Lecture 4

Use Cases and prototyping

Reading:

Writing Effective Use Cases, A. Cockburn (alistair.cockburn.us/get/2465) – MUST READ (Examples)

http://www.cragsystems.co.uk/use_case_tutorial/ == Recommended

Ian Summorvillie, Software Engineering, Chapter 4 (Further Reading)

Outline

- What is a use case?
- Terminology
- Styles of use cases
- Steps for creating a use case
- User interface Prototyping

Use cases

- **Functional Requirements**
  - Tell us what the user will do with the system.
  - Not enough details for developers
  - Need a way to describe the sequence of interactions to accomplish a functional requirement.

- **use case**: a written description of the *user*'s *interaction* with the *software* product to accomplish a *goal*
  - It is an **example behavior** of the system.
    - *clearly written steps lead to a "main success scenario"*
    - written from actor's point of view, not the system's

- Use cases capture **functional requirements**
  - Not necessary one to one mapping between them

Benefits of doing use cases?

- Establish an understanding between the customer and the system developers of the detailed requirement
  - **success scenarios**

- Alert developers of problematic situations, error cases to test
  - **extension scenarios**

- Capture a level of functionality to plan around
  - **list of goals**

Qualities of a good use case

- Focuses on interaction
  - *Starts* with a *request* from an actor to the system.
  - *Ends* with the production of *all the answers* to the request.

- Focuses on essential behaviors, from the *actor's point of view*
  - Does not describe internal system activities.
  - Does not describe the GUI in detail.

- Concise, clear, accessible to non-programmers
  - Easy to read.
  - Summary fits on a page.
  - Main success scenario and extensions.

Use cases vs. internal features

Use Cases

- call someone
- receive a call
- send a message
- memorize a number

Point of view: user

Internal Functions

- transmit / receive data
- energy (battery)
- user I/O (display, keys, ...)
- phone-book mgmt.

Point of view: developer / designer

Use cases and requirements

- Which of these requirements would probably be represented or mentioned directly in a use case?

  1. Special deals may not run longer than 6 months.
  2. Customers only become preferred after 1 year.
  3. A customer has one and only one sales contact.
  4. Database response time is less than 2 seconds.
  5. Web site uptime requirement is 99.8%.
  6. Number of simultaneous users will be 200 max.

- Answer: None!
  – Most of these are non-functional requirements, so the use cases wouldn't mention them.
  – The user doesn't see them directly.

Terminology

- **actor**: an entity that acts on the system
  - **primary actor**: initiates interaction to achieve *goal*
  - **supporting actor**: performs sub-goals in use case
- **goal**: action that actor wants to accomplish
  - user (E.g register device) – MOST IMPORTANT
  - summary (E.g configure users in system),
  - subfunction (E.g logging in, locate device in DB)
- **stakeholder**: anyone interested in the system
  - supplier, stock agency, vendor
  - stakeholder might not "act" in any scenario

Use cases are always initiated by actors and describe the **flow of events** that these actors are involved in.

Terminology

- Actor can be a human being or another system
  - **External system** – E.g billing system needs to get the courses a student registered to from registration system
  - **Human** – a student, teacher, registrar

- An actor is not a person but a **role** played by a person or an external system.
  - E.g the same person can be a student or an instructor.

Styles of use cases

1. **Use case diagram**
   - shows all use cases in system
   - In UML (Unified Modelling Language – a famous design language)

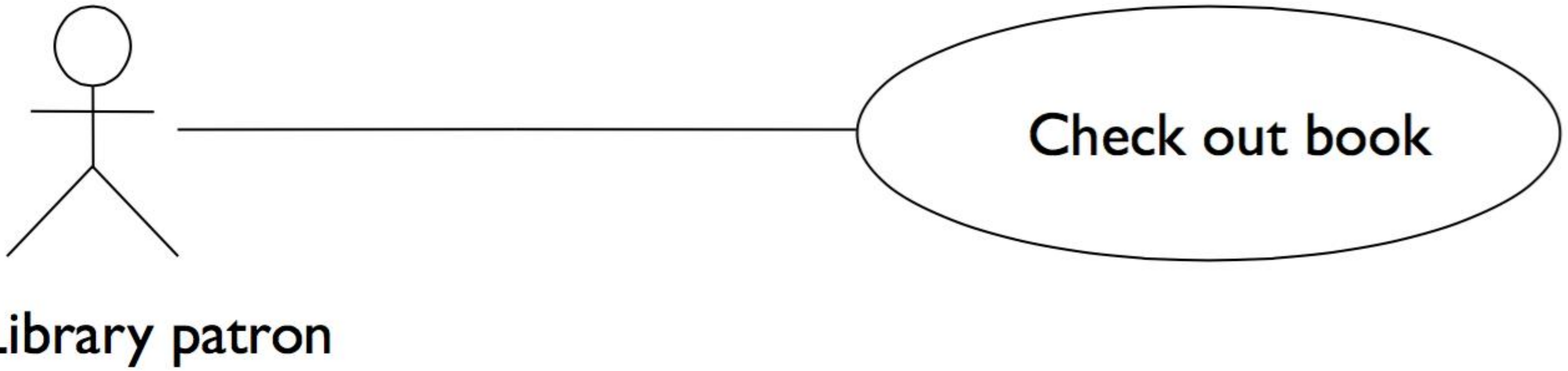
2. **Informal use case**
   - a short paragraph

3. **Formal use case**
   - a multi-part structured description

Let's examine each of these in detail...

1. Use case diagram

- The overall list of the 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 (verb phrases)
  - **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
    - E.g Check out book depends on login



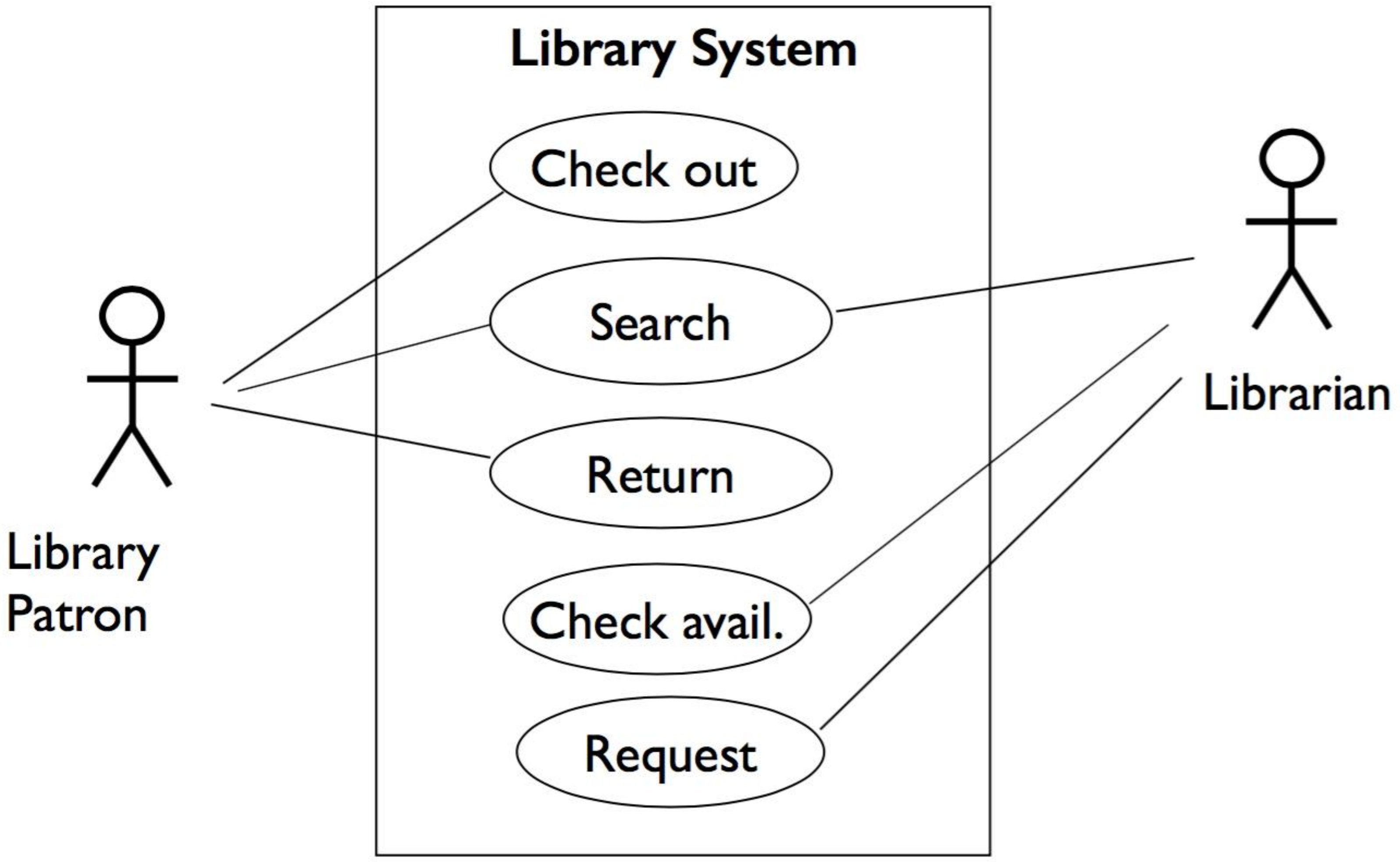
Actor-goal lists: function content of the system

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

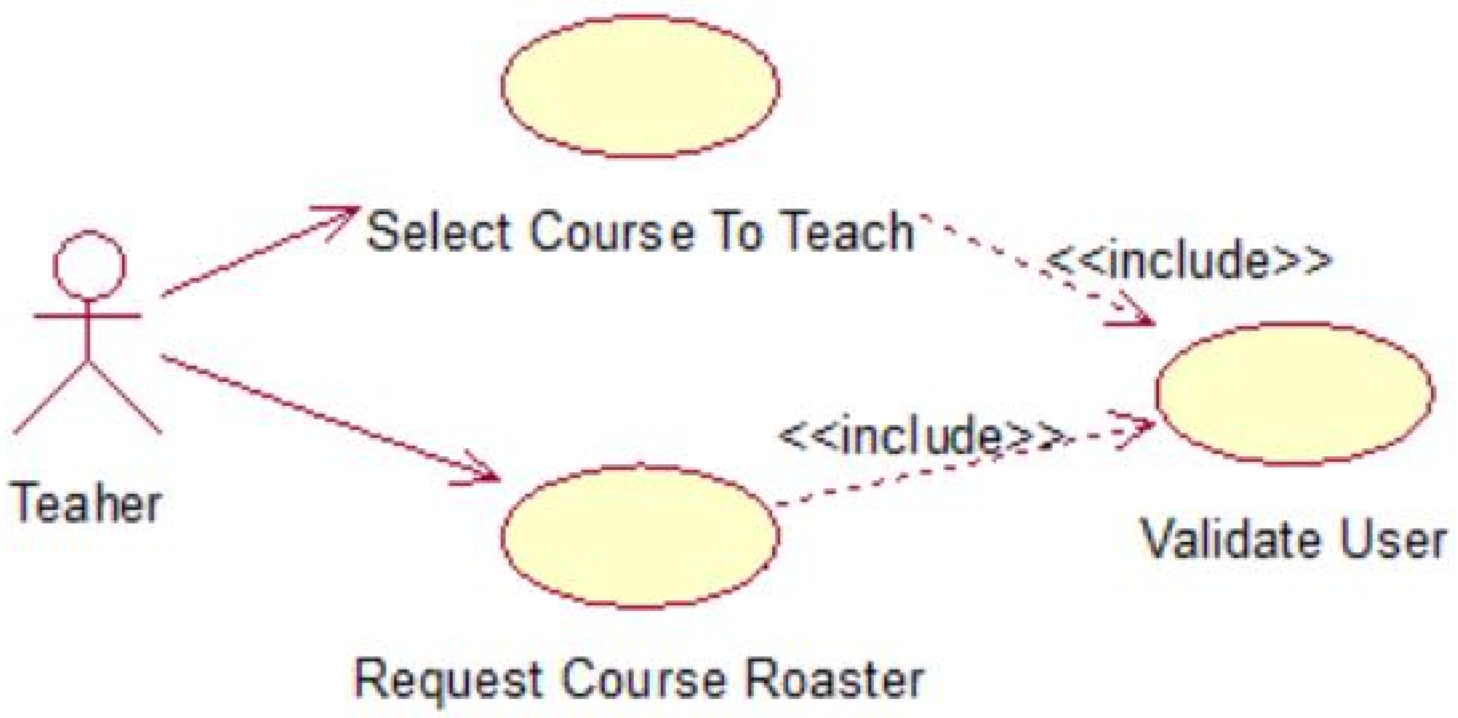
It can be useful to create a list or table of primary actors and their "goals" (use cases they start).

The diagram will then capture this material

Use case summary diagrams



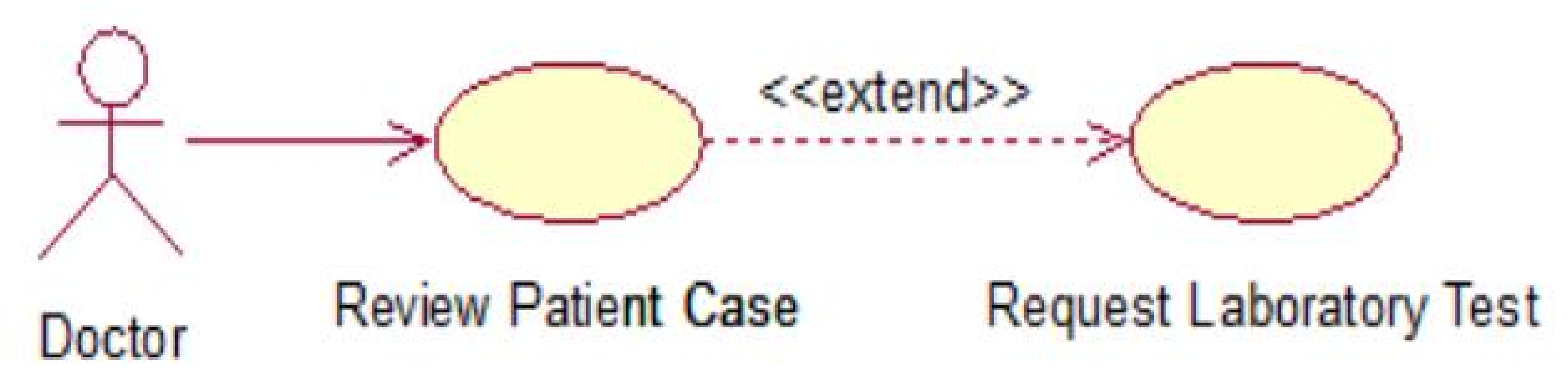
<<includes>> relationship



Validate Use case may be used in other contexts

- **<<includes>> (aka<<uses>>) relationship** used for events that are in the flow of events of the source use case.
- << >> are called stereotypes in UML

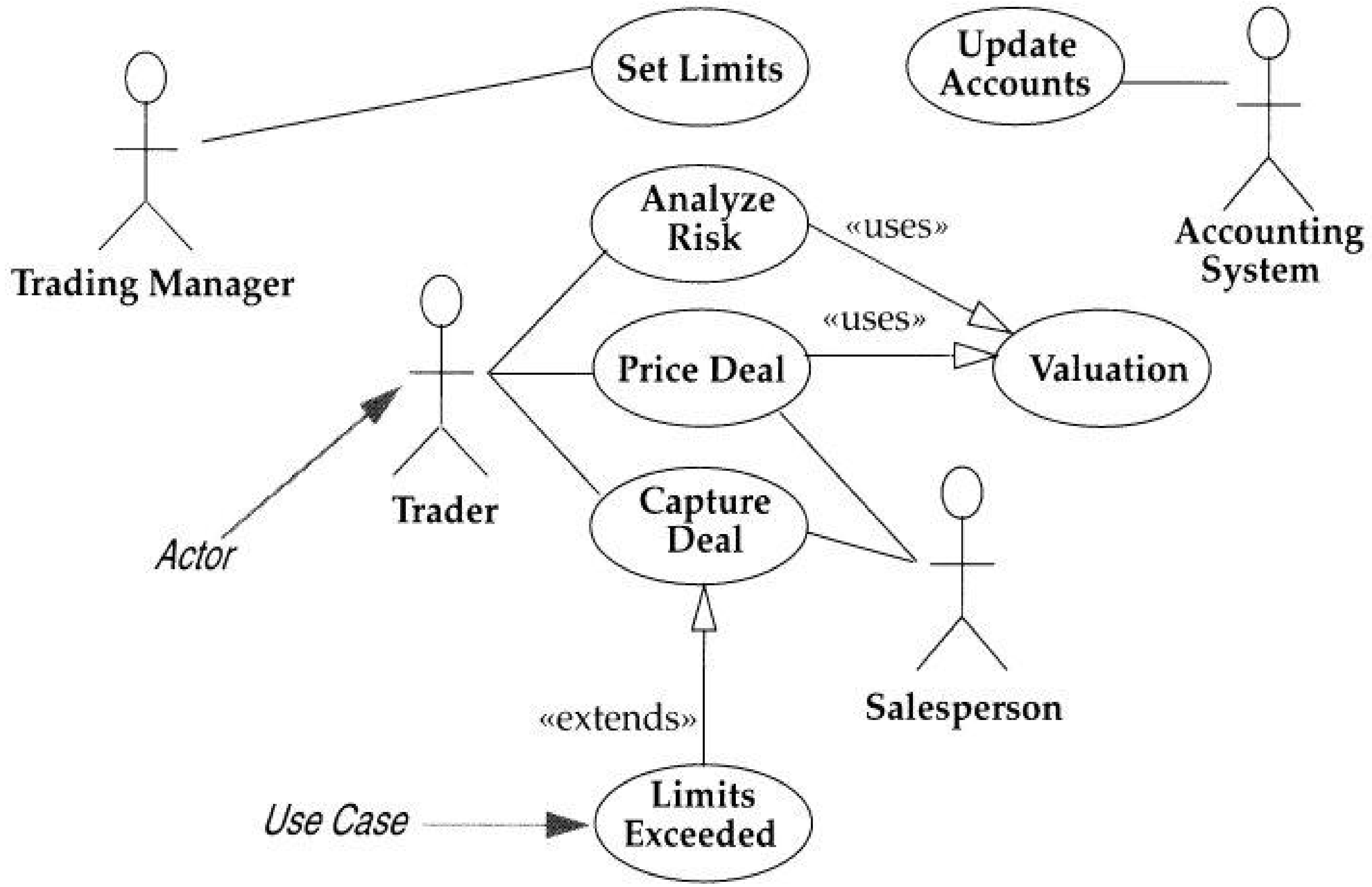
<<extends>> Relationship



<<extend>>used for exceptional conditions, especially those that can occur at any time.

- E.g Request Laboratory Test case may be performed after Reviewing Patient’s Case

Use case summary diagram



2. Informal use case

- **informal use case**: a paragraph describing the scenario
- Example (for a library management system):
  - 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 book

Informal use case with structured text

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

You will probably use something in this general style.

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

What is an extension?

- A possible branch in a use case scenario, often triggered by an error or failure in the process.
  - Useful for finding edge cases that need to be handled and tested.
- Do
  - Think about how every step of the use case could fail.
  - Give a plausible response to each extension from the system.
  - Response should either jump to another step of the case, or end it.
- Don’t
  - List things outside the use case ("User's power goes out").
  - Make unreasonable assumptions ("DB will never fail").
  - List a remedy that your system can't actually implement.

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	Book is reserved
Failure end condition	Book is not reserved
Trigger	Patron logs into system

Parts that make up a formal use case (continued on the next slide).

Formal Use case

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 6. Patron selects location and reserves book 7. System confirms reservation and continues from step 3
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

3. Formal use case

(goal of primary actor)

(level of goal [summary, user, subfunction])

(primary actor)

"Place an order" (User goal / Clerk)

(action steps:
full sentences showing
who takes the action!
3 - 9 steps long.)

Main scenario:

1. Clerk specifies customer, item and quantity.
2. System accepts and queues the order.

...

(condition causing different actions)

Extensions:

1a. Low credit & Customer is 'Preferred':

1a1. System gives them credit anyway.

(action step(s)
handling those conditions)

1b. Low credit & not 'Preferred' customer:

1b1. Clerk performs Sign Up Preferred Customer scenario and accepts only prepayment.

(calling another use case)

2a. Low on stock: Customer accepts rain-check:

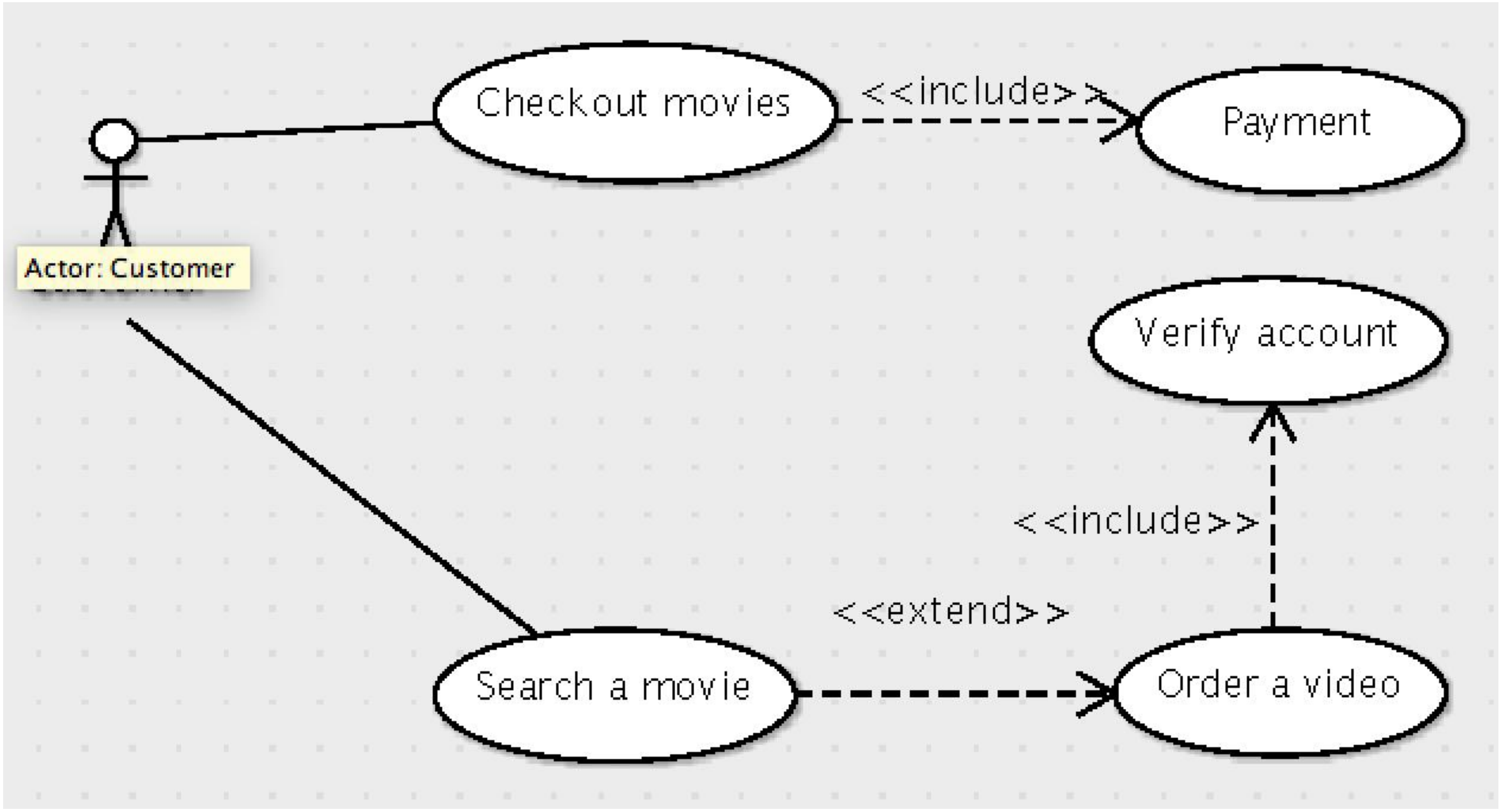
2a1. Clerk reduces order to available stock level.

Use case exercise

- Consider a Netflix-like video rental web system.
  - A customer with an account can use their membership and credit card in the web app to order a video for rental.
  - The software can look up movies and actors by keywords.
  - A customer can check out up to 3 movies, for 5 days each.
  - Late fees can be paid at the time of return or at next checkout.

- Exercise:
  - Come up with **3-4 use case names** for this software.
  - Identify some of the **actors and stakeholders** in this system.

Solution


Checkout movies
<<include>>
Payment
Actor: Customer
Verify account
<<include>>
<<extend>>
Search a movie
Order a video

What notation is good?

- There are standard templates for requirements documents, diagrams, etc. with specific rules.
- Is this a good thing? Should we use these standards or make up our own?
  - Standards are helpful as a template or starting point.
  - But don't be a slave to formal rules or use a model/scheme that doesn't fit your project's needs.

Cockburn's 4 use case steps

1. Identify actors and goals

- What computers, subsystems, people will drive our system? (actors)
- What does each actor need our system to do? (goals)

2. Write the main success scenario

- easiest to read; everything else is a complication on this
- capture each actor's intent and responsibility
  - say what information passes between them
  - number each line

Alistair Cockburn

Cockburn's 4 use case steps

3. List the failure extensions

- usually almost every step can fail (bad credit, out of stock...)
  - note failure condition separately, after main success scenario
- Describe failure-handling
  - recoverable: back to main course (low stock + reduce quantity)
  - non-recoverable: fails (out of stock, or not a valued customer)
  - each scenario goes from trigger to completion
- Label with step number and letter:
  - 5a failure condition
    - 5a.1 use case continued with failure scenario
    - 5a.2 continued

Cockburn's 4 use case steps

- **4. List the variations**
  - Many steps can have alternative behaviors or scenarios
  - Label with step number and alternative
    - 5’. Alternative 1 for step 5
    - 5”. Alternative 2 for step 5

Usability

- **Usability**: The effectiveness with which users can accomplish tasks in a (software) system, as measured by:
  - **Learnability**: is it easy to learn?
  - **Efficiency**: once learned, is it fast to use?
  - **Safety**: are errors few and recoverable?

- Importance criteria – Learnability, Efficiency,
  - Depends on the user
    - Novices need learnability.
    - Experts need efficiency.
    - But no user is uniformly a novice or an expert.
  - Depends on the task
    - Missile launchers need safety.
    - Subway turnstiles need efficiency.

Usability Matters: the cost of getting it wrong

50% of all "malfunctioning" electronic devices returned to stores are in full working order, but users can't figure out how to operate them.

- Elke den Ouden, 2006

- You can read further for detail information.

Three Mile Island: nuclear reactor meltdown caused by an ambiguous user interface

A good user interface is hard to design ...

- You are not the user
  - Most software engineering is about communicating with other programmers.
  - UI is about communicating with users.
- Users are always right ...
  - Consistent problems are the system’s fault.
- Except when they aren’t
  - Users don’t always know what they want.

Achieving usability: Best practices

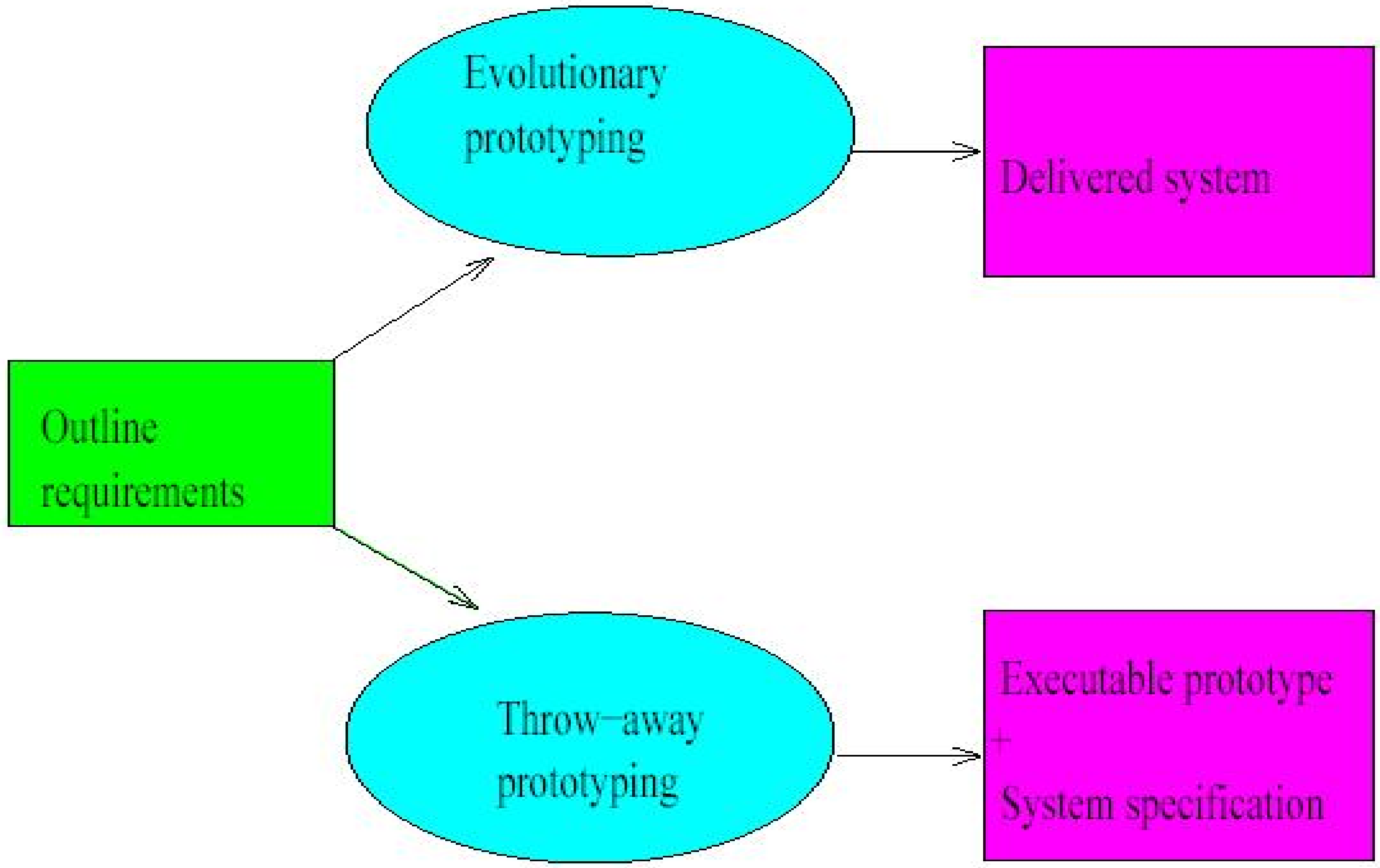
- some methods to achieve good usability:
  - user testing / field studies
    - having users use the product and gathering data
  - evaluations and reviews by UI experts
  - prototyping
    - paper prototyping
    - code prototyping
- Good UI design focuses on the *user,* not developer or system.

User Interface: Prototyping

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

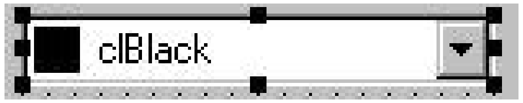
- What are some possible benefits of prototyping?
  - aids UI design
  - help discover additional requirements
  - help discover test cases and provide a basis for testing
  - allows interaction with user and customer to ensure satisfaction
  - team-building

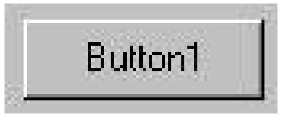
Types of Prototyping



Some prototyping methods

- UI builders





- draw a GUI visually by dragging/dropping UI controls on screen
- Code prototyping
  - Writing a "rough" version of your code

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

Why paper prototyping?

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

ABC 3G
04:22 PM
Setup
Add
Al's Information
Age:
85
Gender:
Male
Address:
1234 Main Street
Anytown, USA 1001

When to do (paper) prototyping?

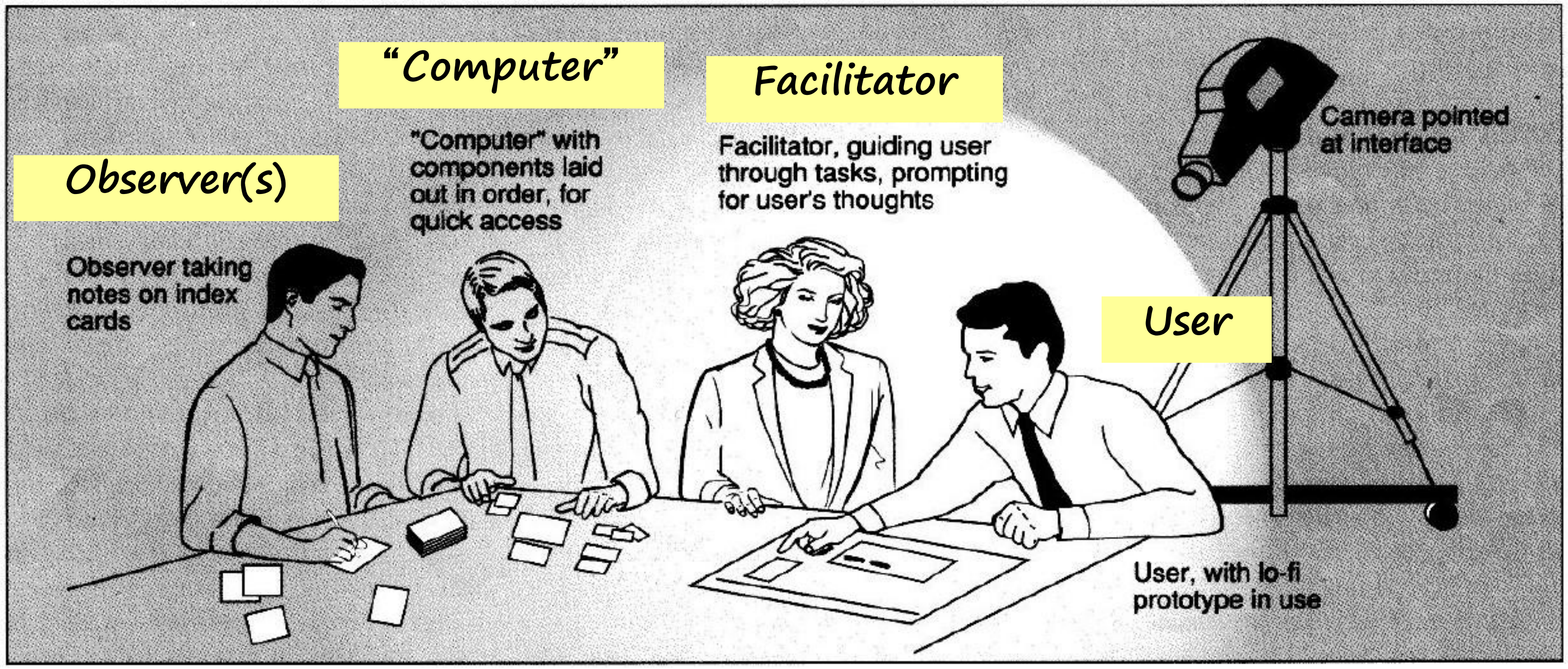
- *Question*. Requirements are the **what** and design is the **how**. Which is paper prototyping?

- *Answer*. 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
  - Included as a requirements artifact to initially envision the system

ABC 3G
04:22 PM
Setup
Add
Al's Information
Age:
85
Gender:
Male
Address:
1234 Main Street
Anytown, USA 1001

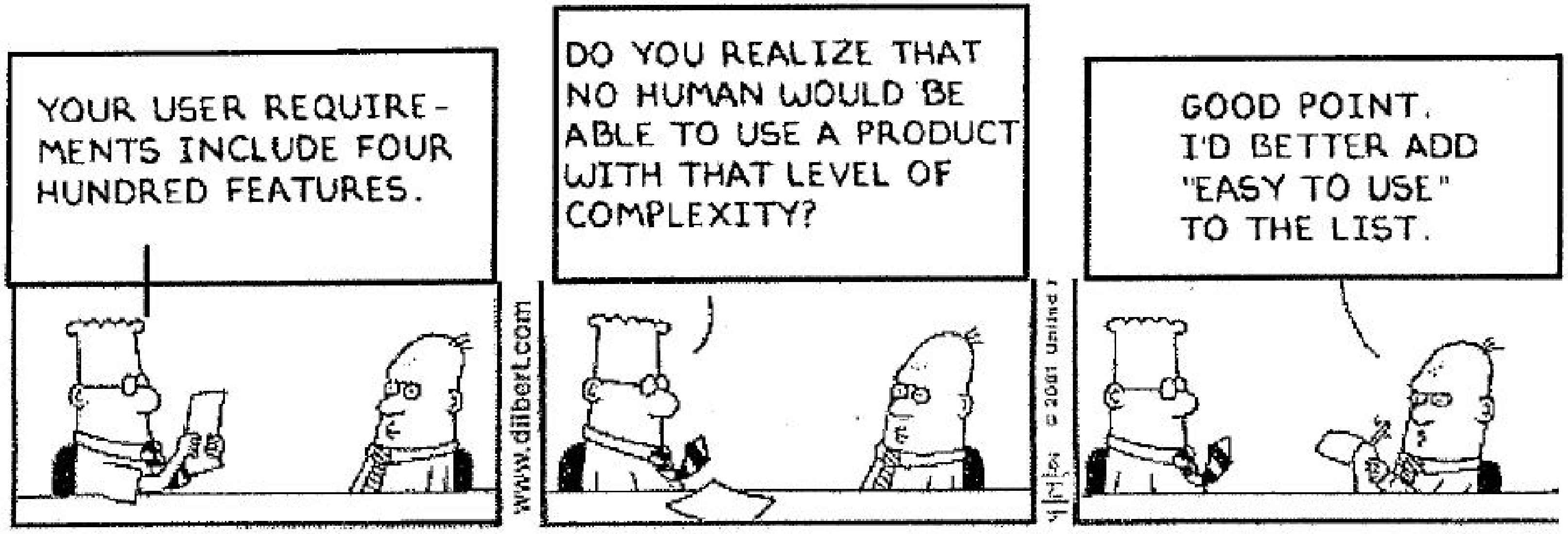
Paper prototype usability testing

- user is given tasks to perform using paper prototype
- Facilitator guides the user through tasks, prompting for feedback.
- session can be observed by people or camera
- one developer can "play computer"


"Computer"
Facilitator
Observer(s)
"Computer" with components laid out in order, for quick access
Facilitator, guiding user through tasks, prompting for user's thoughts
Camera pointed at interface
Observer taking notes on index cards
User
User, with lo-fi prototype in use

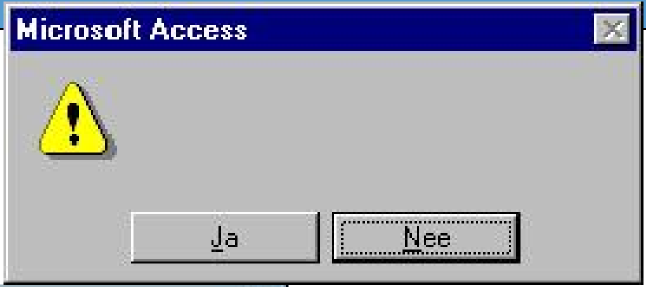
Schneiderman's 8 Golden Rules

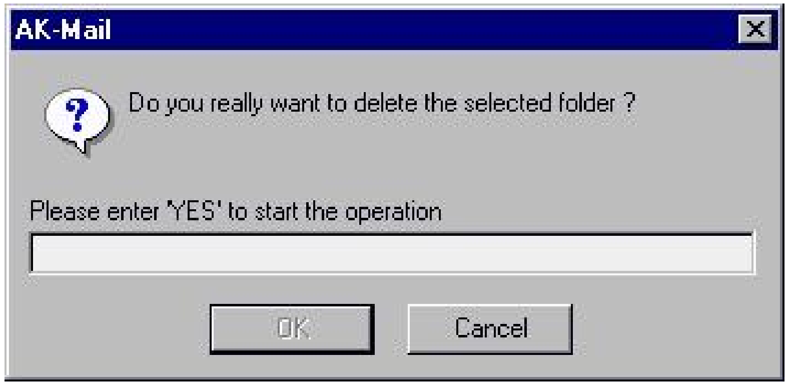
- Strive for consistency.
- Give shortcuts to expert user.
- Offer informative feedback.
- Make each interaction with the user yield a result.
- Offer simple error handling.
- Permit easy undo of actions.
- Let the user be in control.
- Reduce short-term memory load on the user.


YOUR USER REQUIREMENTS INCLUDE FOUR HUNDRED FEATURES.
DO YOU REALIZE THAT NO HUMAN WOULD BE ABLE TO USE A PRODUCT WITH THAT LEVEL OF COMPLEXITY?
GOOD POINT. I'D BETTER ADD "EASY TO USE" TO THE LIST.
www.dilbert.com
© 2001 United

http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html

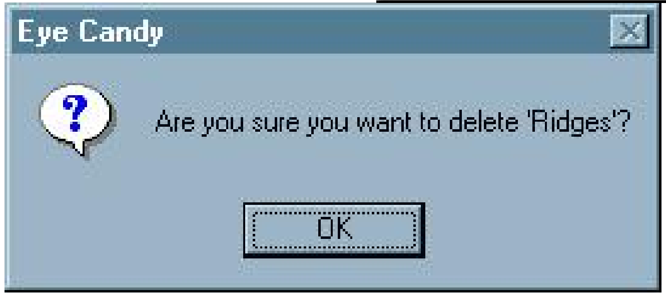
Bad error messages


Microsoft Access

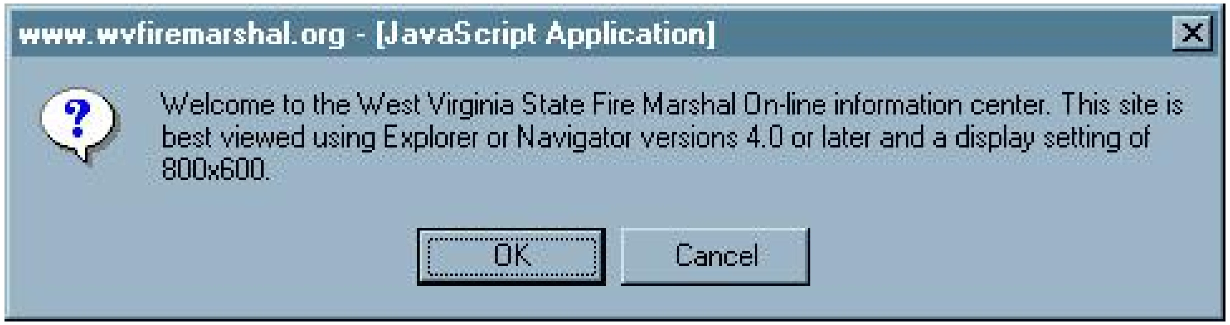

AK-Mail

Do you really want to delete the selected folder ?

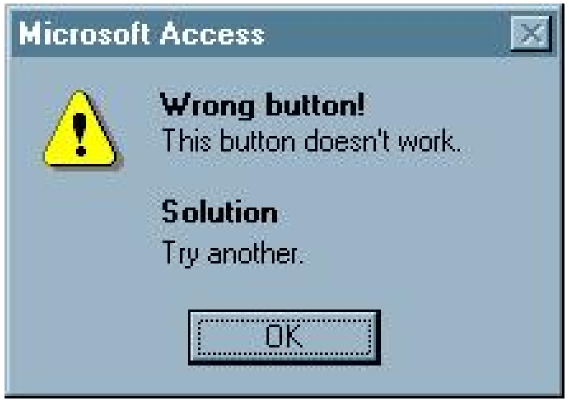
Please enter 'YES' to start the operation


Eye Candy

Are you sure you want to delete 'Ridges'?

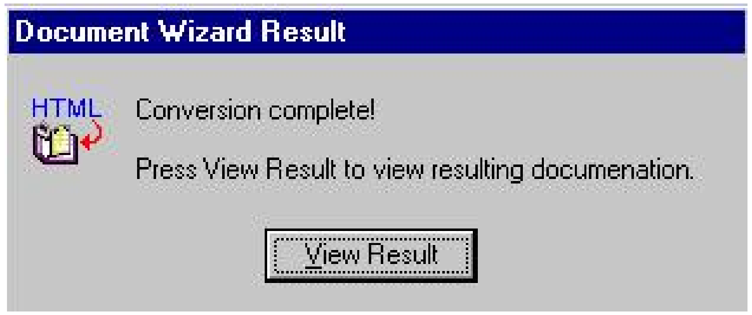

www.wvfiremarshal.org - [JavaScript Application]

Welcome to the West Virginia State Fire Marshal On-line information center. This site is best viewed using Explorer or Navigator versions 4.0 or later and a display setting of 800x600.


Microsoft Access

Wrong button!
This button doesn't work.

Solution
Try another.


Document Wizard Result

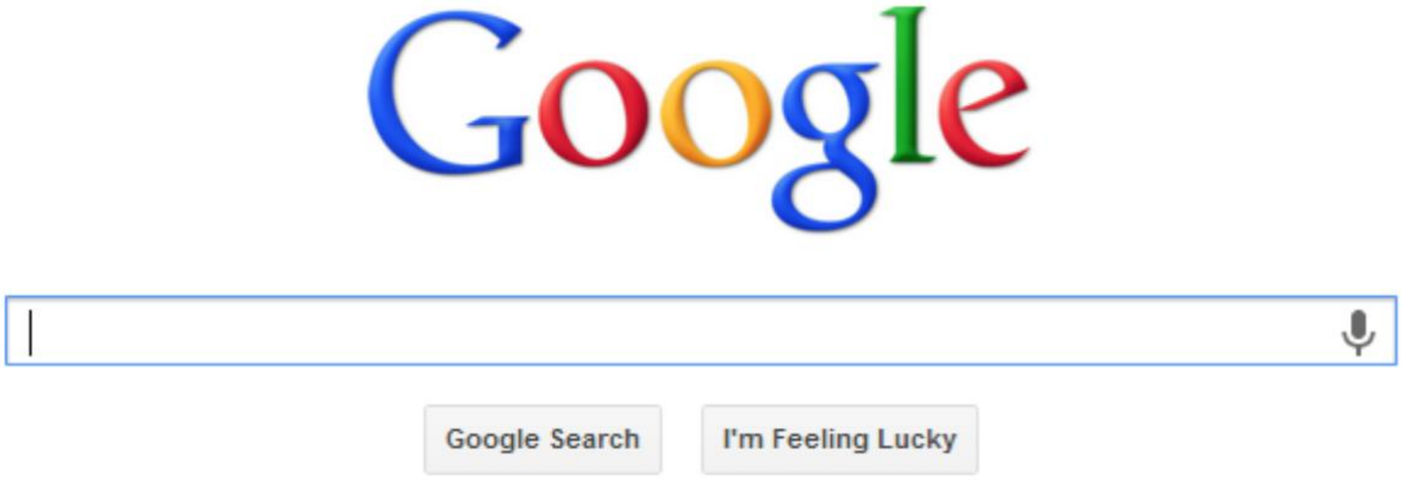
Conversion complete!

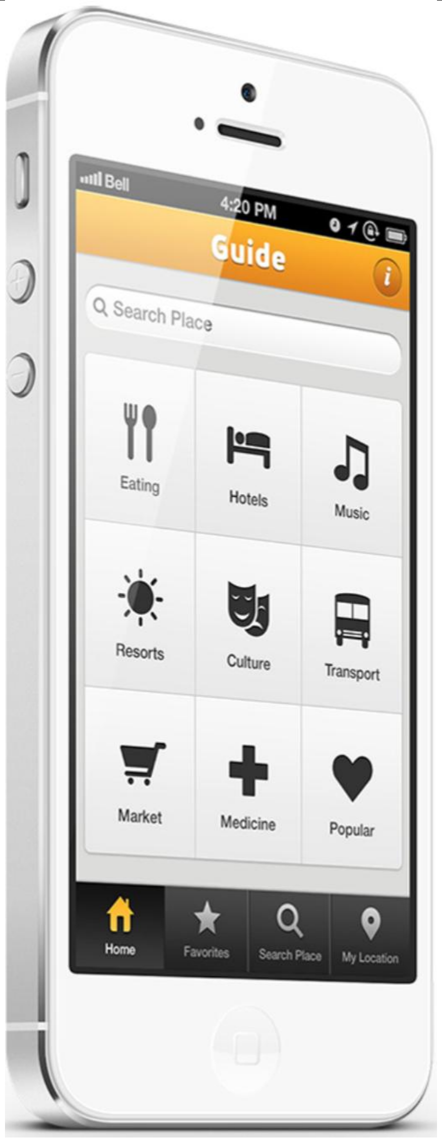
Press View Result to view resulting documenation.

UI Hall of Fame

Reading Assignment :

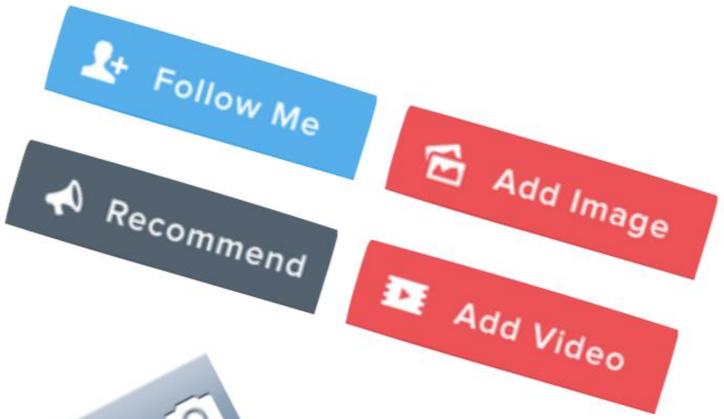
- Read about UI hall of fame/shame

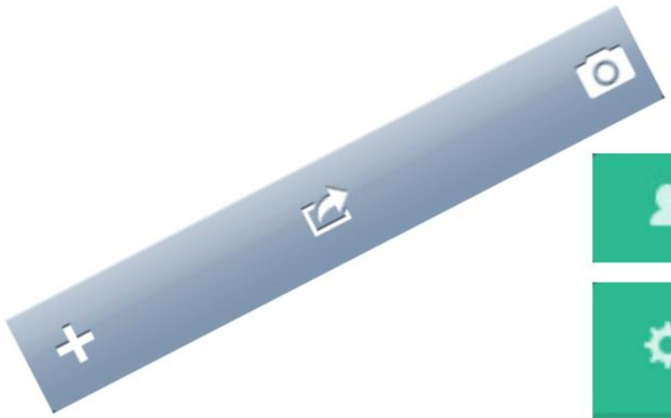


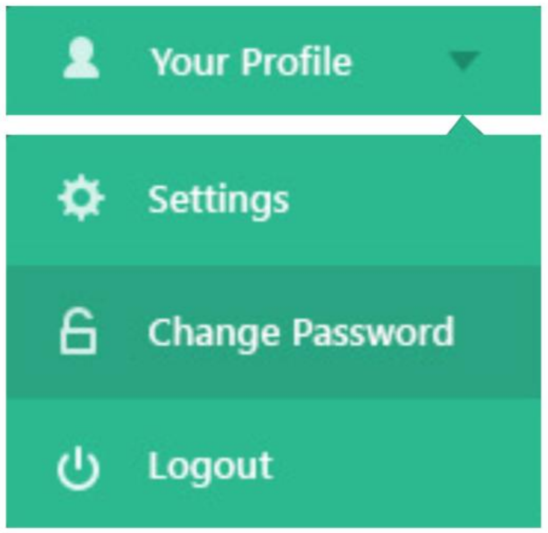


UI design: components

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

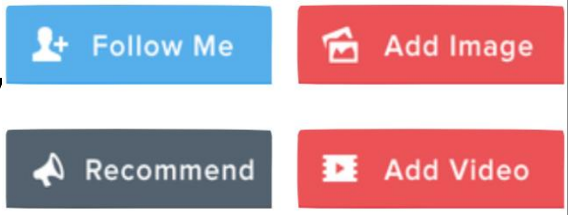






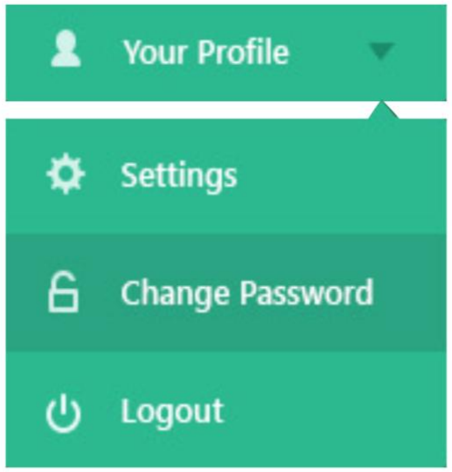
UI design - buttons, toolbars, menus

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



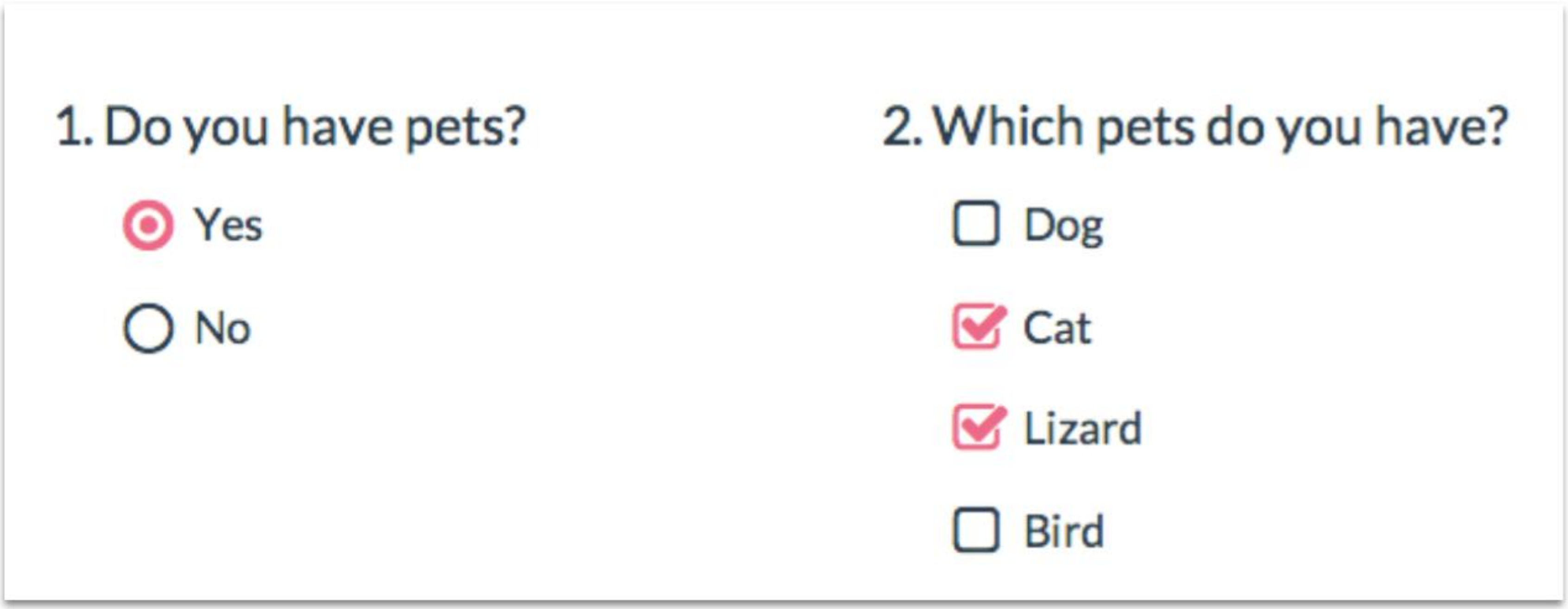
- Use **toolbars** for common actions.

- Use **menus** for infrequent actions applicable to many screens.
  - *Users hate 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
- Use **radio buttons** for a small number of related choices, when only one can be activated at a time


1. Do you have pets?

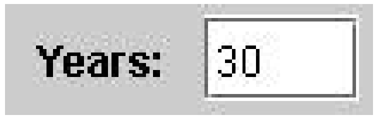
[✓ Yes] No

2. Which pets do you have?

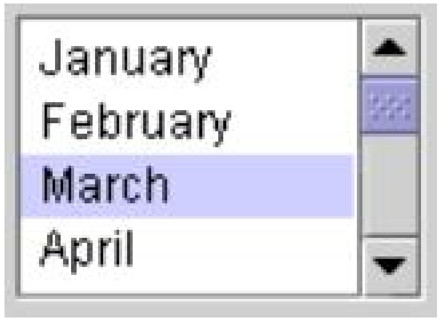
Dog [✓ Cat] [✓ Lizard] Bird

Lists, combo boxes, etc.

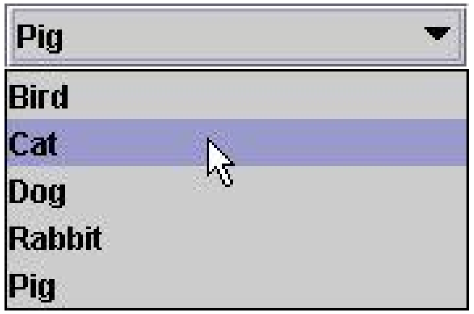
- use **text fields** (usually with a label) when the user may type in anything they want
  - you will usually have to **validate** the input



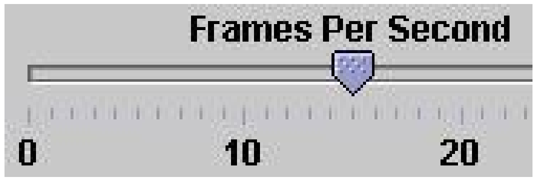
- use **lists** when there are many fixed choices (too many for radio buttons to be practical) and you want *all* choices visible at once

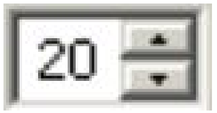


- use **combo boxes** when there are many fixed choices, but you don't want to take up screen space by showing them all at once



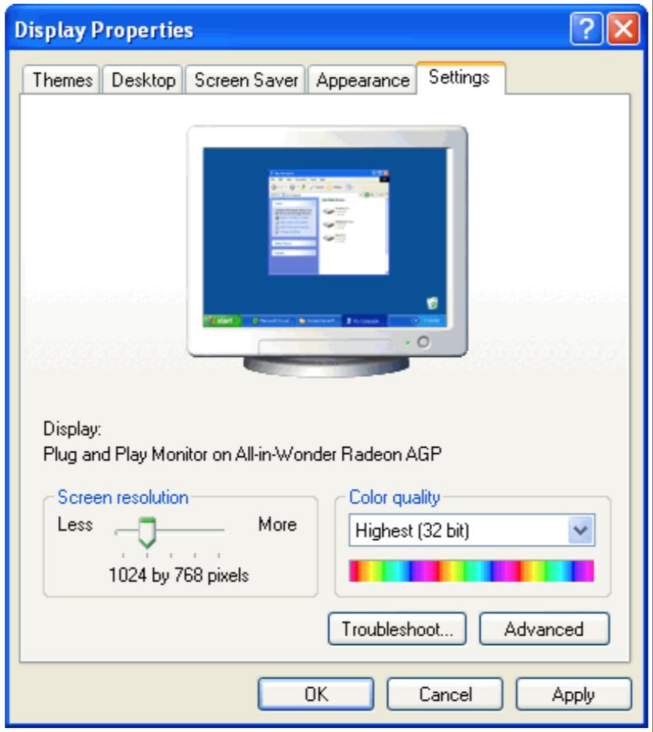
- use a **slider** or **spinner** for a numeric value with fixed range





UI design - multiple screens

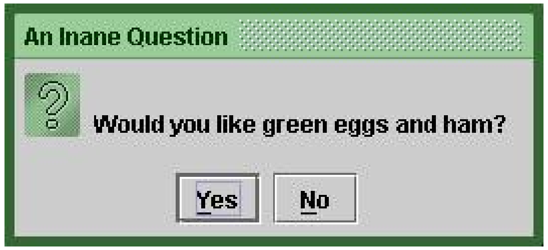
- you can use a **tabbed pane** when there are many screens that the user may want to switch between at any moment
  - or multiple pages, if it's a web site
- use **dialog boxes** or **option panes** to present temporary screens or options
  - users *hate* popup dialogs; use them very rarely
  - don't prompt for lots of user input by popping up dialogs
    - instead, put the choices on the existing window as buttons, etc.



Display:
Plug and Play Monitor on All-in-Wonder Radeon AGP

Screen resolution
Less More
1024 by 768 pixels

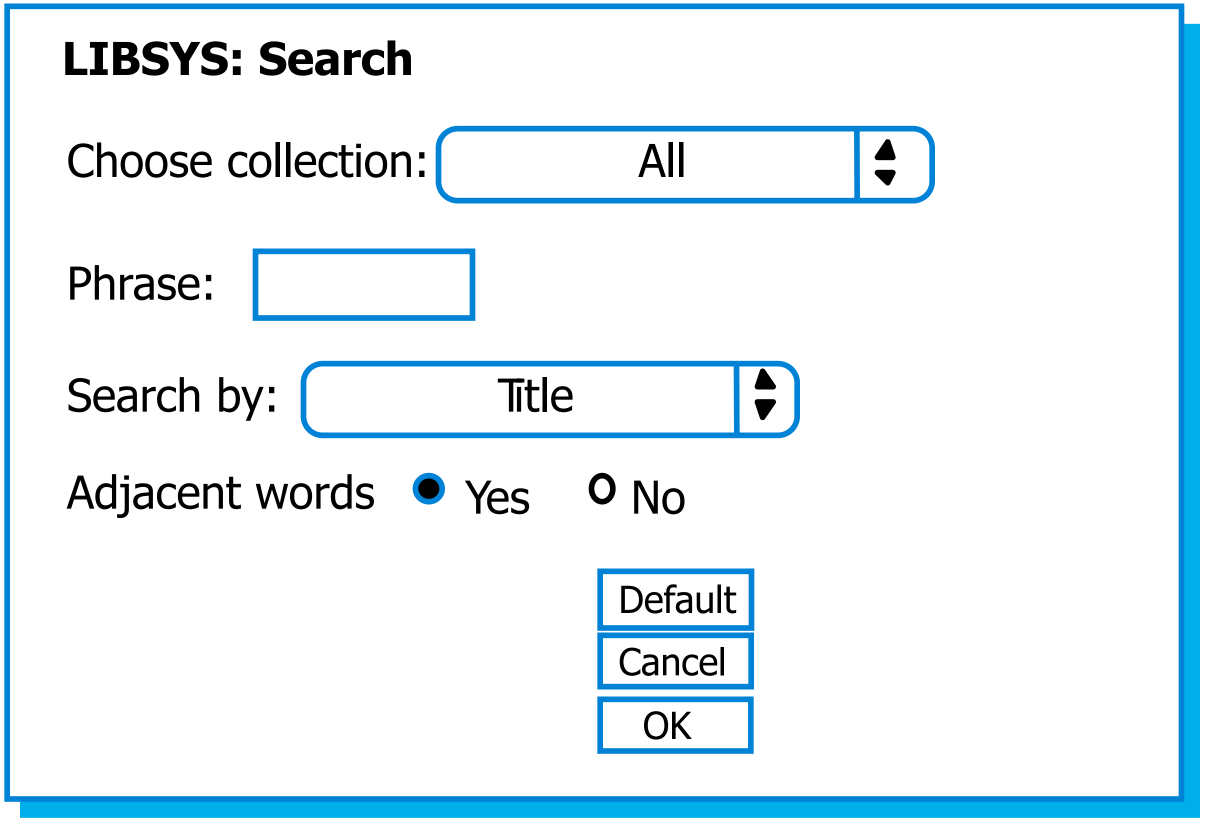
Color quality
Highest (32 bit)



Would you like green eggs and ham?

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)



LIBSYS: Search

Choose collection: All

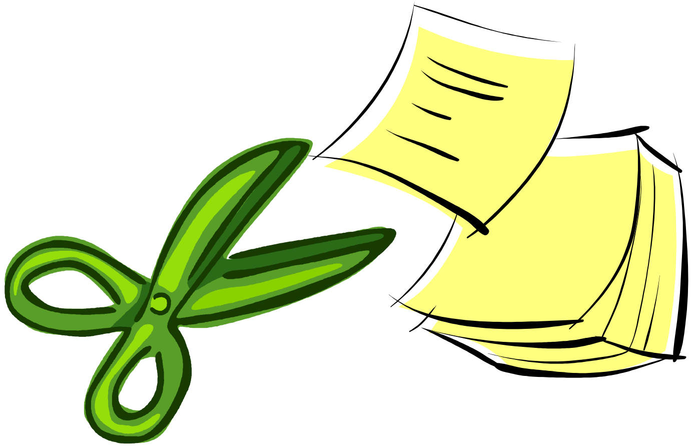
Phrase:

Search by: Title

Adjacent words [✓ Yes] No

Creating a paper prototype

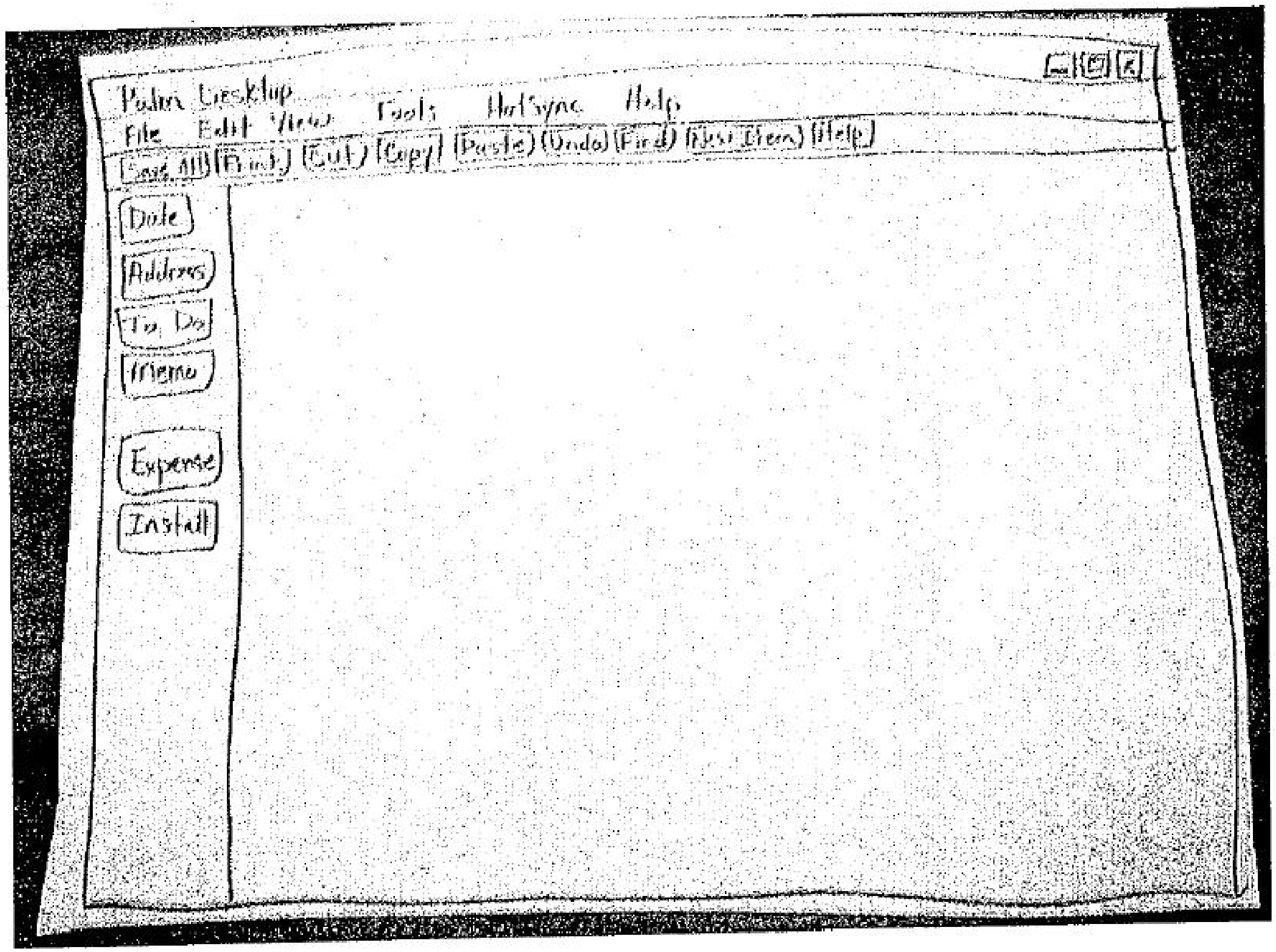
- gather materials
  - paper, pencils/pens
  - tape, scissors
  - highlighters, transparencies



- 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.

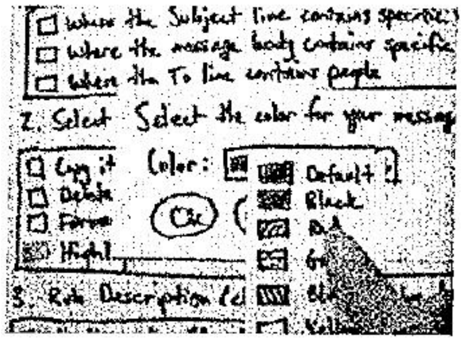
Application backgrounds

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



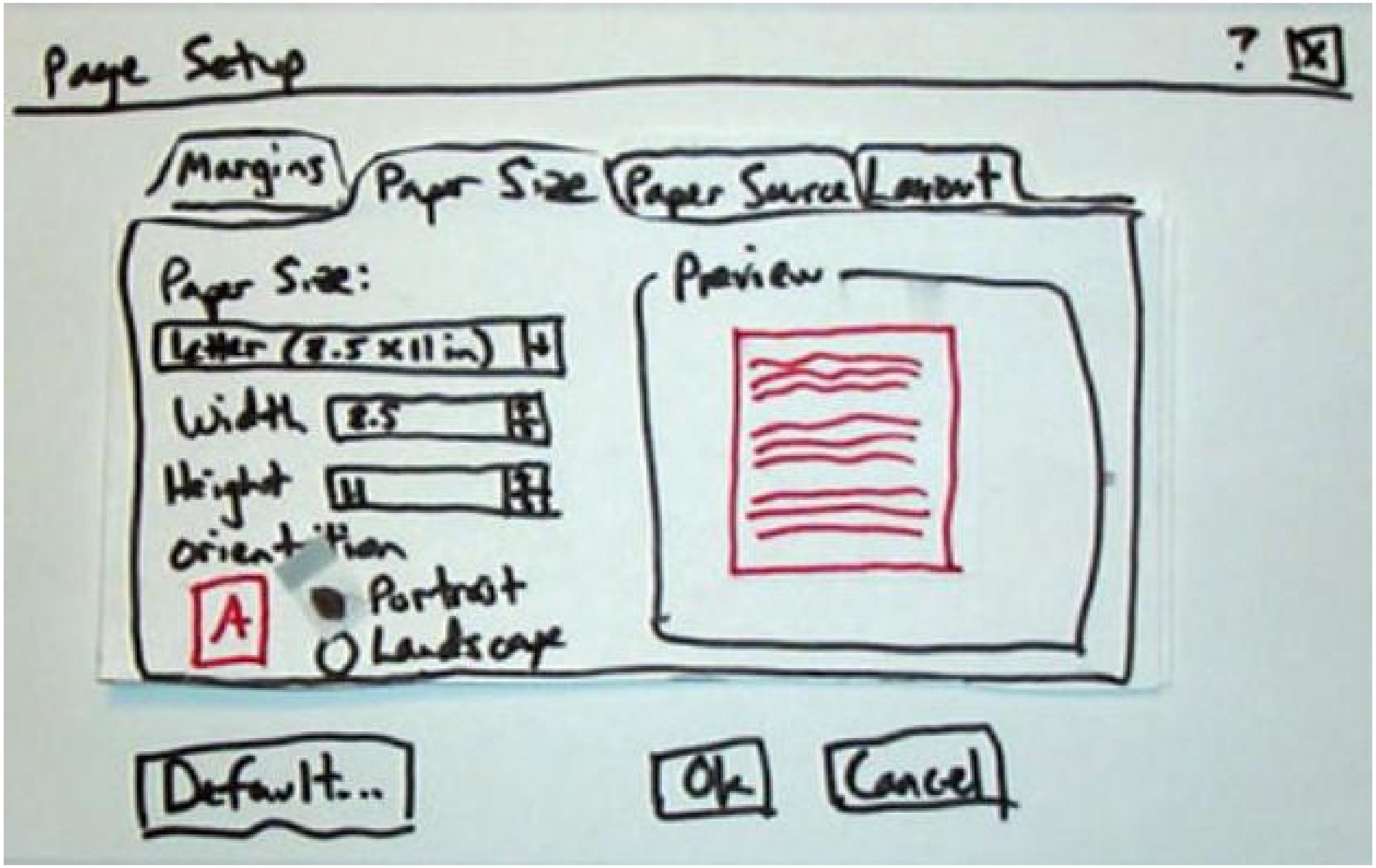
Interactive widgets

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



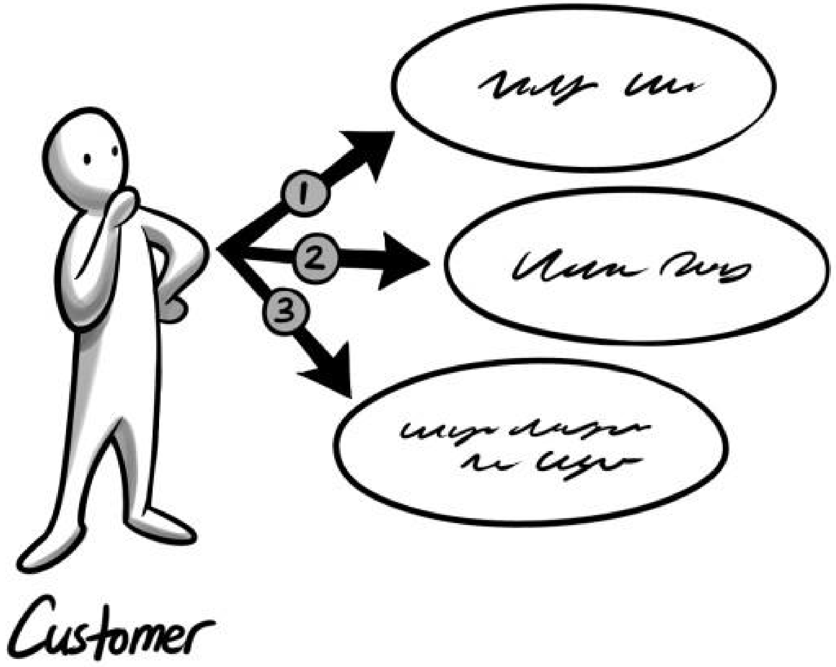
where the Subject line contains specific
where the message body contains specific
where the To line contains people
2. Select Select the color for your message
Copy it
Delete
Forward
Highlight
Color:
Default
Black
OK
3. Rule Description

Example paper prot. screen



Summary

- Uses case describe example system behaviors (contracts) from the user's point of view.
- Can be diagrams, informal paragraphs, formal use cases.
- 4 steps to create use cases



- Tasks

1, Finishing up functional and non functional requirement

2, develop use case for your specific system on paper

3, sketch your user interface.