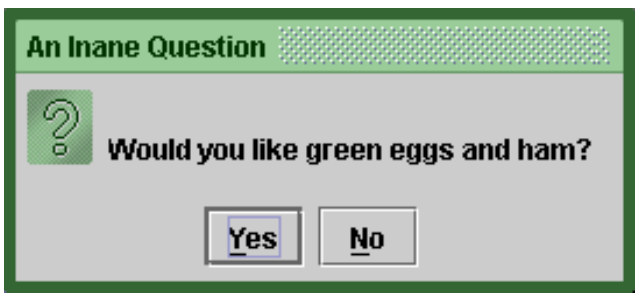


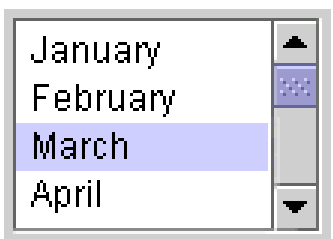
(from Designing the User Interface, by Ben Schneiderman of UMD, HCI/UI expert)

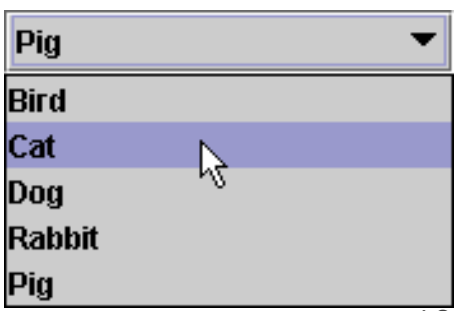
UI design and components

- When should we use:
  - A button?
  - A check box?
  - A radio button?
  - A text field?
  - A list?
  - A combo box?
  - A menu?
  - A dialog box?
  - Other..?

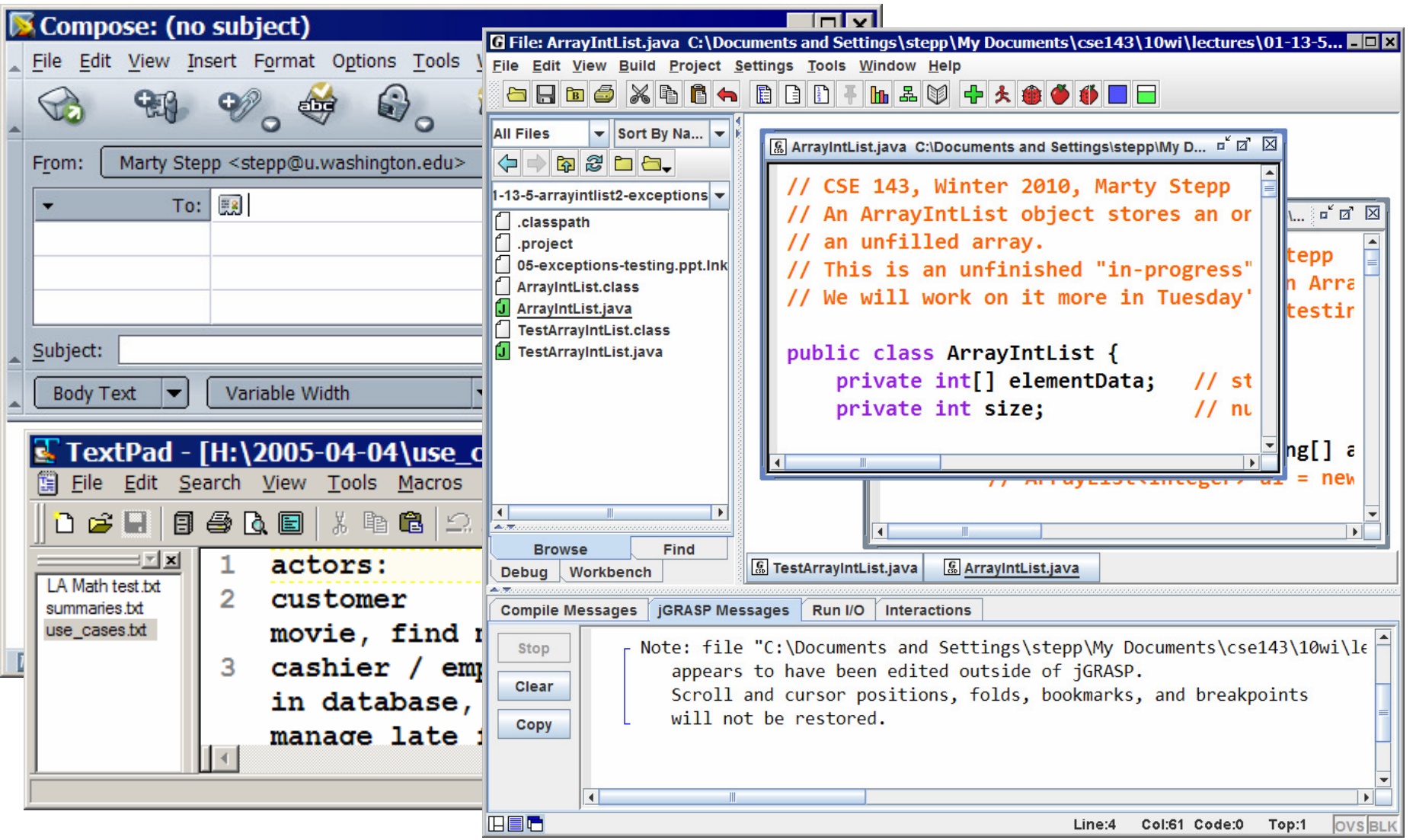








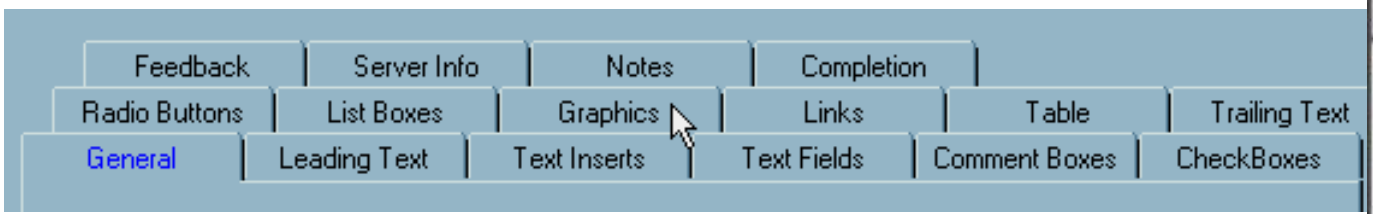
UI design examples

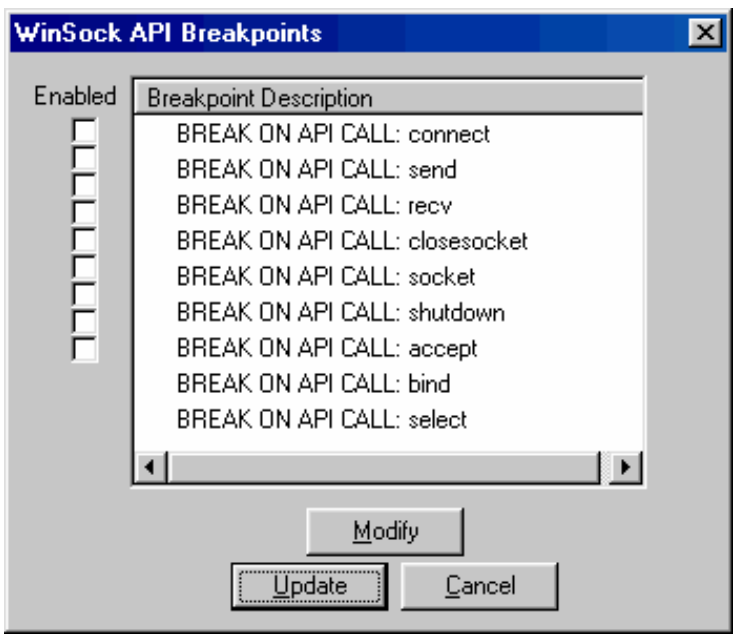


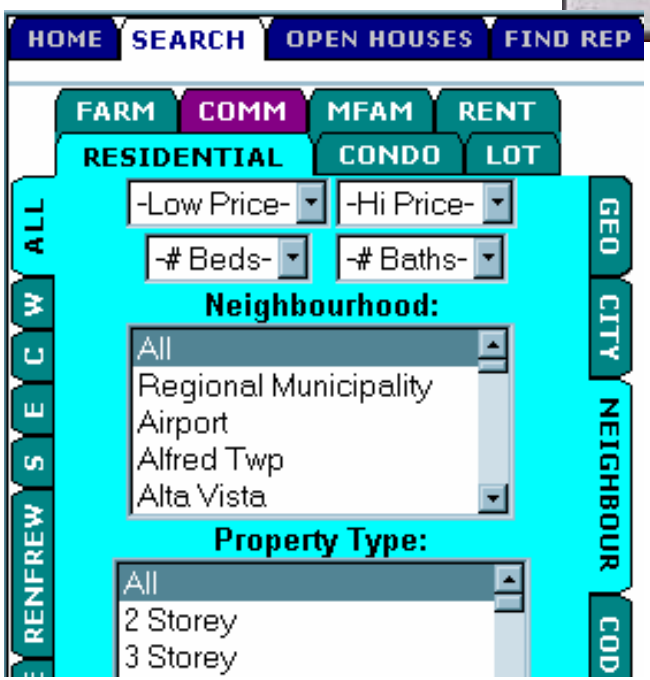
Apple Mac user interfaces

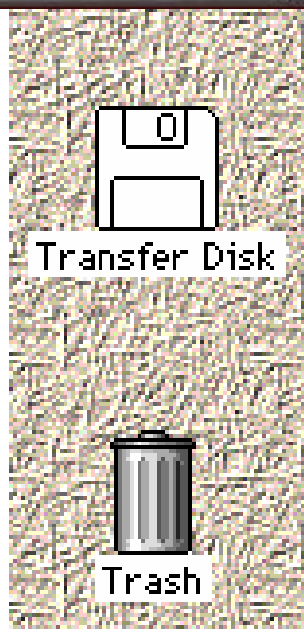
UI Hall of Shame

- http://homepage.mac.com/bradster/iarchitect/shame.htm

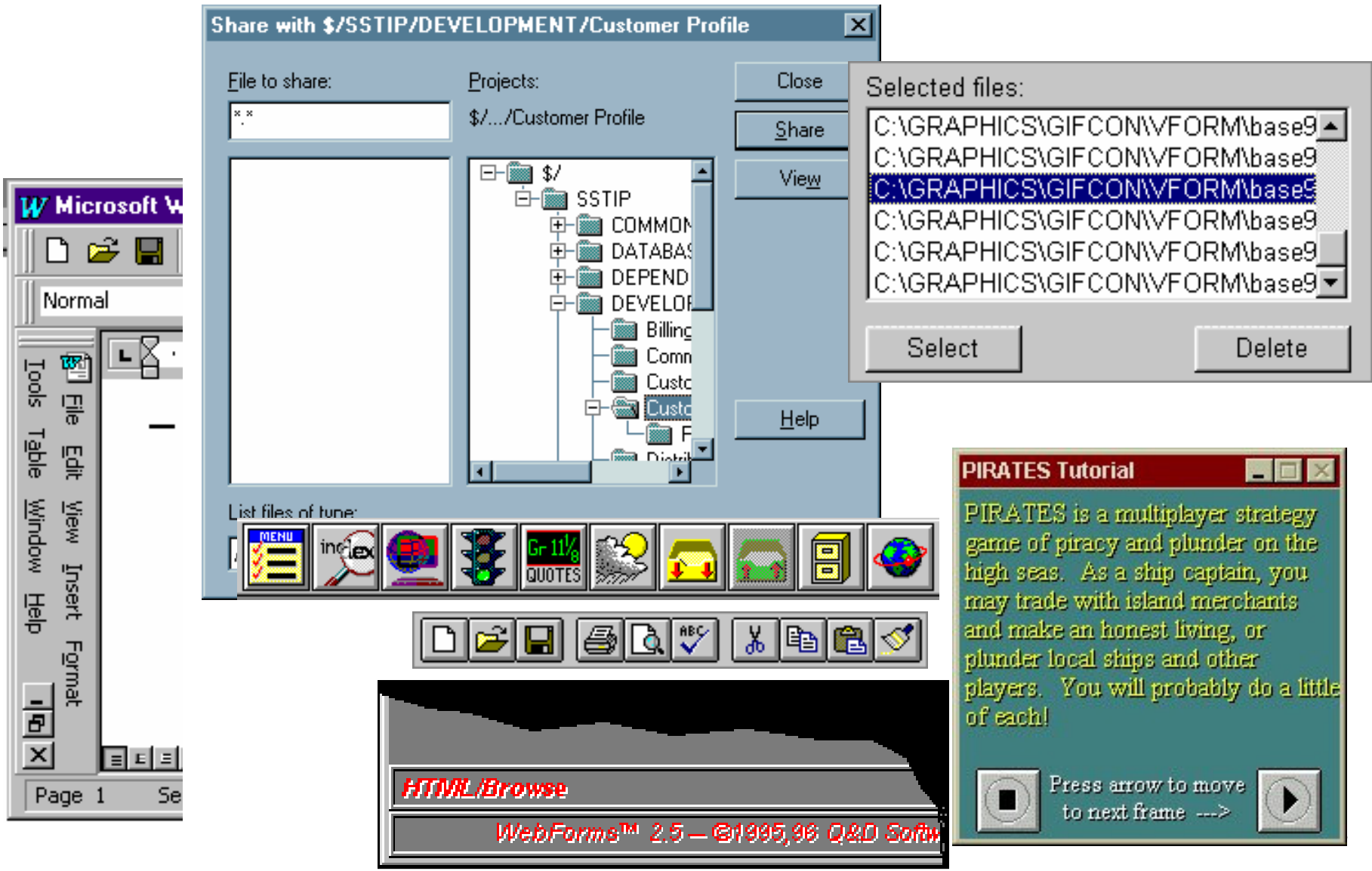




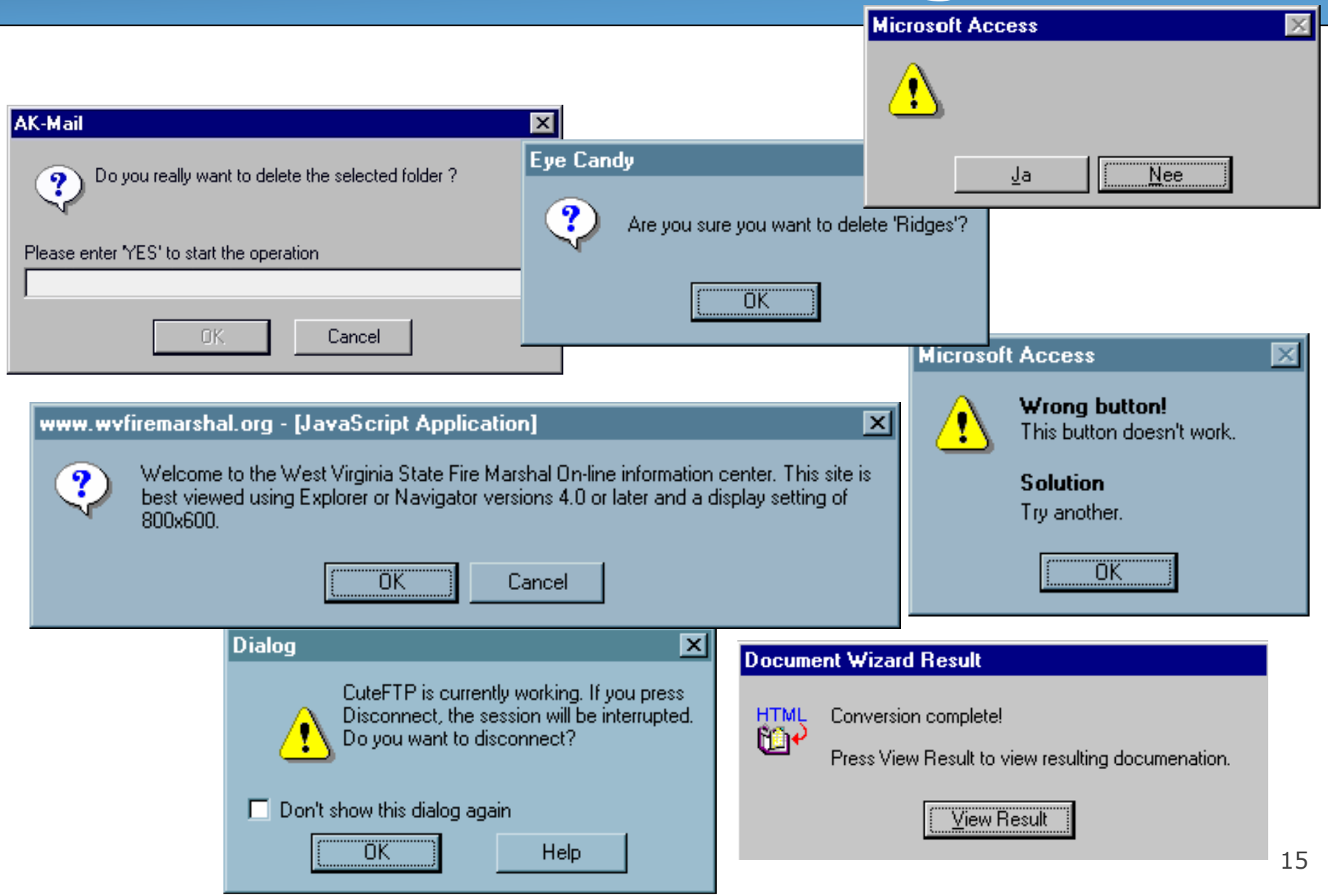




Layout and color



Bad error messages

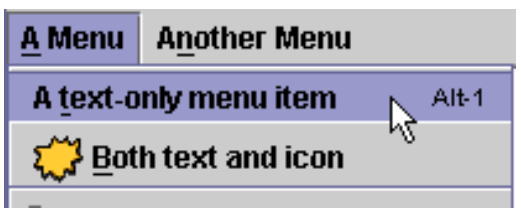


UI design - buttons, menus

- Use **buttons** for single independent actions that are relevant to the current screen.
  - Try to use button text with verb phrases such as "Save" or "Cancel", not generic: "OK", "Yes", "No"
  - use Mnemonics or Accelerators (Ctrl-S)
  - tool tips are helpful, but don't rely on them (many users don't know to hover to find them)

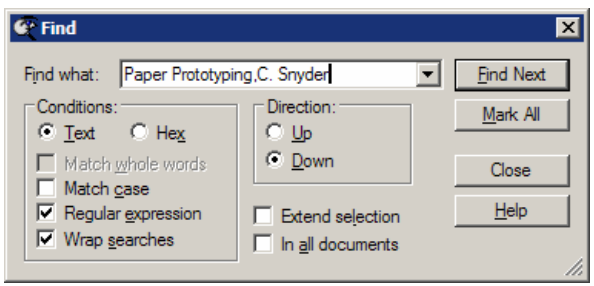
- Use **toolbars** for common actions.

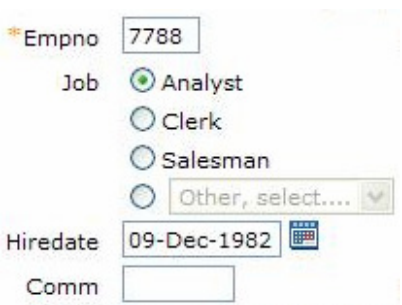
- Use **menus** for infrequent actions applicable to many screens.
  - *Users don't like menus!* Try not to rely too much on menus. Provide another way to access the same functionality (toolbar, hotkey, etc)

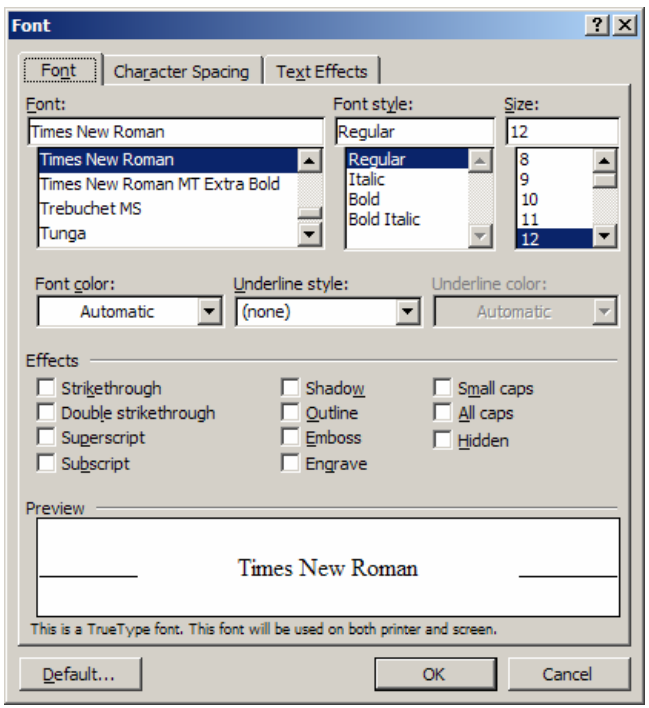


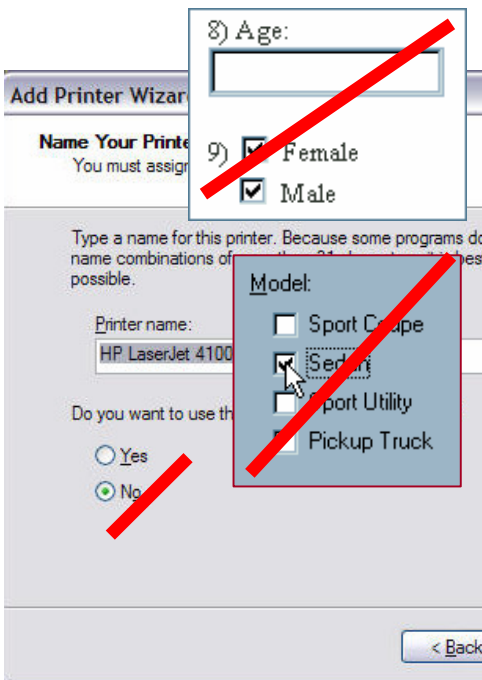
Checkboxes, radio buttons

- Use **check boxes** for independent on/off switches (boolean)
- Use **radio buttons** for a small number of related choices, when only one can be activated at a time (enum / constants)





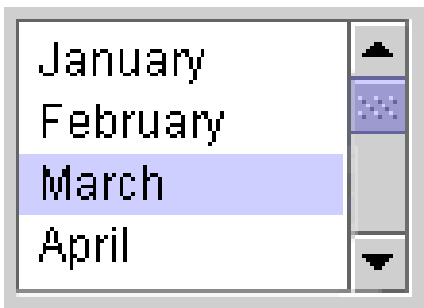




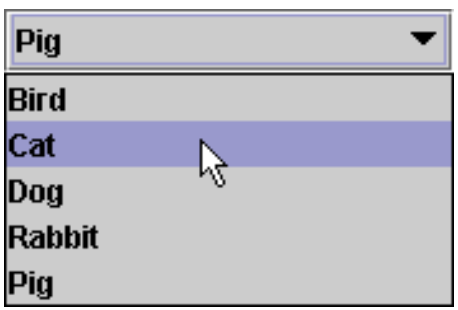
Lists, combo boxes, etc.

- use **text fields** (usually with a label) when the user may type in anything they want
  - you will usually have to **validate** the input

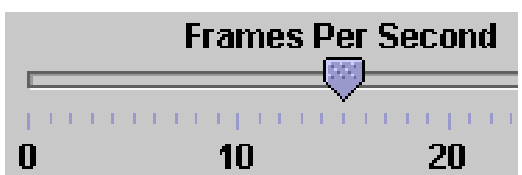
- use **lists** when there are many fixed choices (too many for radio buttons to be practical) and you want *all* choices visible at once



- use **combo boxes** when there are many fixed choices, but you don't want to take up screen space by showing them all at once

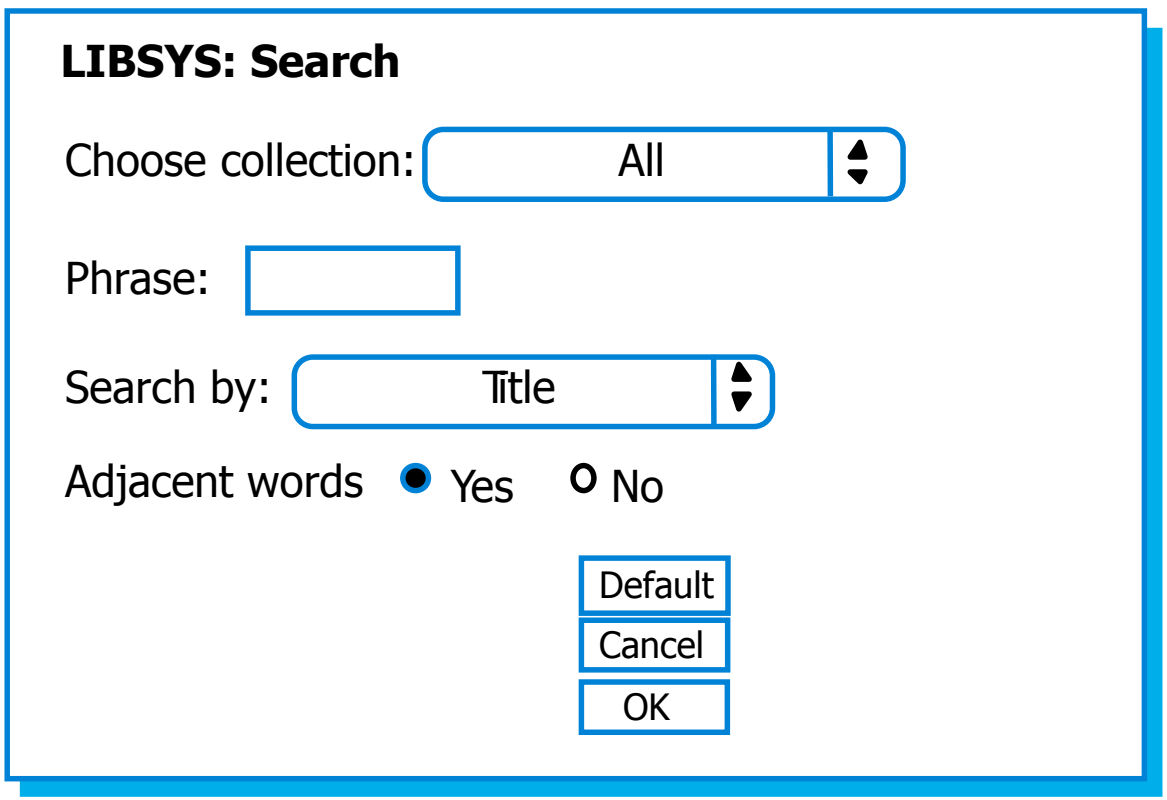


- use a **slider** or **spinner** for a numeric value with fixed range



An example UI

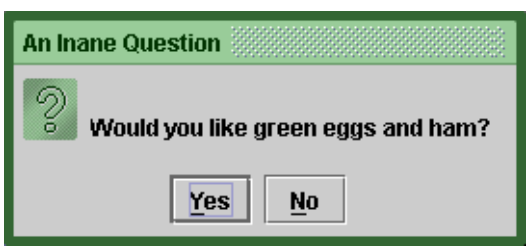
- Did the designer of this UI choose the right components?
  - assume there are 30 collections and 3 ways to search (by title, author, relevancy)



UI design - multiple screens

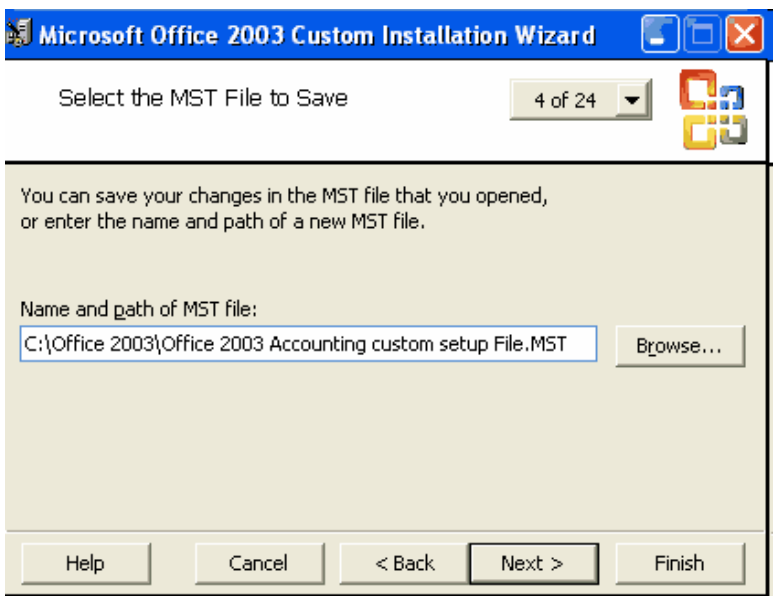
- you can use a **tabbed pane** when there are many screens that the user may want to switch between at any moment
  - or multiple pages, if it's a web site

- use **dialog boxes** or **option panes** to present temporary screens or options
  - users *hate* popup dialogs; use them very rarely
  - don't prompt for lots of user input by popping up dialogs
    - instead, put the choices on the existing window as buttons, etc.



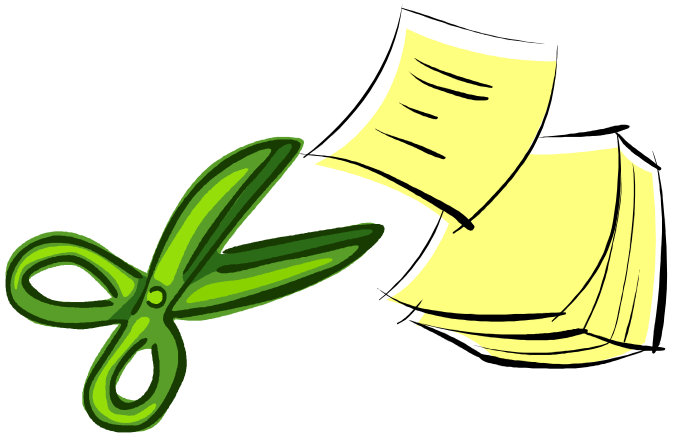
"Wizards"

- **wizard**: series of dialog boxes to progress through a task
- In the mid-1990s, Microsoft changed most of its Windows apps to use "wizards" for installation and settings.
  - Why did they do this?
  - What are the pros and cons of a "wizard" UI?



Creating a paper prototype

- gather materials
  - paper, pencils/pens
  - tape, scissors
  - highlighters, transparencies

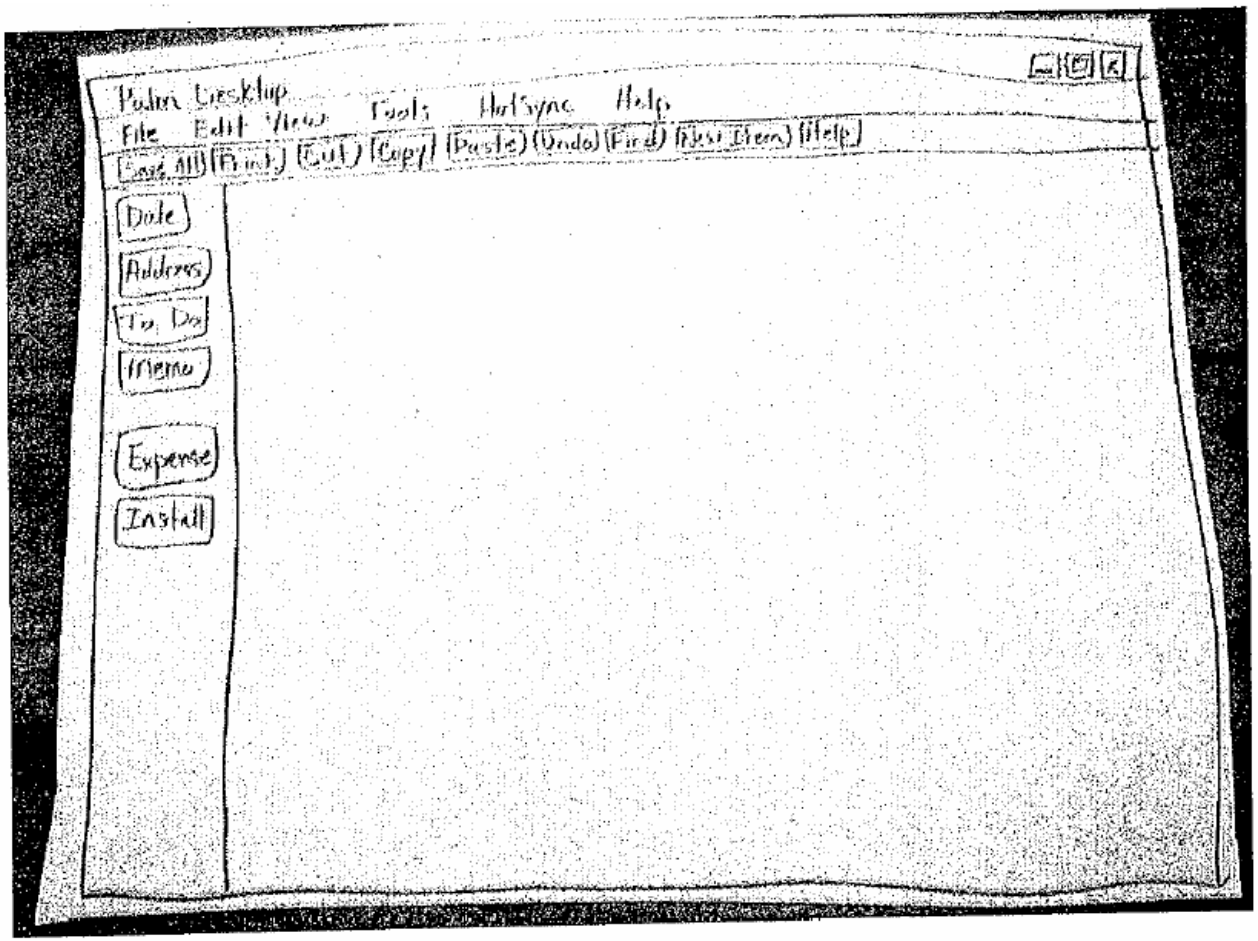


- identify the screens in your UI
  - consider use cases, inputs and outputs to user

- think about how to get from one screen to next
  - this will help choose between tabs, dialogs, etc.

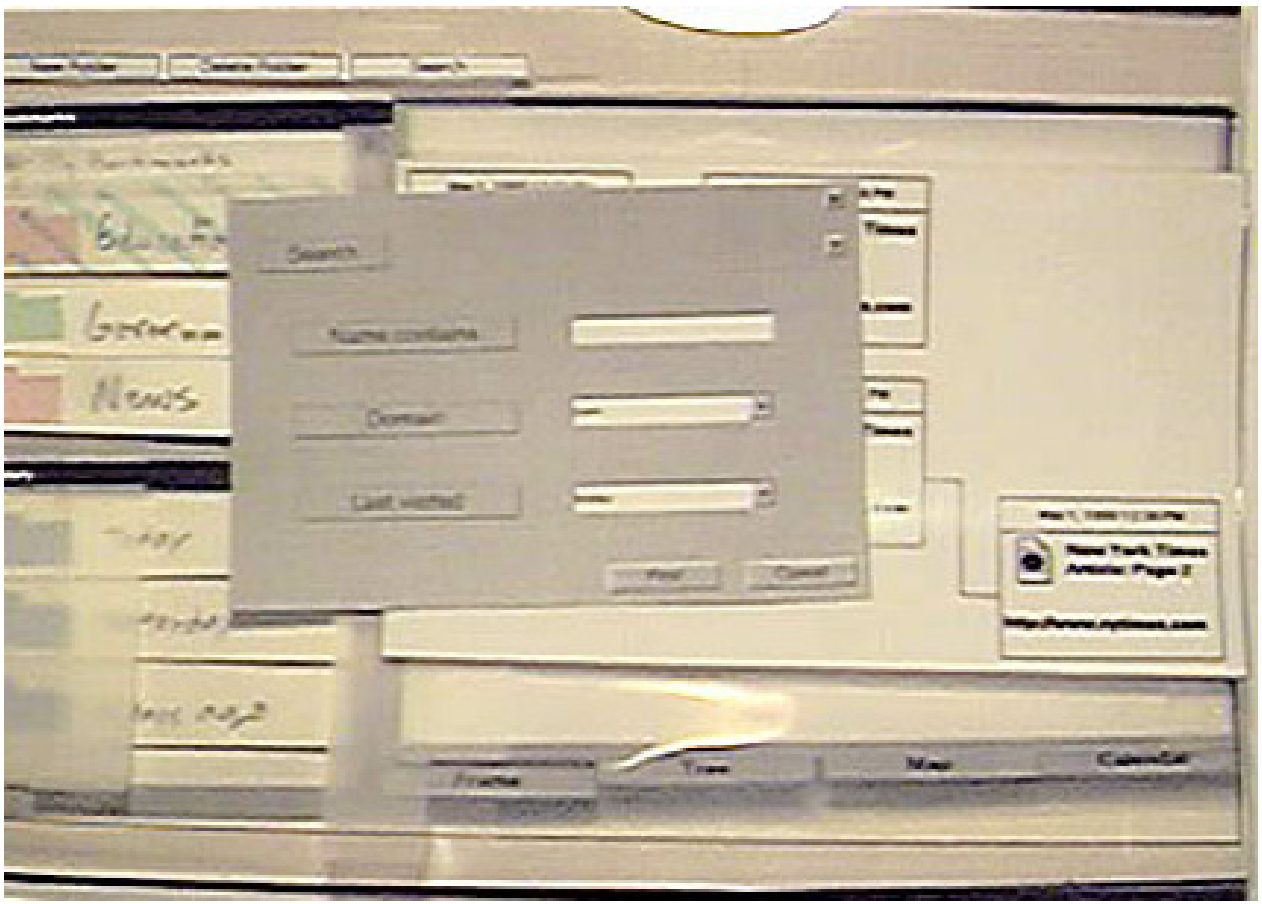
Application backgrounds

- draw the app background (the parts that matter for the prototyping) on its own, then lay the various subscreens on top



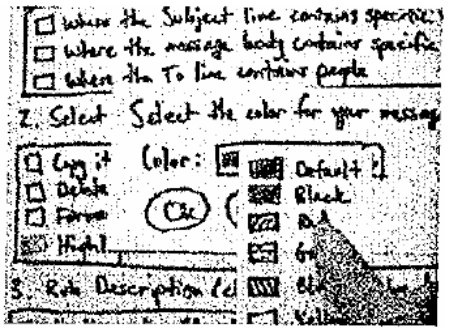
Representing a changing UI

- layers of UI can be placed on top of background as user clicks various options

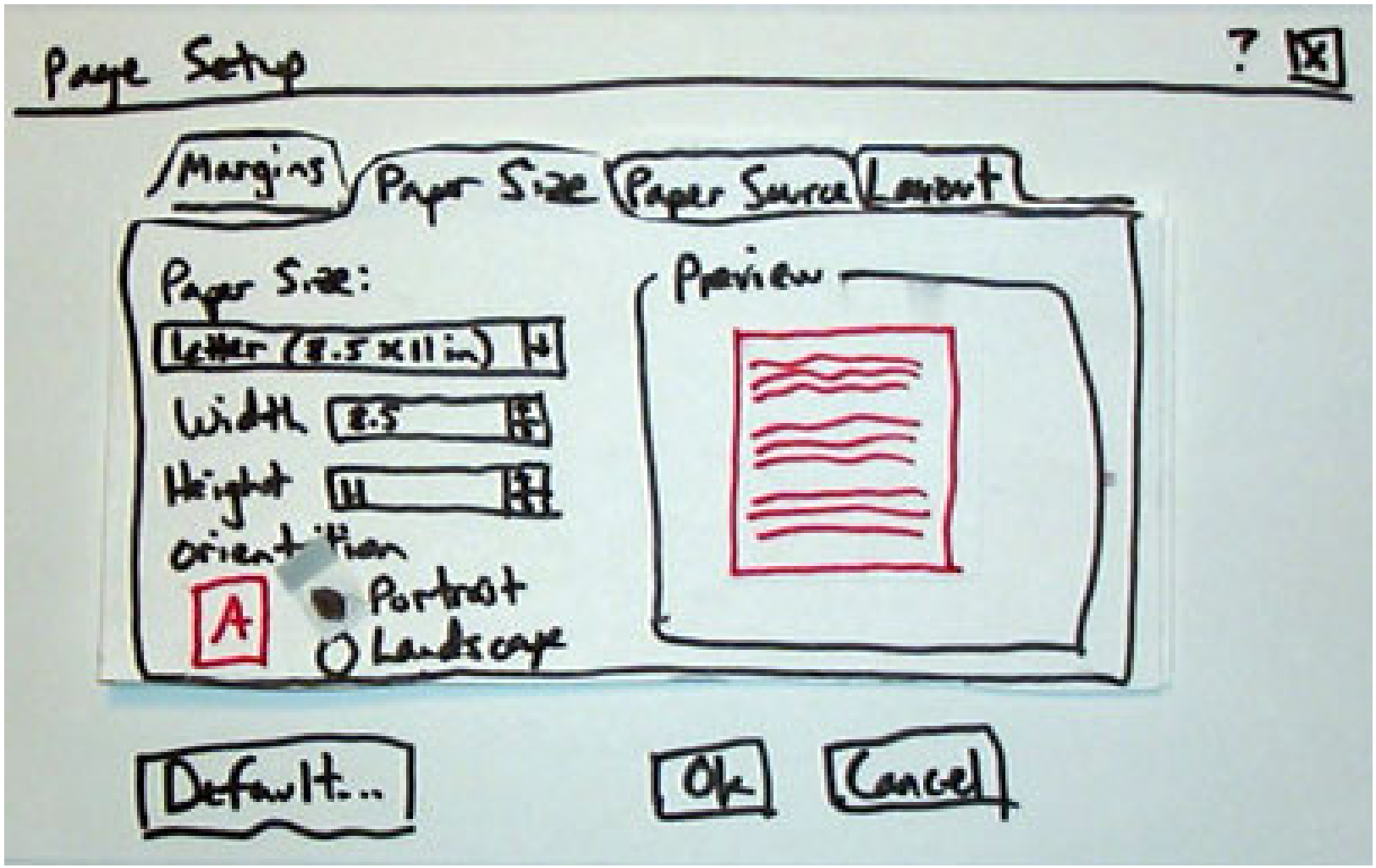


Interactive widgets

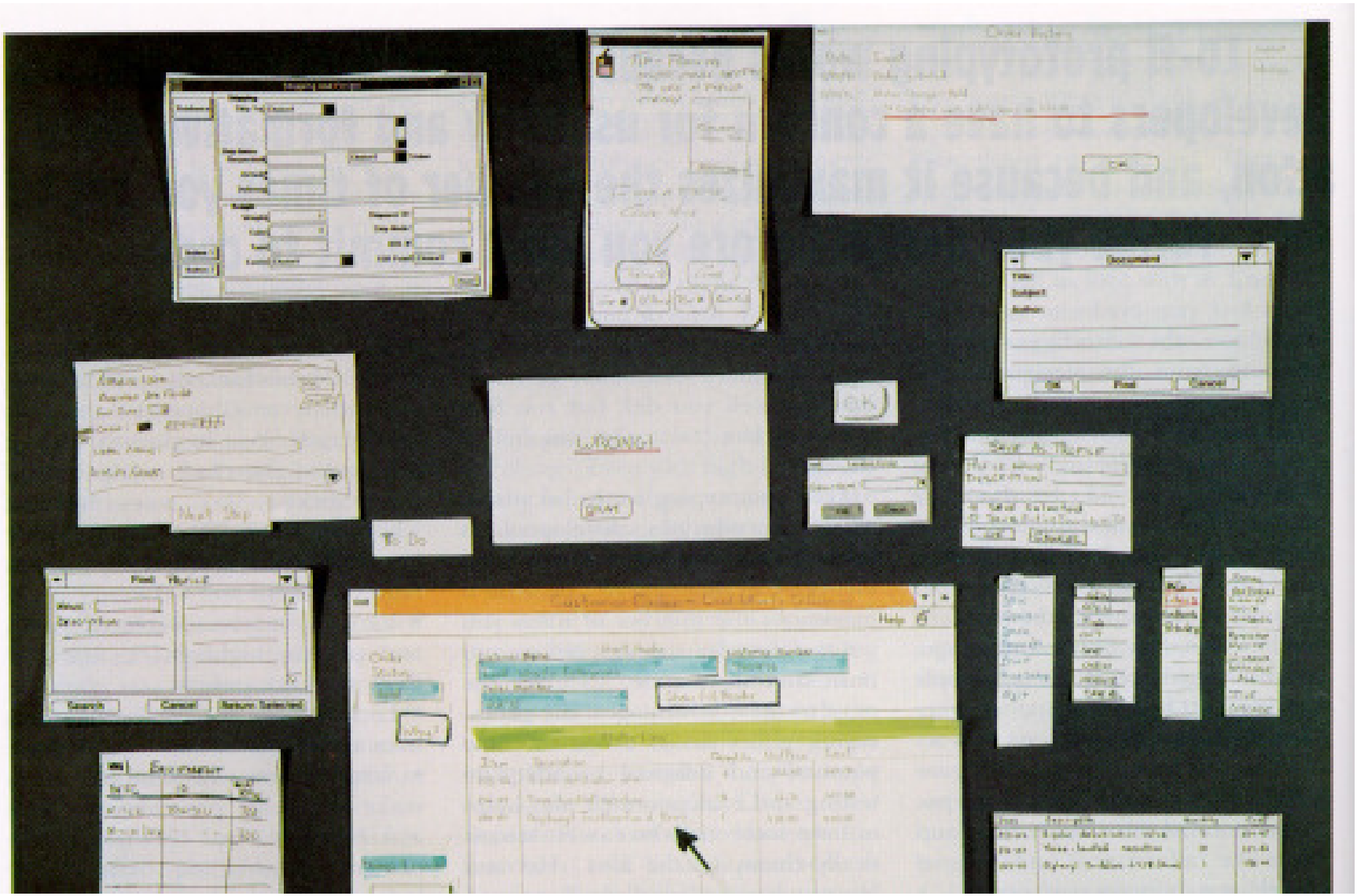
widget	how to simulate it
buttons or check boxes	tape
tabs and dialog boxes	index cards or small papers
text fields	removable tape
combo boxes	put the expanded choices on a separate paper / Post-It
selections	highlighted piece of tape
a disabled widget	cut out a separate gray version that can be placed on top of the normal one



Example paper prot. screen



Example paper prototype



Prototyping exercise

- Let's draw a prototype for a music player (e.g. iTunes).
  - Assume that the program lets you store, organize, and play songs and music videos.
  - Draw the main player UI and whatever widgets are required to do a **search for a song or video**.
  - After the prototypes are done, we'll try walking through each UI.

- Things to think about:
  - How many clicks are needed? What controls to use?
  - Could your parents figure it out without guidance?