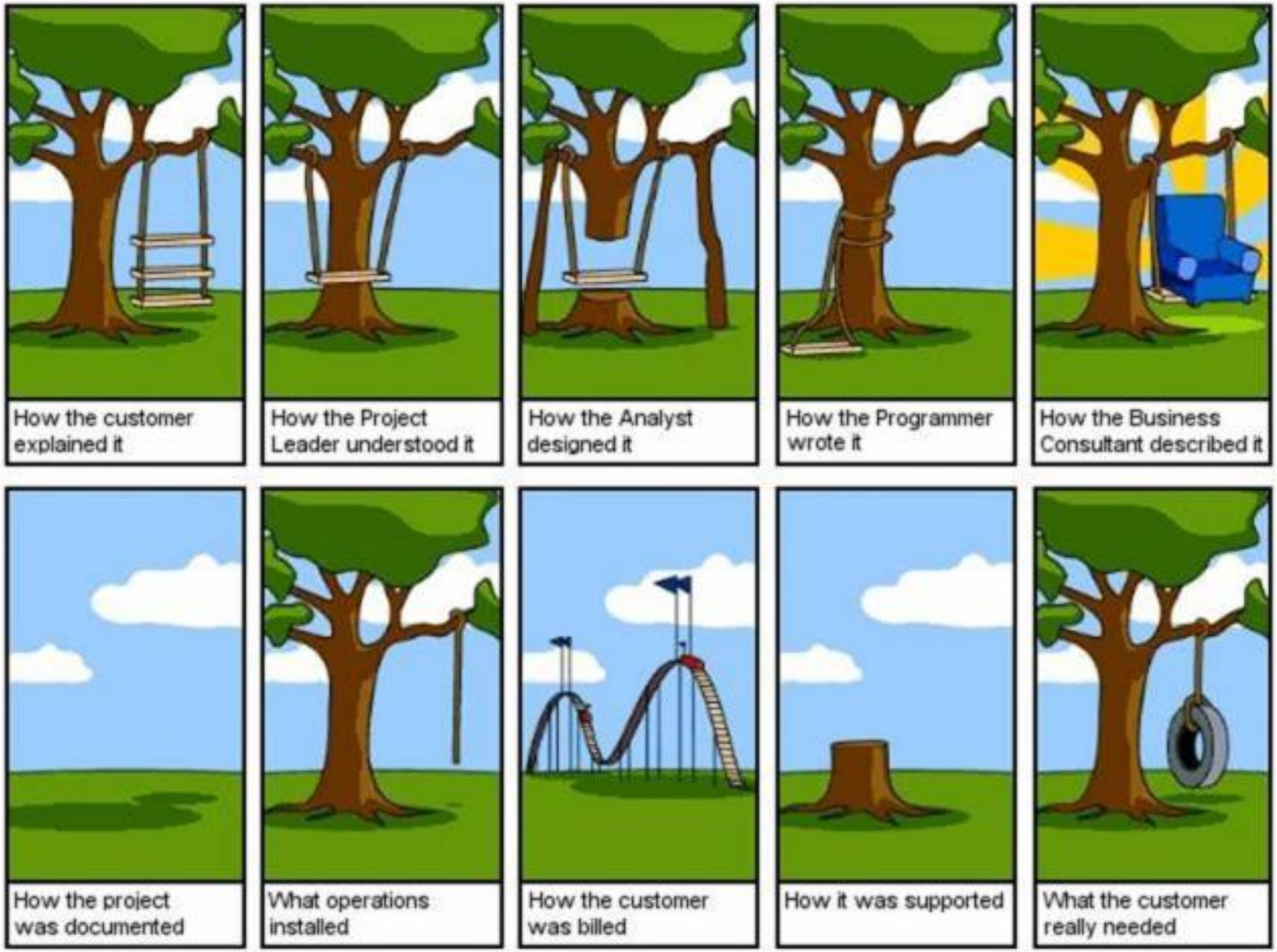
Requirements



Project group selection on Moodle

Discussion forum

News forum

Open All Close All

News Topic 1

Project milestones Topic 2

2 Idea Proposal

Idea Proposal Presentations

You can view everyone's the idea proposal presentations, to help you form project groups. (The presentations are all together, in a .zip file.)

Self-assemble project groups

Homework Assignments Topic 3

Project group selection on Moodle

Group	Group description	Count	Members	Action
Fighting Muskrats		0/4		Become member of Fighting Muskrats
Group 1		0/4		Become member of Group 1
Group Pi		0/4		Become member of Group Pi
Group Z		0/4		Become member of Group Z
Replacement Refs		0/4		Become member of Replacement Refs
Slurm		0/4		Become member of Slurm
The Bad Group		0/4		Become member of The Bad Group
This is not a group		0/4		Become member of This is not a group
Turing		0/4		Become member of Turing
Universe B		0/4		Become member of Universe B

Lecture outline

- Requirements
  - What are requirements?
  - How can we gather requirements?
  - How can we document requirements?
  - Use cases
- User interfaces
  - What's good?
  - What's bad?

What do requirements and UI have to do with research projects?

- The research is a product
  - You are making a tool, developing a technique, answering a question.
  - Users of your work are just like software users.
- The interface of your tool/technique is important to its success
- The interface of your work is important to your (project') success
  - reports
  - presentations

Lecture outline

→Requirements

- What are requirements?
- How can we gather requirements?
- How can we document requirements?
- Use cases

- User interfaces
  - What's good?
  - What's bad?

Software requirements

- **requirements**: specify what to build
  - "what" and not "how“
  - the system design, not the software design
  - the problem, not the (detailed) solution

“what vs. how”: it’s relative

- “One person’s what is another person’s how.”
  - “One person’s constant is another person’s variable.” [Perlis]

What	How
Parsing	Stack
Stack	Array or Linked List
Linked List	Doubly Linked List

Why requirements?

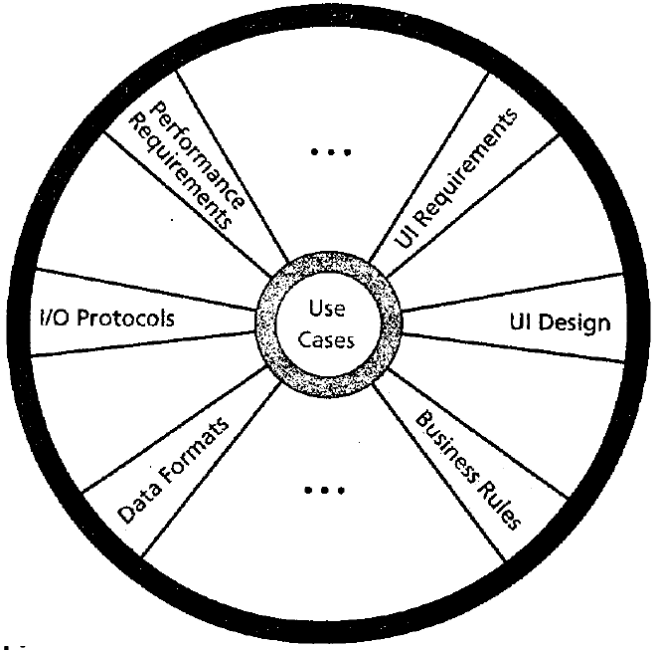
- Some goals of doing requirements:
  - understand precisely what is required of the software
  - communicate this understanding precisely to all development parties
  - control production to ensure that system meets specs (including changes)

- Roles of requirements
  - customers: show what should be delivered; contractual base
  - managers: a scheduling / progress indicator
  - designers: provide a spec to design
  - coders: list a range of acceptable implementations / output
  - QA / testers: a basis for testing, validation, verification

Cockburn's requirements list

Requirements Outline: A template of all functional requirements

1. purpose and scope
2. terms / glossary
3. **use cases**
4. technology used
5. other
   - 5a. development process - participants, values (fast-good-cheap), visibility, competition, dependencies
   - 5b. business rules / constraints
   - 5c. performance demands
   - 5d. security (now a hot topic), documentation
   - 5e. usability
   - 5f. portability
   - 5g. unresolved / deferred
6. human issues: legal, political, organizational, training

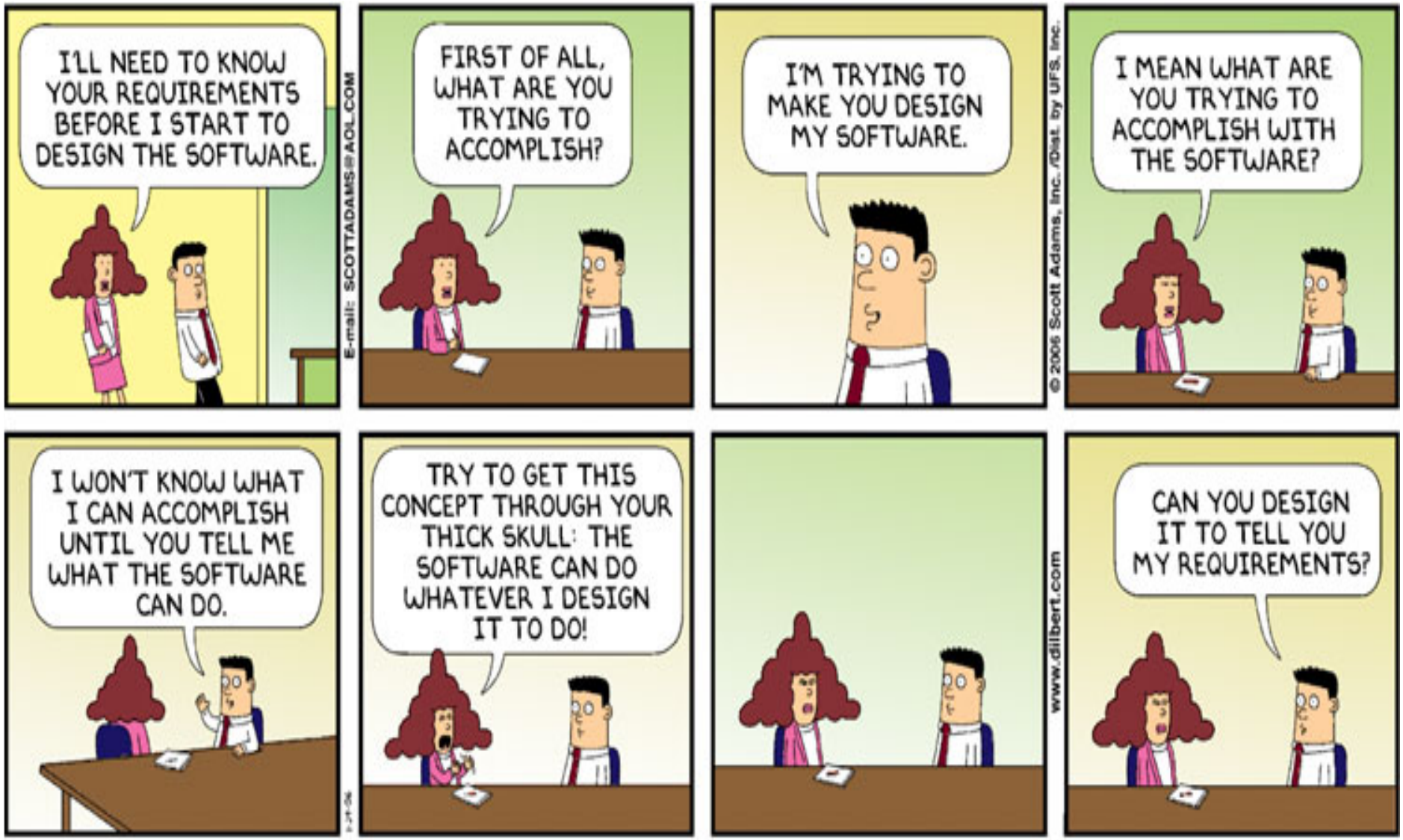


How do we gather requirements?

Let's start with two facts:

- Standish Group survey of over 8000 projects, the number one reason that projects succeed is user involvement
- Easy access to end users is one of three critical success factors in rapid-development projects (McConnell)

Typical situation



How do we specify requirements?

- Prototype
- Use cases
- List of features
- Paper (UI) prototype

- System Requirements Specification Document

A good use case

- starts with a request from an actor to the system
- ends with the production of all answers to the request
- defines the interactions (between system and actors) related to the function
- from the actor's point of view, not the system's
- focuses on interaction, not internal system activities
- doesn't describe the GUI in detail
- has 3-9 steps in the main success scenario
- is easy to read
- summary fits on a page

Use cases

A use case characterizes a way of using a system. It represents a dialog between a user and the system, from the user's point of view.

Example:
Jane has a meeting at 10AM; when Jim tries to schedule another meeting for her at 10AM, he is notified about the conflict

Use case terminology

Actor: someone who interacts with the system

Primary actor: person who initiates the action

Goal: desired outcome of the primary actor

Level: top or implementation

What are some possible actors?

Do use cases capture these?

Which of these requirements should be represented directly in a use case?

1. Order cost = order item costs × 1.06 (tax)
2. Promotions may not run longer than 6 months.
3. Customers only become Preferred after 1 year
4. A customer has one and only one sales contact
5. Response time is less than 2 seconds
6. Uptime requirement is 99.8%
7. Number of simultaneous users will be 200 max

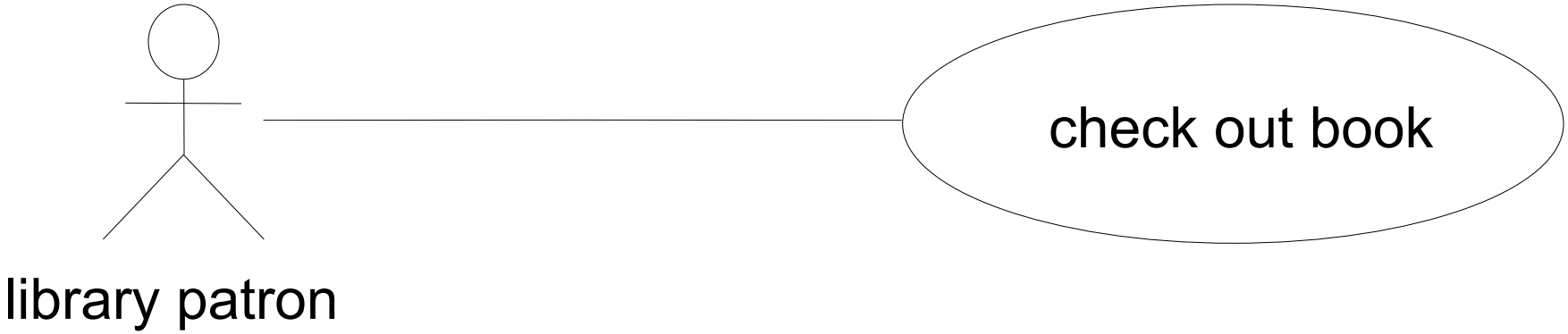
Three ways to write down use cases

- Diagrams
  - unified modeling language (UML)
- Informal language
- Formal specification

Use case summary diagrams

The overall list of your system's use cases can be drawn as high-level diagrams, with:

- actors as stick-men, with their names (nouns)
- use cases as ellipses with their names (verbs)
- line associations, connecting an actor to a use case in which that actor participates
- use cases can be connected to other cases that they use / rely on

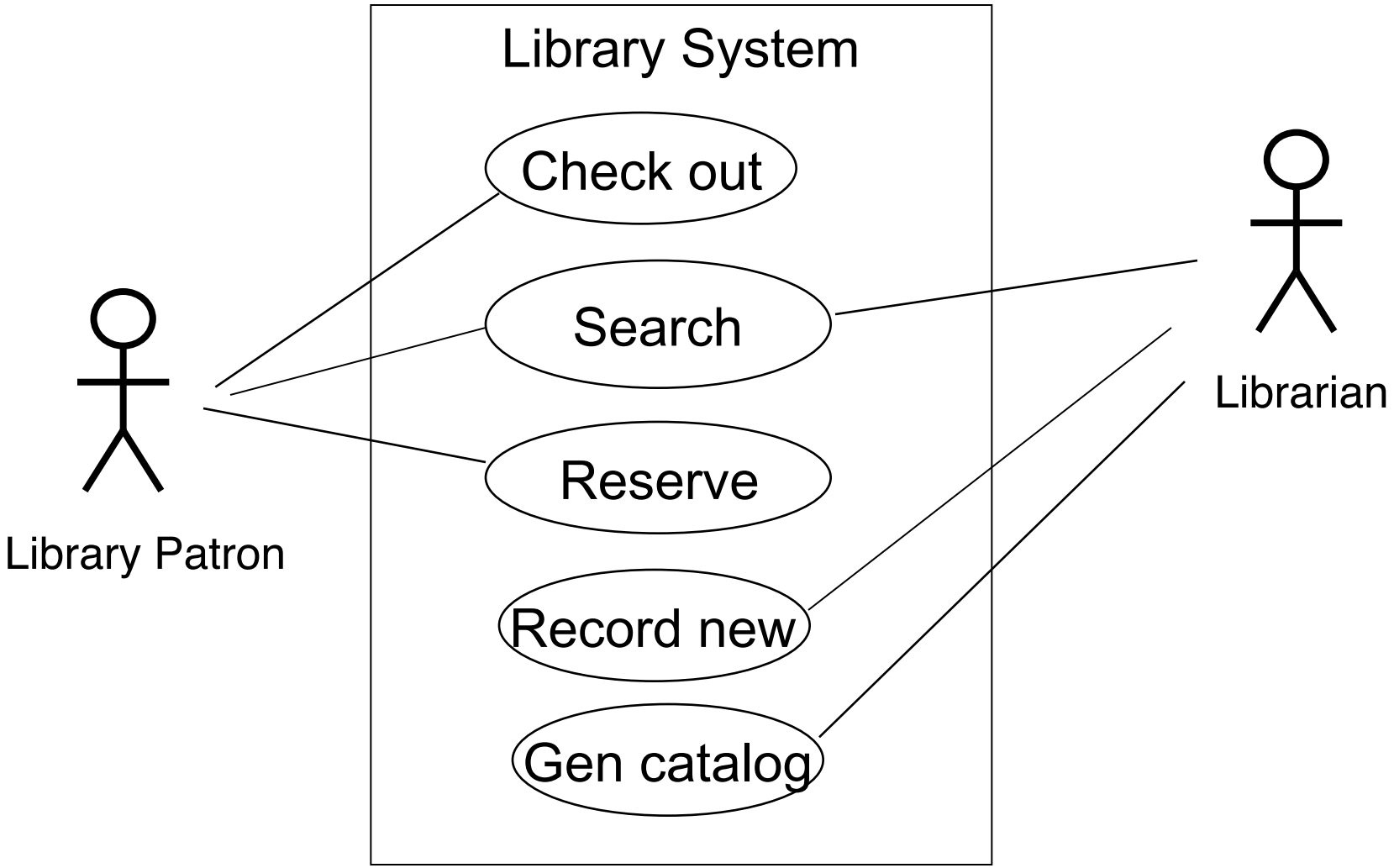


Use case summary diagrams

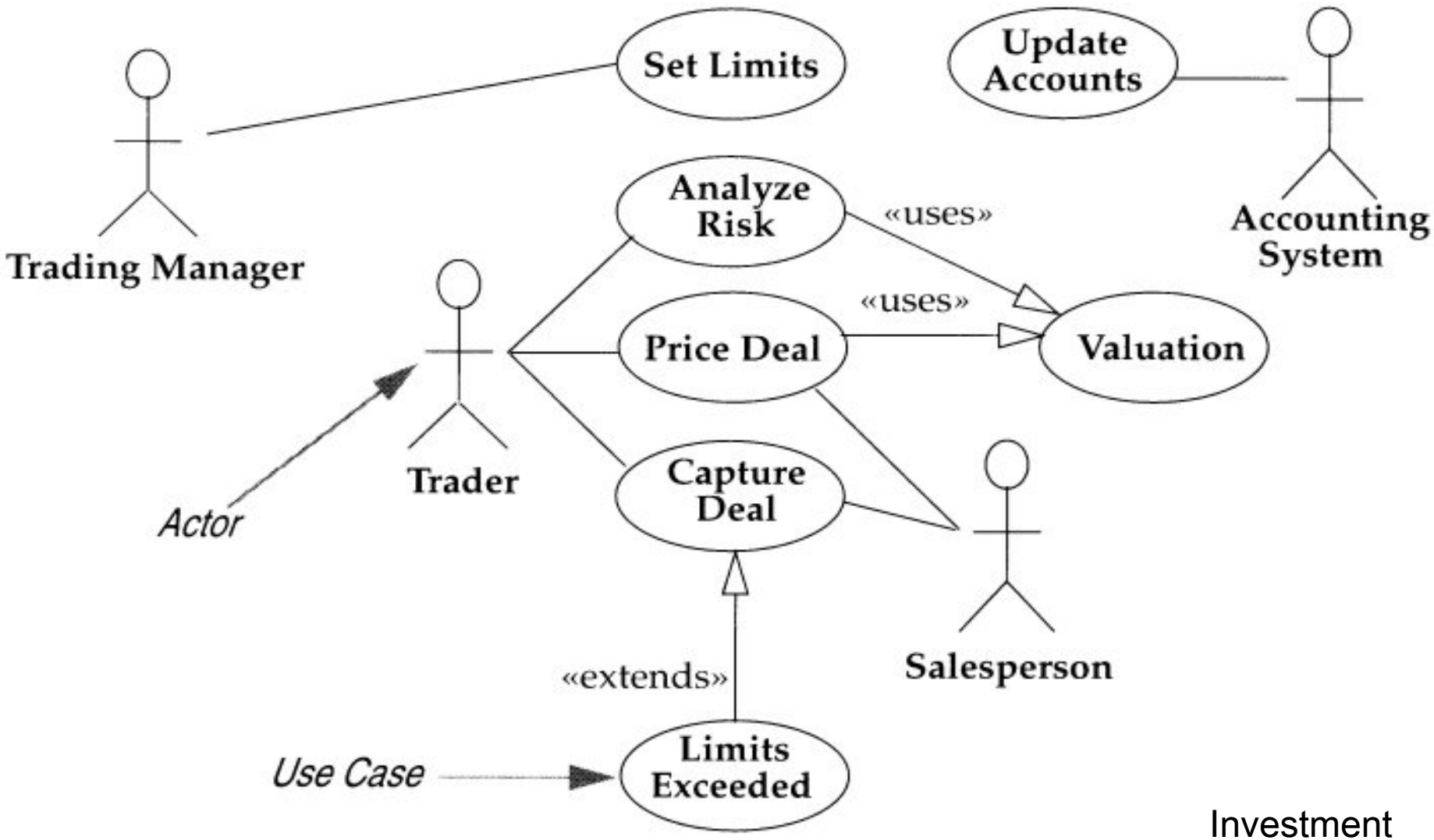
It can be useful to create a list or table of primary actors and their "goals"

Actor	Goal
Library Patron	Search for a book
	Check out a book
	Return a book
Librarian	Search for a book
	Check availability
	Request a book from another library

Use case summary diagram 1



Use case summary diagram 2



Investment System

Informal use case

Informal use case is written as a paragraph describing the scenario/interaction

- Example:
  - Patron Loses a Book
    The library patron reports to the librarian that she has lost a book. The librarian prints out the library record and asks patron to speak with the head librarian, who will arrange for the patron to pay a fee. The system will be updated to reflect lost book, and patron's record is updated as well. The head librarian may authorize purchase of a replacement tape.

Structured natural language

- I
  - I.A
    - I.A.ii
      - I.A.ii.3
        - » I.A.ii.3.q

Although not ideal, it is almost always better than unstructured natural language

Formal use case

Goal	**Patron wishes to reserve a book using the online catalog**
Primary actor	**Patron**
Scope	**Library system**
Level	**User**
Precondition	**Patron is at the login screen**
Success end condition	**Book is reserved**
Failure end condition	**Book is not reserved**
Trigger	**Patron logs into system**

Main Success Scenario	1. Patron enters account and password 2. System verifies and logs patron in 3. System presents catalog with search screen 4. Patron enters book title 5. System finds match and presents location choices to patron 6. Patron selects location and reserves book 7. System confirms reservation and re-presents catalog
Extensions (error scenarios)	2a. Password is incorrect 2a.1 System returns patron to login screen 2a.2 Patron backs out or tries again 5a. System cannot find book 5a.1 …
Variations (alternative scenarios)	4. Patron enters author or subject

Steps to creating a use case

- Identify actors and their goals
- Write the success scenario
  - identify happy path
- List the failure extensions
  - almost every step can fail
- List the variations
  - forks in the scenario

Jacobson example: recycling

The course of events starts when the customer presses the "Start-Button" on the customer panel. The panel's built-in sensors are thereby activated.

The customer can now return deposit items via the customer panel. The sensors inform the system that an object has been inserted, they also measure the deposit item and return the result to the system.

The system uses the measurement result to determine the type of deposit item: can, bottle or crate.

The day total for the received deposit item type is incremented as is the number of returned deposit items of the current type that this customer has returned…

Another example: buy a product

http://ontolog.cim3.net/cgi-bin/wiki.pl?UseCasesSimpleTextExample

1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

- Alternative: **Authorization Failure**
  - At step 6, system fails to authorize credit purchase
  - Allow customer to re-enter credit card information and re-try
- Alternative: **Regular Customer**
  - 3a. System displays current shipping information, pricing information, and last four digits of credit card information
  - 3b. Customer may accept or override these defaults
  - Return to primary scenario at step 6

Pulling it all together

How much is enough?

You have to find a balance.

comprehensible vs. detailed

graphics vs. explicit wording and tables

short and timely vs. complete and late

Your balance may differ with each customer depending on your relationship and flexibility

Lecture outline

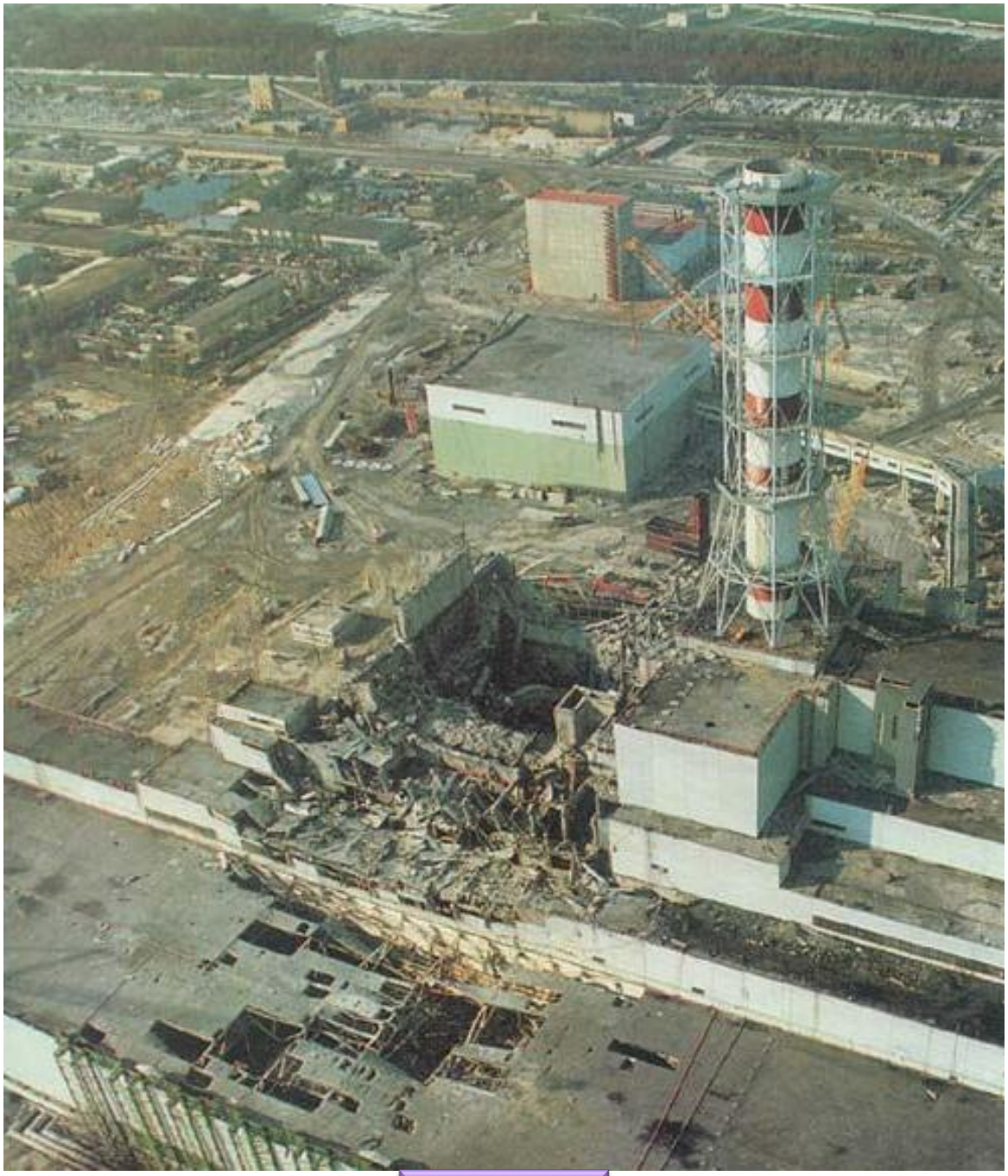
- Requirements
  - What are requirements?
  - How can we gather requirements?
  - How can we document requirements?
  - Use cases

→User interfaces

  - What's good?
  - What's bad?

User Interface

Three Mile Island



Chernobyl

How do we avoid bad UI?

- Learn from past mistakes
- Build prototypes

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 software design

- **usability**: the effectiveness of users achieving tasks
  - Human-Computer Interaction (HCI).
  - Usability and good UI design are closely related.
  - A bad UI can have serious results…

Achieving usability

- User testing and field studies
  - having users use the product and gathering data
- Evaluations and reviews by UI experts
- Prototyping
  - Paper prototyping
  - Code prototyping

- Good UI design focuses on the *user*

  not on the developer, not on the system environment

Prototyping

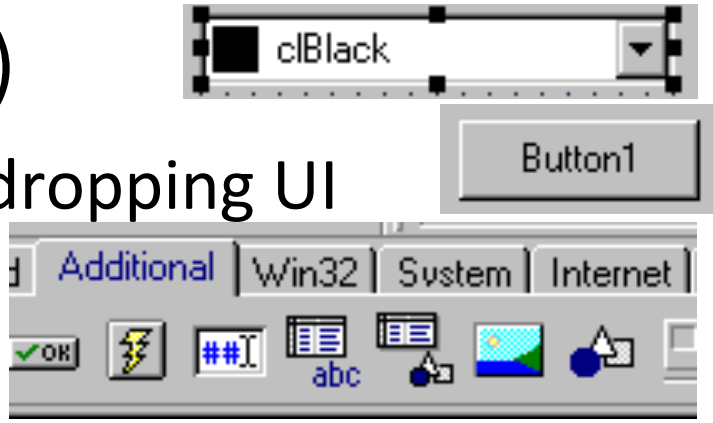
- **prototyping**: Creating a scaled-down or incomplete version of a system to demonstrate or test its aspects.

- Reasons to do prototyping:
  - aids UI design
  - provides basis for testing
  - team-building
  - allows interaction with user to ensure satisfaction

Some prototyping methods

1. UI builders (Visual Studio, ...)

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



2. implementation by hand

   writing a "quick" version of your code

3. **paper prototyping**: a paper version of a UI

Why do paper prototypes?

- 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
- feels less permanent or final

Where does paper prototyping fit?

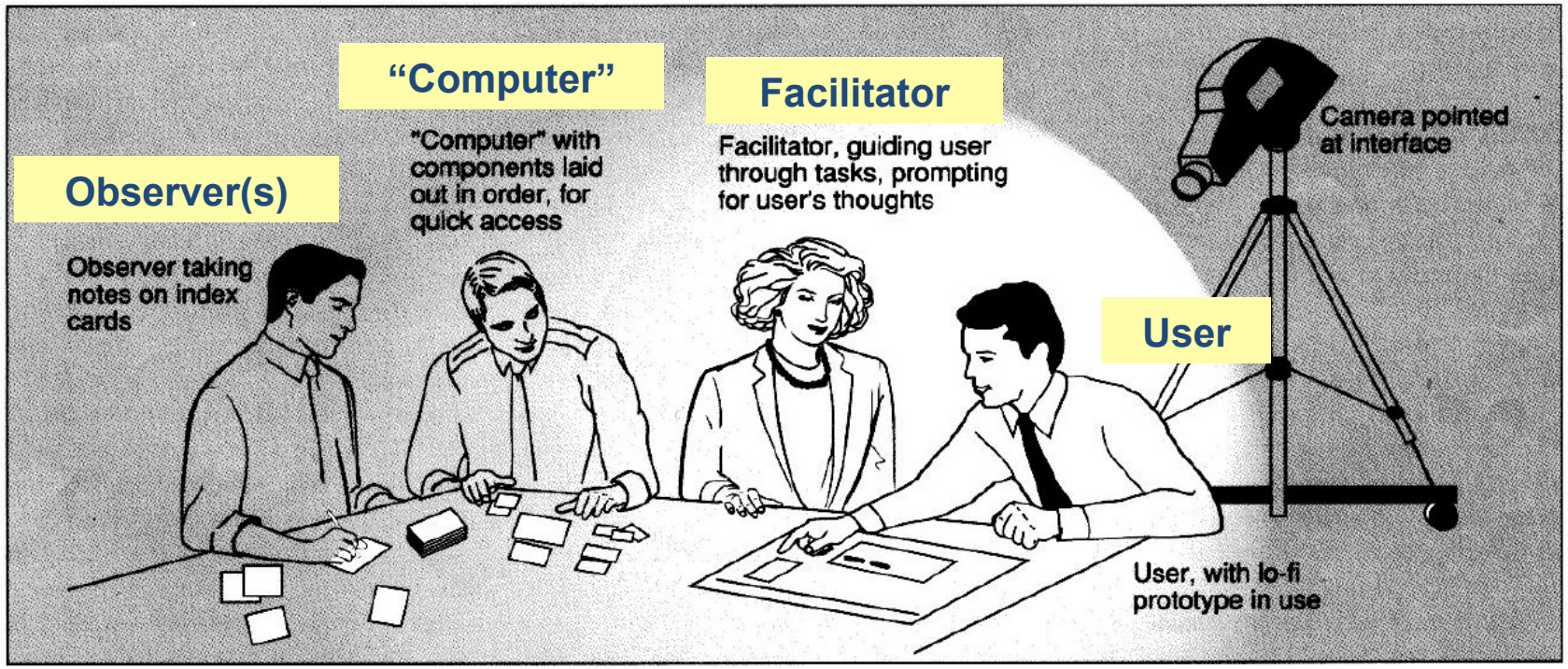
When in the software lifecycle is it most useful to do (paper) prototyping?

- Requirements are the what and design is the how. Which is paper prototyping?

- Prototyping
  - helps uncover requirements and upcoming design issues
  - during or after requirements but before design
  - shows us what is in the UI, but also shows us details of how the user can achieve goals in the UI

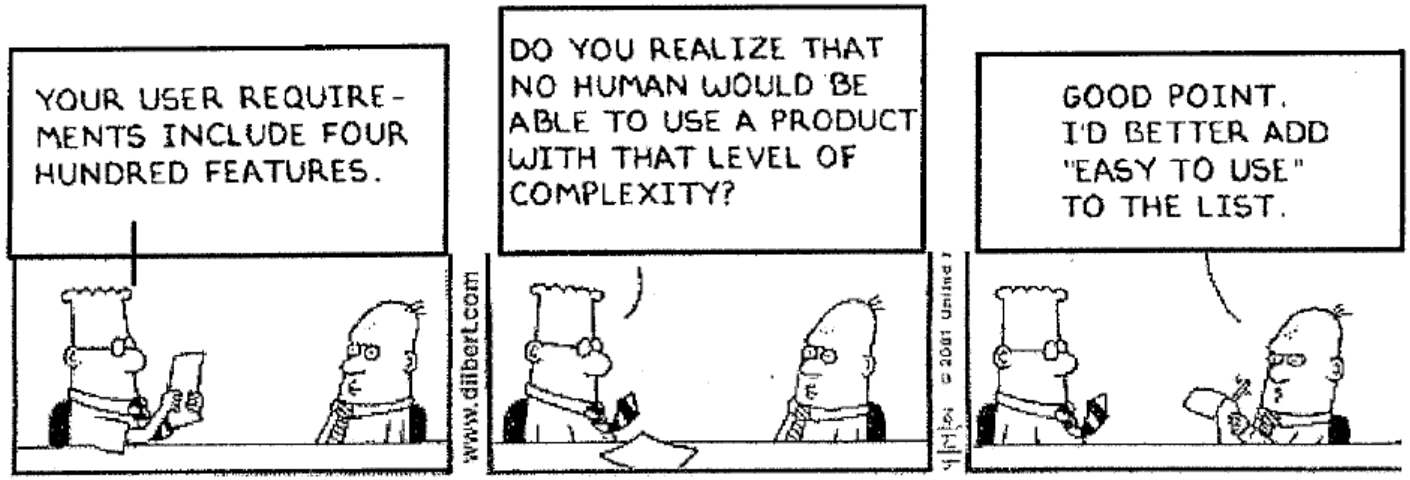
Paper prototyping usability session

- user gets tasks to perform on a paper prototype
- observed by people and/or recorded
- a developer can "play computer"



Schneiderman's 8 Golden Rules

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



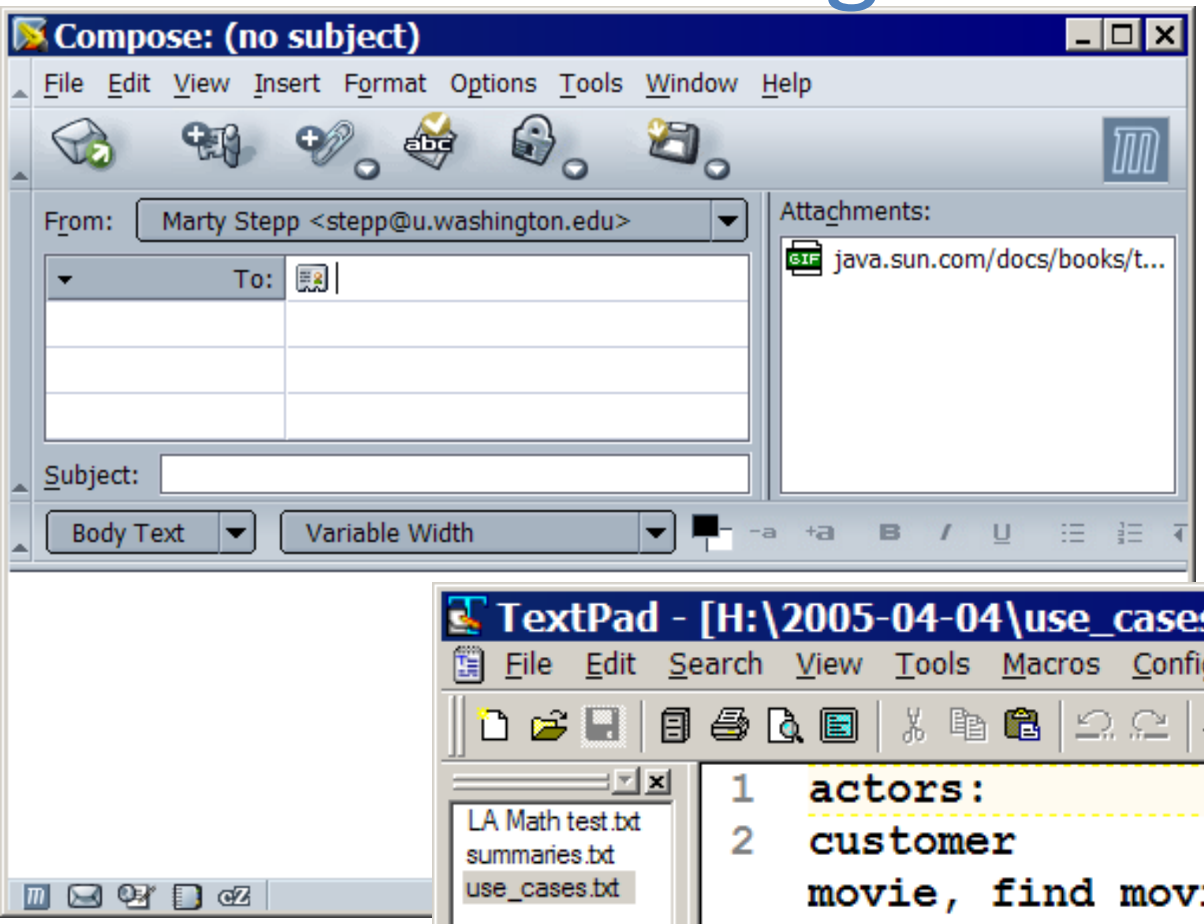
(from Designing the User Interface, by Ben Schneiderman of UMD, noted HCI and UI design expert)

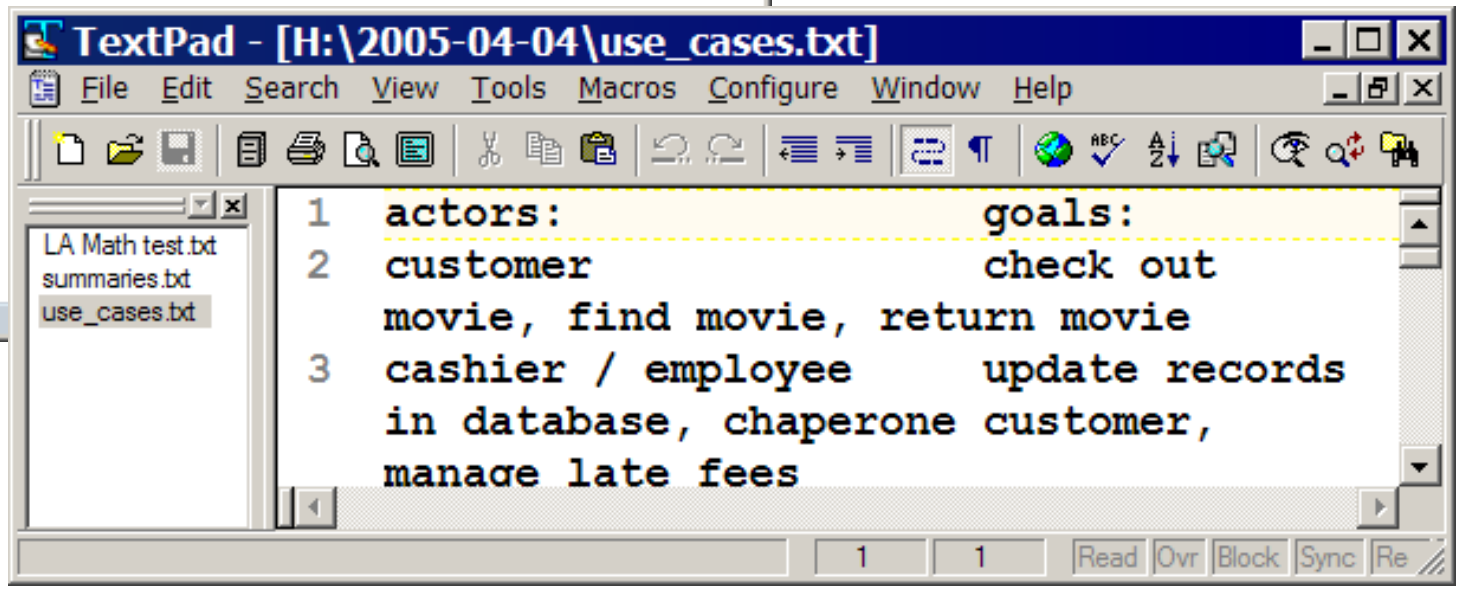
Ben Schneiderman

- Coming to give a talk here!
- Monday, Oct 1
- 12 noon
- CS 150

- Information Visualization for Medical Informatics

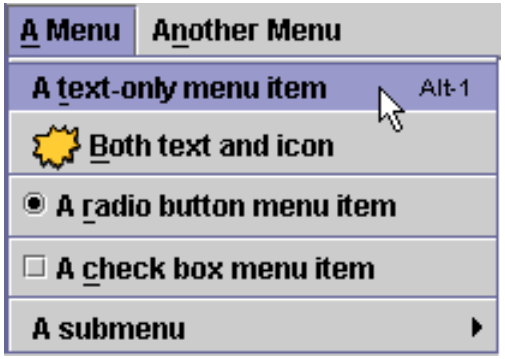
UI design examples

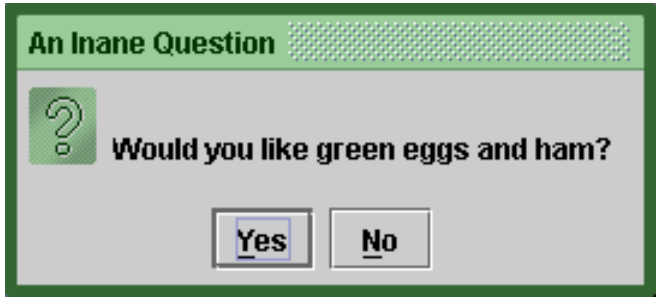


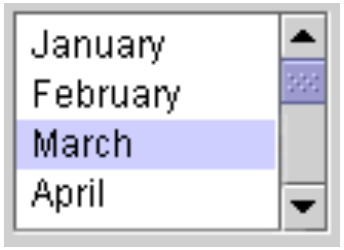


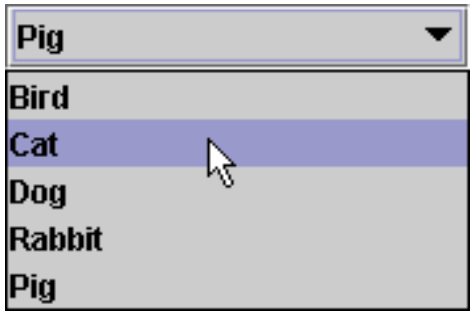
UI design, components

- 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..?



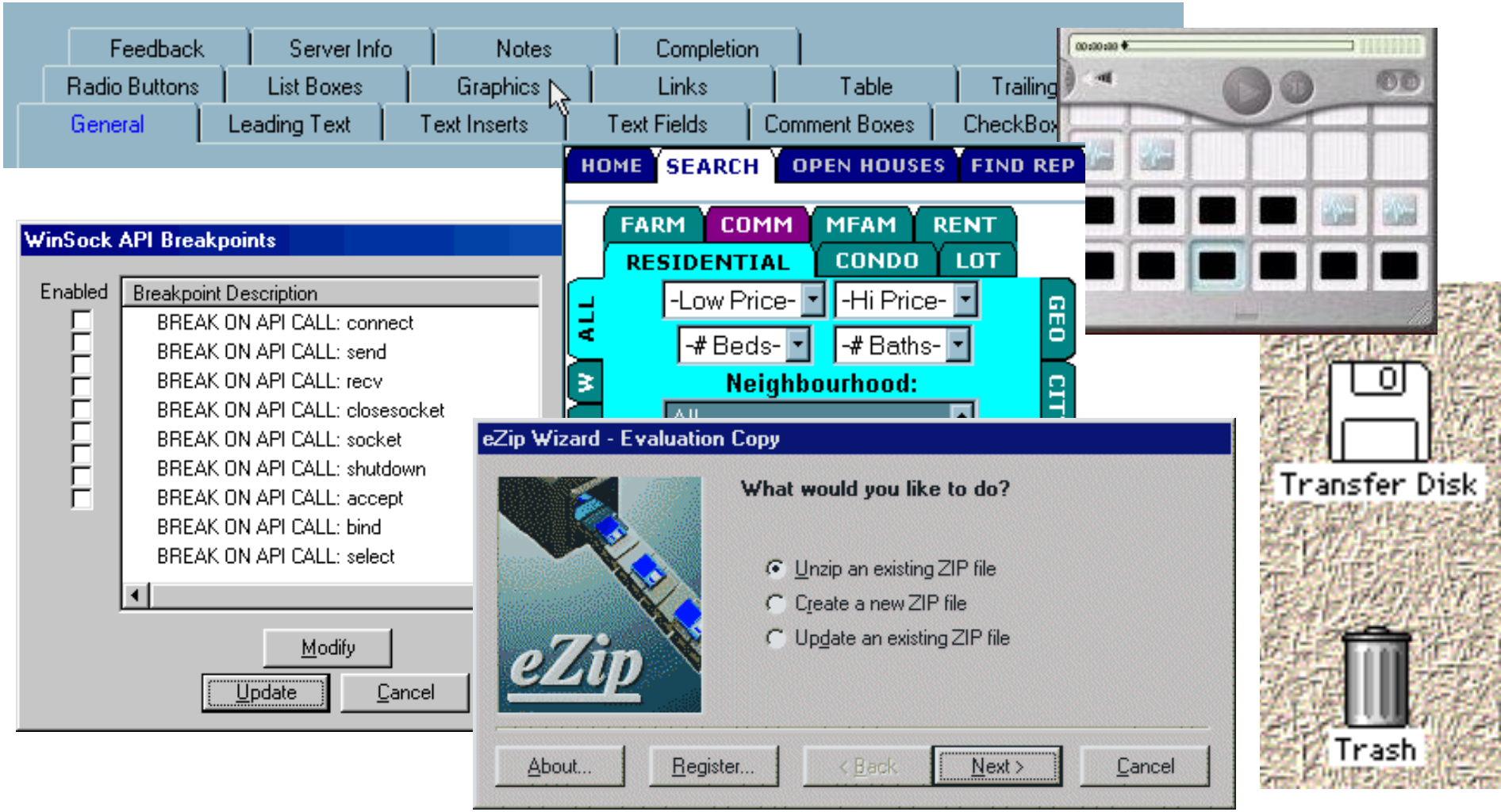




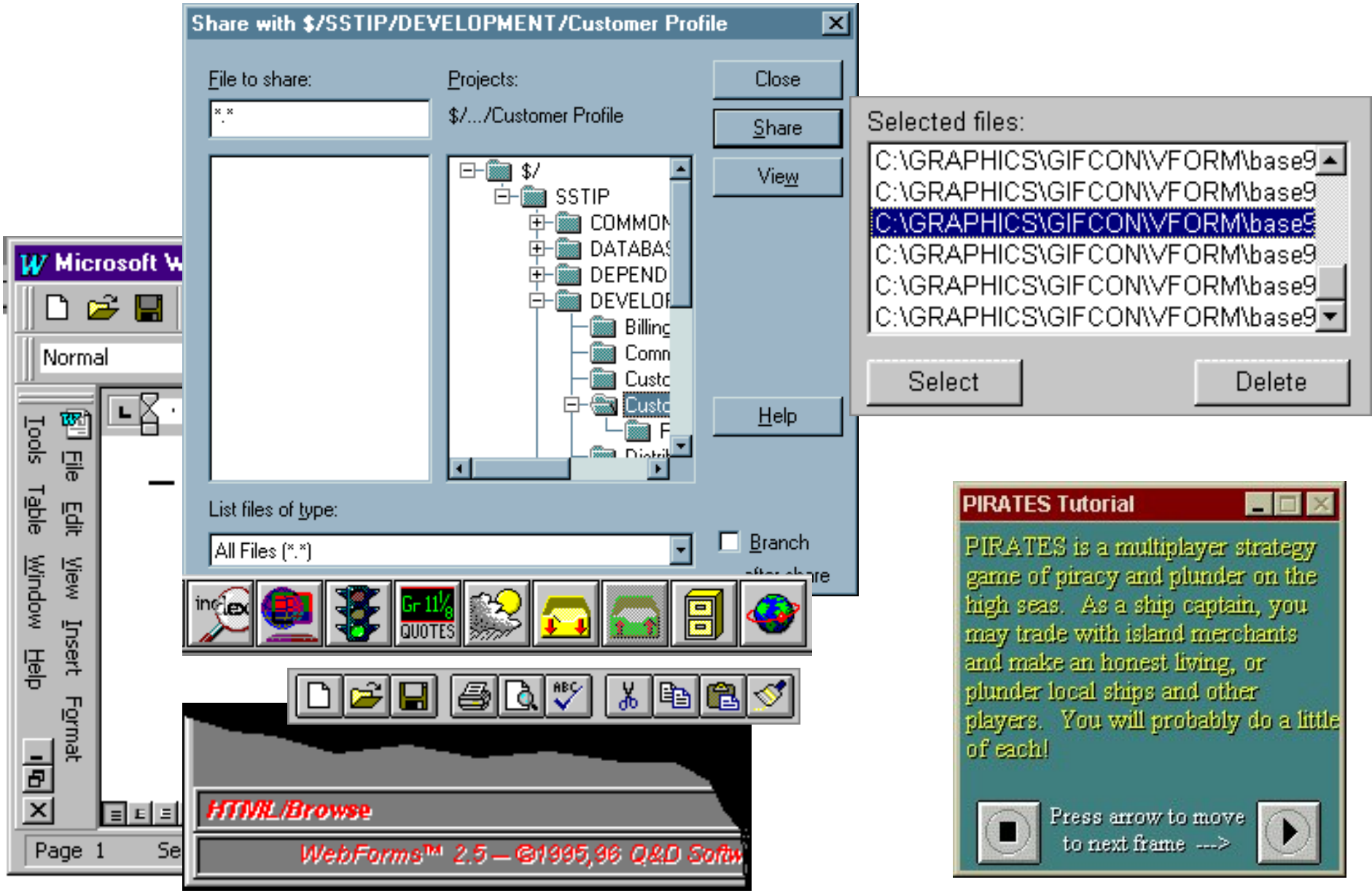


UI Hall of Shame

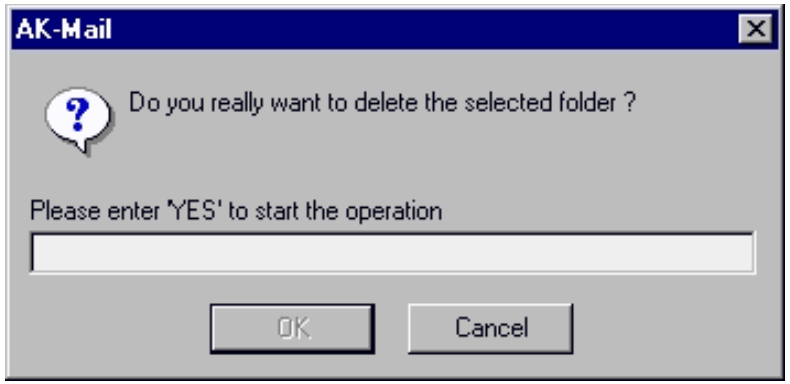
http://homepage.mac.com/bradster/iarchitect/shame.htm

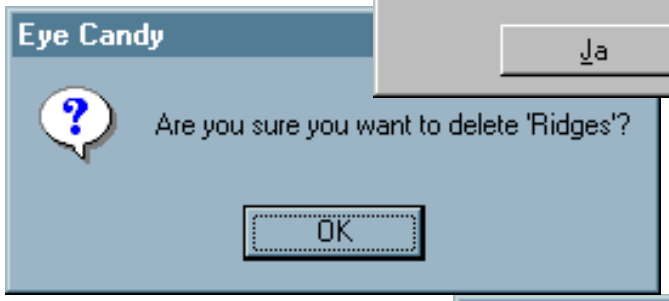


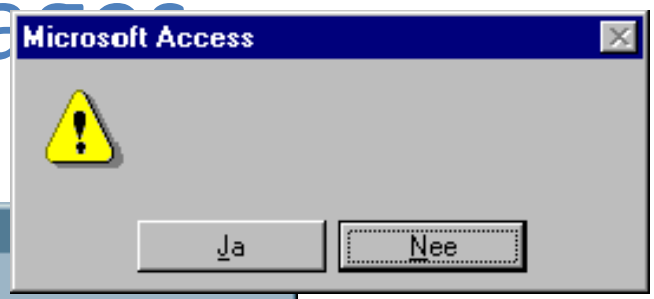
Layout and color

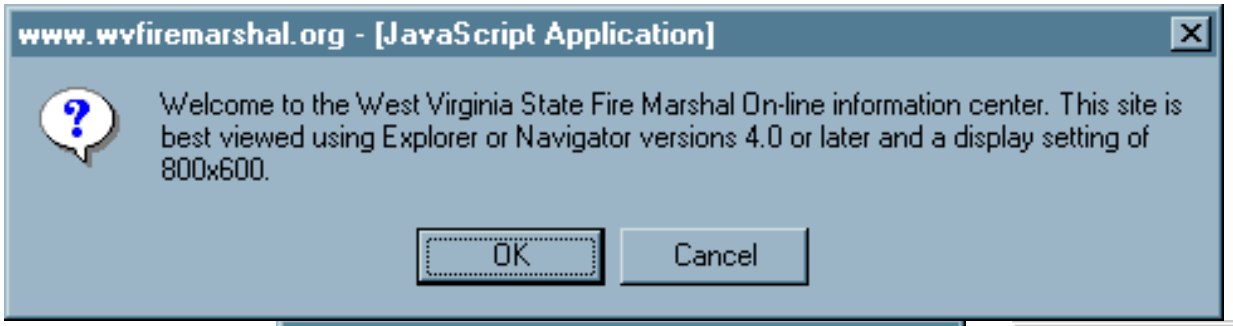


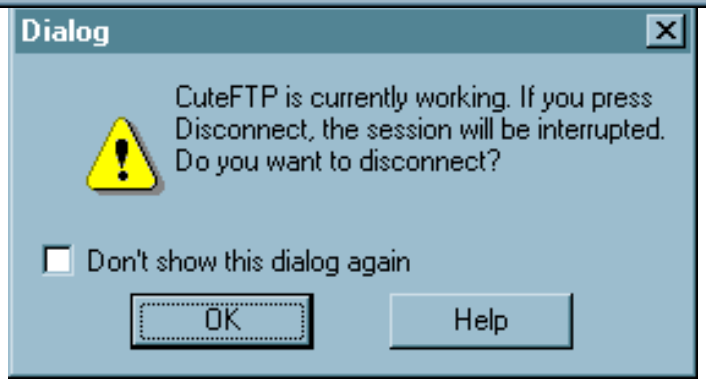
Bad error messages

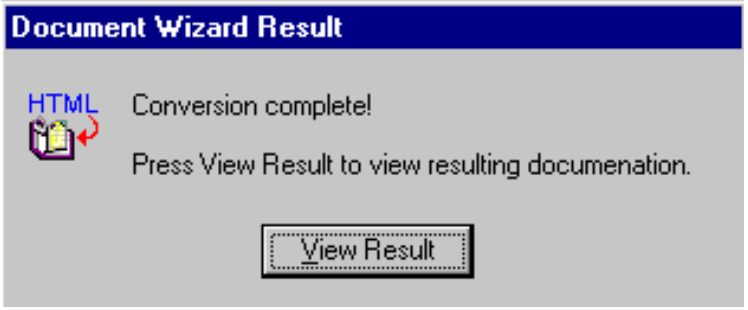








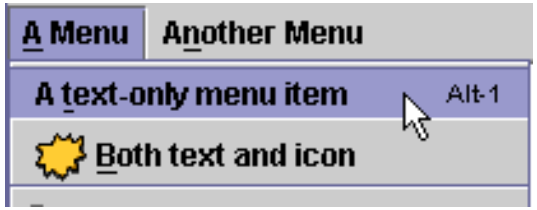




UI design – buttons, menus

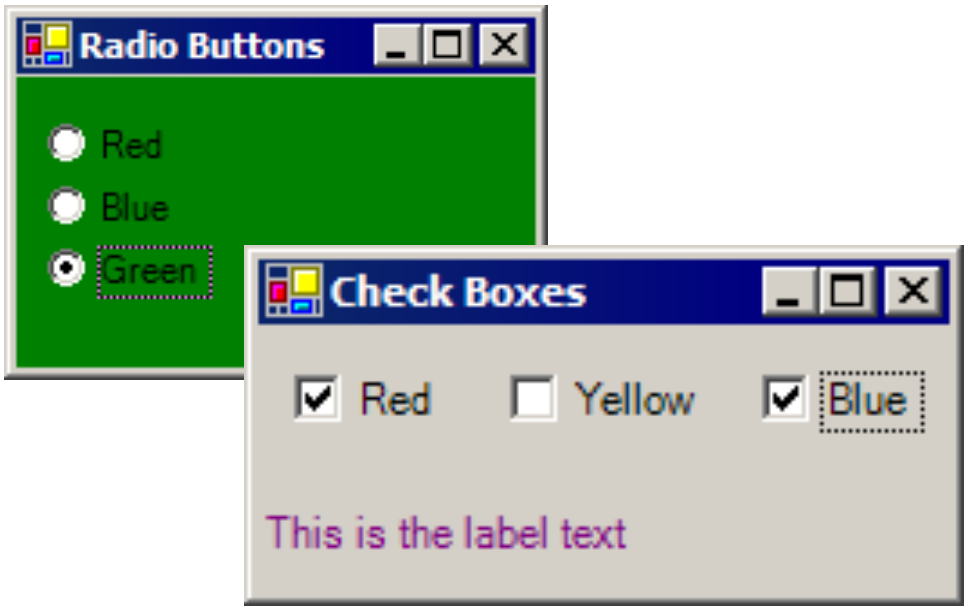
- Use **buttons** for single independent actions that are relevant to the current screen.
  - Try to use button text with verb phrases such as "Save" or "Cancel", not generic: "OK", "Yes", "No"
  - use Mnemonics or Accelerators (Ctrl-S)

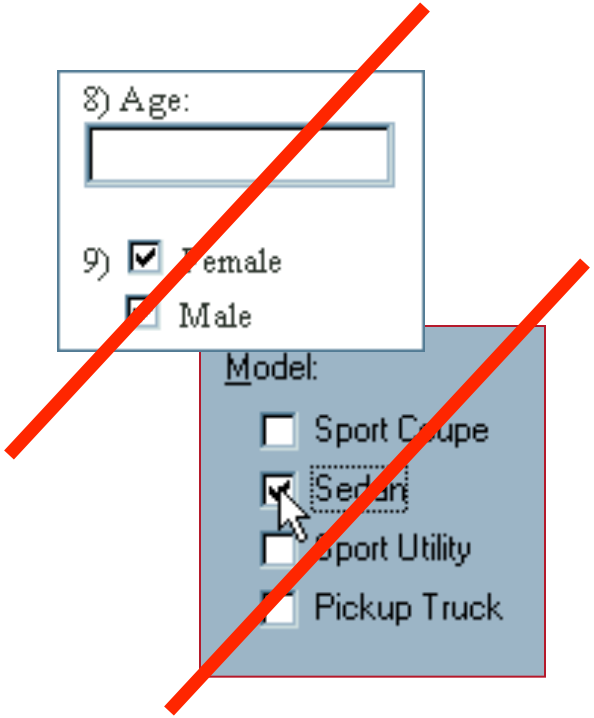
- Use **toolbars** for common actions.
- Use **menus** for infrequent actions that may be applicable to many or all screens.
  - *Users hate menus!* Try not to rely too much on menus. Provide another way to access the same functionality (toolbar, hotkey, etc.)



UI design – checkboxes, radio buttons

- Use **check boxes** for independent on/off switches
- Use **radio buttons** for related choices,
  when only one choice can be activated at a time

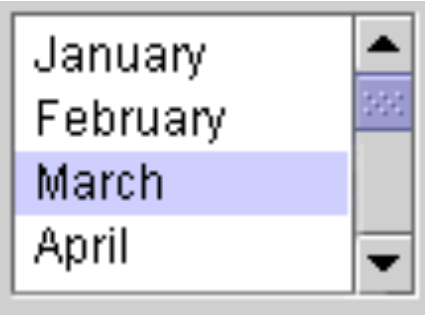




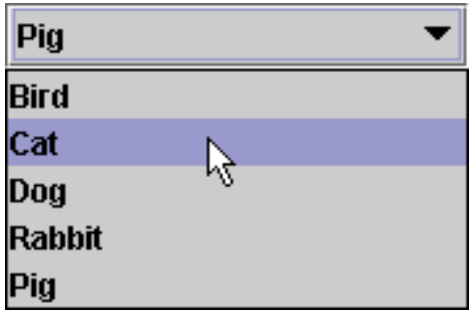
UI design – lists, combo boxes

- use **text fields** (usually with a label) when the user may type in anything they want

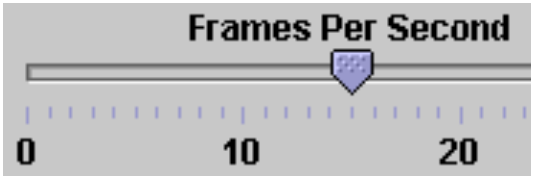
- use **lists** when there are many fixed choices (too many for radio buttons); *all* choices visible on screen at once

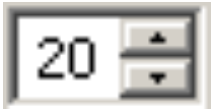


- use **combo boxes** when there are many fixed choices; don't take up screen real estate by showing them all at once



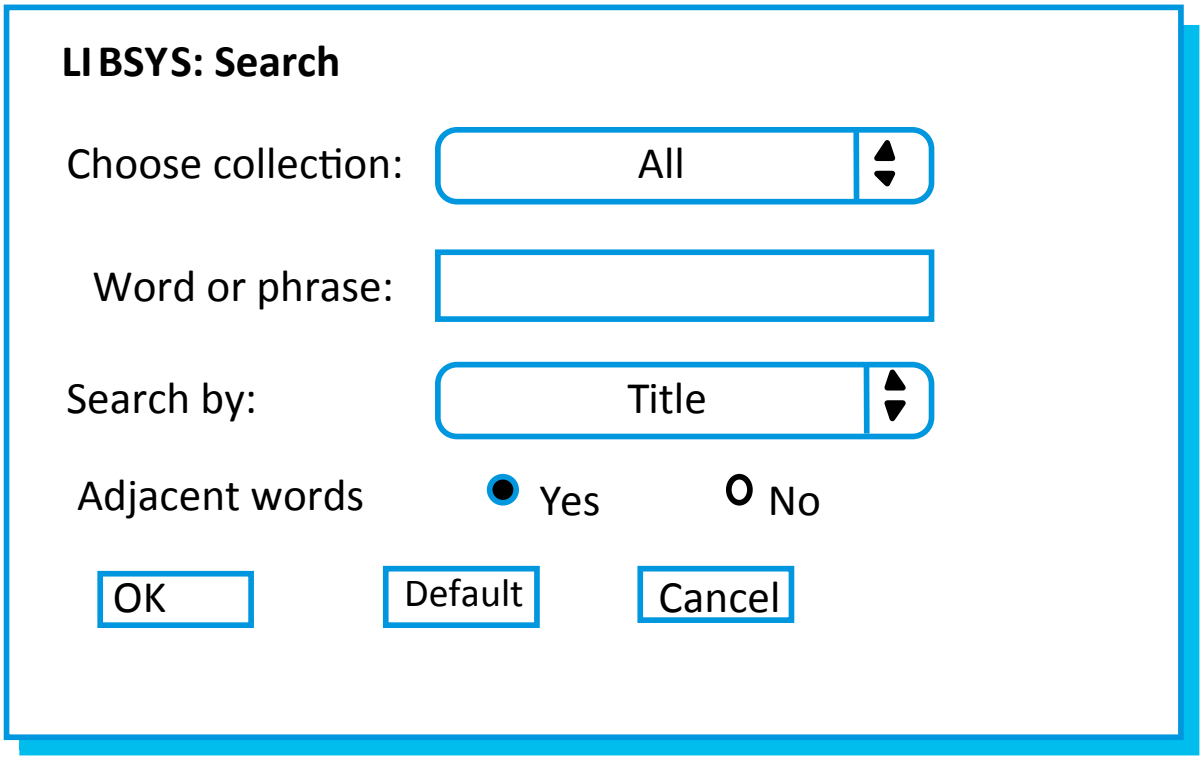
- use a **slider** or **spinner** for a numeric value





An example UI

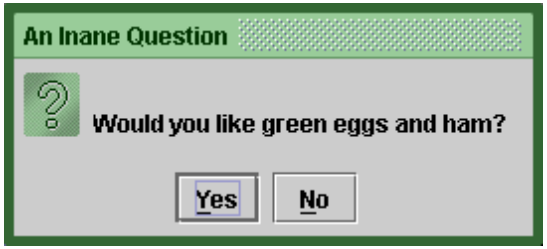
- Good UI dialog?
  Did the designer choose the right components?
  assume there are 20 collections and 3 ways to search



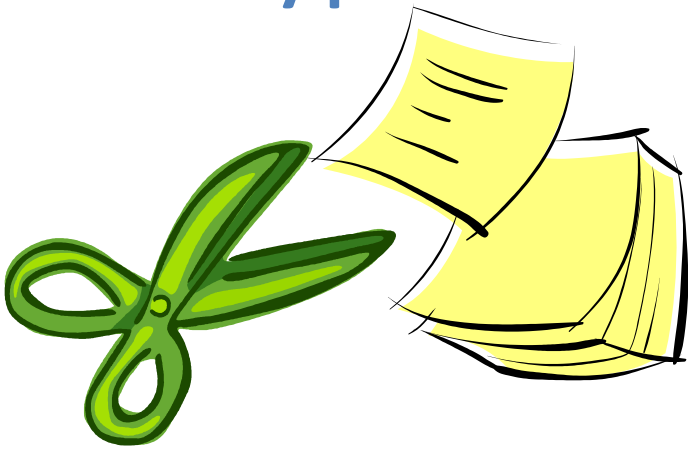
UI design – multiple screens

- use a **tabbed pane** when there are many screens that the user may want to switch between at any moment

- use **dialog boxes** or **option panes** to present temporary screens or options



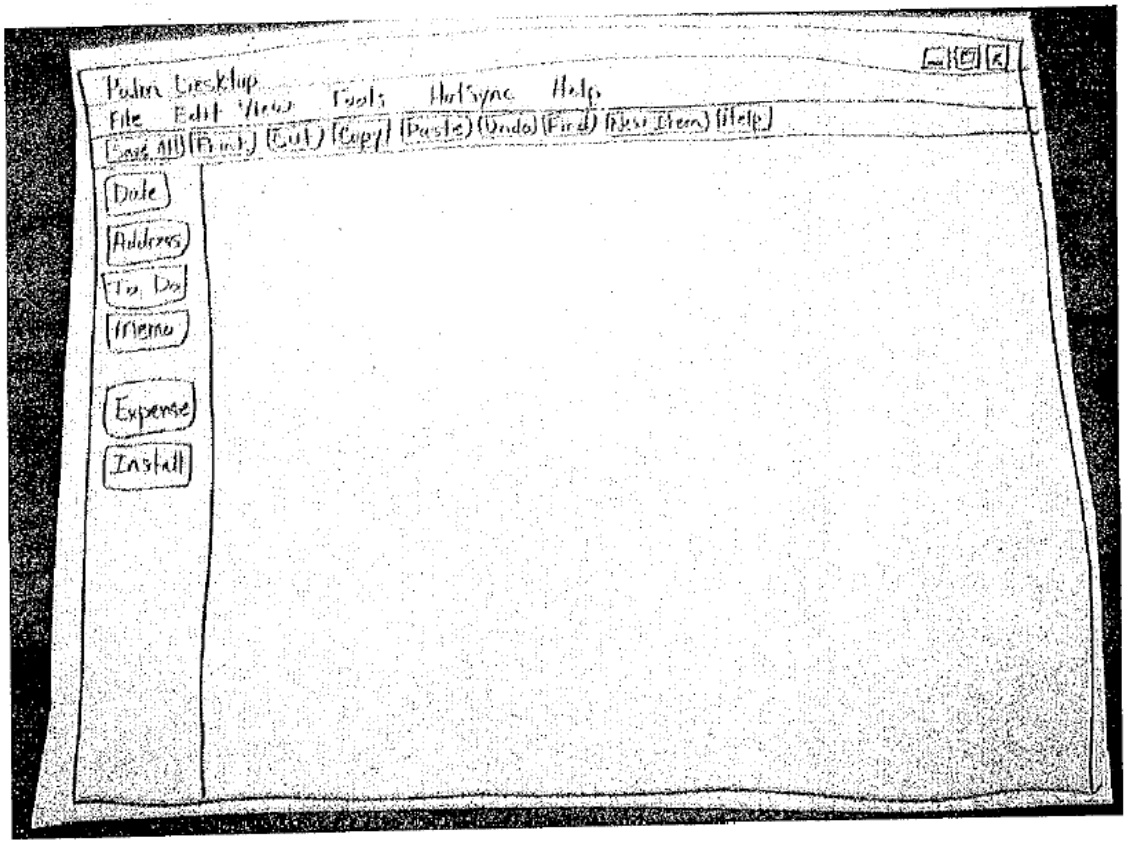
Creating a paper prototype



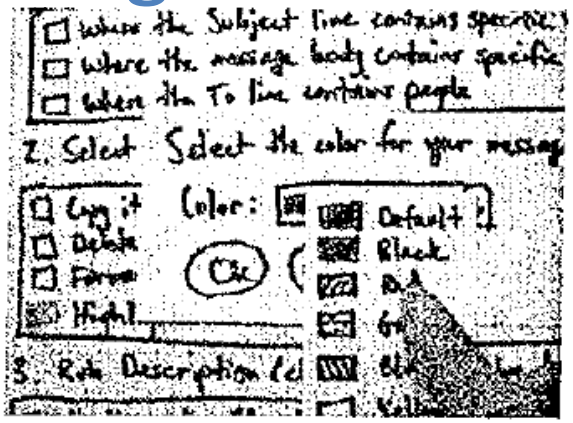
- gather materials
  - paper, pencils/pens
  - tape, scissors
  - highlighters, transparencies
- identify the screens in your UI
  - consider use cases, inputs and outputs to user
- think about how to get from one screen to next
  - this will help choose between tabs, dialogs, etc.

Application backgrounds

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



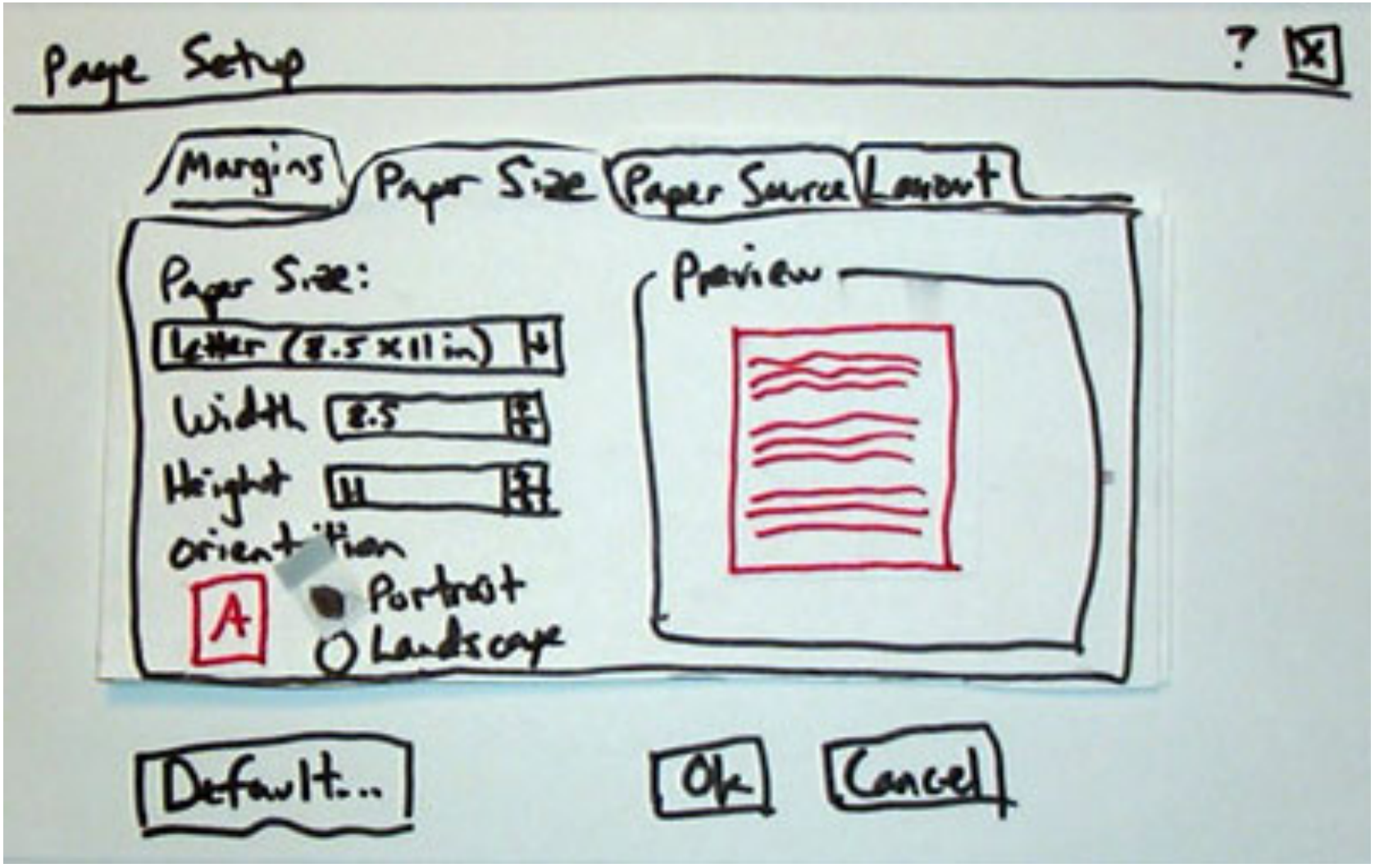
Representing interactive widgets



- buttons / check boxes: tape
- tabs, dialog boxes: index cards
- text fields: removable tape
- combo boxes: put the choices on a separate piece of paper that pops up when they click
- selections: a highlighted piece of tape or transparency
- disabled widgets: make a gray version that can sit on top of the normal enabled version

- computer beeps: say "beep"

Example paper prototype screen



Prototyping exercise

- In your project groups, draw a rough prototype for a music player (e.g., WinAmp or 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 together.

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