

**6.813/6.831 • USER INTERFACE DESIGN AND IMPLEMENTATION**

Spring 2012 Massachusetts Institute of Technology  
Department of Electrical Engineering and Computer Science

**PS1: IMPLEMENTING AN HTML/JAVASCRIPT APPLICATION**

Due at 11:59pm, Sunday, March 4, 2012, by uploading your submission to Stellar.

This problem set is for the undergraduate 6.813 class only.

This problem set asks you to build a little game consisting of a small user interface which asks the user to translate words from a language to another. You'll be using the set of core technologies that drive most modern web applications, namely HTML, CSS, and Javascript. We specify the design of the user interface. Your job is to implement it using HTML, CSS, and Javascript.

To do this assignment, you'll need to know how to:

- write HTML: create correct HTML pages that display in modern browsers
- use HTML form elements (like `<button>`, `<input>`) to create a web application
- use CSS to layout and style HTML elements
- use Javascript and jQuery to add event handlers and bindings to respond to user input;
- use jQuery UI to augment the standard HTML `<input>` with an autocomplete feature.

✓ I'm good

Here are some useful reference sources for HTML/CSS/Javascript/jQuery/jQueryUI:

- HTML Dog (a fairly good guide for both HTML and CSS)
- CSS Tutorial (a tutorial for learning CSS)
- How jQuery Works (a beginning tutorial on jQuery, the Javascript library we are using for this assignment)
- jQuery Documentation (the official documentation for jQuery is an excellent resource)
- jQuery UI's Autocomplete Documentation (jQuery UI lets you create widgets using a higher lever of abstraction than jQuery alone. For part 4, you'll need to use the jQuery UI autocomplete widget)
- O'Reilly Safari has several e-books on HTML, CSS, and Javascript that are free for MIT students to read (MIT certificate required). Dynamic HTML: The Definitive Reference, 3rd Edition is a good one to start with.

**Provided Resources**

We provide you with the following:

- ps1.zip: a zip file containing the code you will be editing for this problem set.

You can import this zip file directly into Eclipse using File/Import/Existing Projects into Workspace/From Archive file, or use whatever text editor you would like. In the contents folder of the project are the following files and folders:

- jquery: the code for jQuery and jQuery UI, the Javascript libraries you will be using for this assignment.
- dicts.json: a set of bilingual dictionaries, containing words without accents or capital letters, extracted from English Wiktionary dumps, in JSON notation.
- translate\_game.html: a skeleton file for your user interface HTML code.
- translate\_game.css: a skeleton file for your user interface CSS code.
- translate\_game.js: a skeleton file for your user interface JS code.

Feel free to change these files as you see fit.

If you are using Eclipse to edit these files, we highly recommend installing the appropriate plug-ins for web development support. On the latest release of Eclipse (Indigo), you can do this by going to Help/Install New Software..., then selecting the Indigo update site, then selecting Web, XML, and Java EE Development/Eclipse Web Developer Tools and Javascript Development Tools.

**1- Basic Layout (20%)**

First, build an interface that looks like figure 1. In particular:

- There should be margins of at least 10px between and around the elements
- The instruction message on top should be quite large and aligned with the elements below it
- A three-column layout should be used in order to align the row containing the labels ("Spanish", "English", "Answer") with the row containing the current entry ("espejo", `<input>` element and See Answer button), and with the rows containing the past



- entries (bottom).
- Create two static past entries. One for a correct translation ("calabaza", "pumpkin") using a blue font color, and a check mark (Hint: jQuery UI comes with plenty of symbols, and you can just inspect them to see which CSS class to use). The other for an incorrect translation ("postmodernidad", "phone") using a red font color, crossing out the incorrect entry (Hint: think CSS...), and displaying the correct translation on the third column.

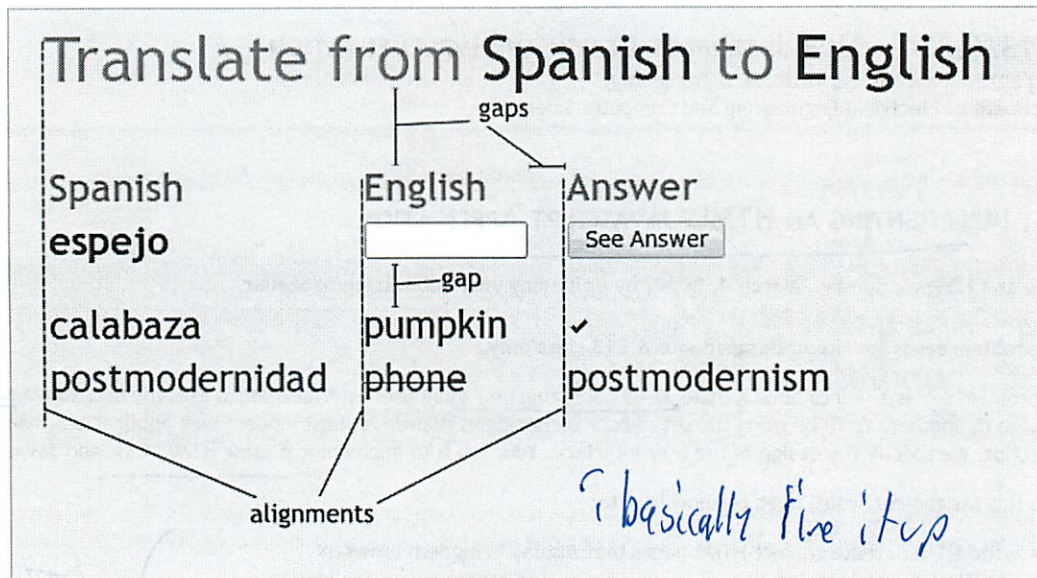


Figure 1

No behavior is required to earn the points for this part, just a static layout.

## 2- Adding Behavior (20%)

Improve your interface so that when the program is first run:

- The names of the languages you need to translate from (Spanish) and to (English) should now come from variables on the JS code (namely, `lang_from` and `lang_to`). Note that jQuery makes those kind of tasks particularly easy.
- The program chooses a random word from the dictionary for the user to translate (instead of `espejo`, on figure 1)
- The input element has focus, so that the user can start typing as soon as the page is loaded.
- Clicking on the **See Answer** button prompts for another word, and clears the `<input>`. Don't check if the answer was right or wrong for now: just move on to the next word.
- Note: For now, just keep your 2 static answers from part 1 in the list of previous answers

## 3- Displaying Answers (20%)

Update your code so that instead of the two static answers you entered in part 1, the list of previous answers list gets populated with the user entries. The entry which was submitted last should appear at the top of the list. Use the styles you defined in Part 1 for correct and incorrect entries. When deciding whether an entry is correct, only perform exact matches: synonyms, capitalization differences etc... should be reported as incorrect translations.

## 4- Adding Autocomplete (20%)

The input field contains the user's guess. Use jQuery UI to make this field an autocomplete widget (see Figure 2), so that:

- It offers suggestions as soon as the user types two or more characters. Those selections should be the English words from the dictionary which contain the letters that the user has typed so far.
- Selecting a suggestion has the same behavior as clicking on the **See Answer** button

*this is pretty similar to what I have done before*



## Translate from Spanish to English

Spanish	English	Answer
escarabajo	bee	<input type="button" value="See Answer"/>
manganeso	beetle	manganese
aire	bee	✓
naranja	beef	orange
	bumblebee	
	beer	
	beech	

Figure 2

### 5- Improving usability (20%)

Make changes to your code so that pressing **enter** submits the user's reply even if the user didn't select a match from the autocomplete menu. Make sure that:

- Pressing **enter** when selecting a item from the autocomplete menu doesn't submit that item twice (don't forget to test that behavior on several browsers).
- The input field is empty and has focus immediately after submitting an entry, so that the user can submit a new entry for the next word
- The autocomplete widget closes properly even if the user ends up not using it in order to select an item.

### Further (Optional) Improvements

If you found this assignment easy and you're inclined to go further, here are some ideas for optional improvements:

- Let the user choose the what language to translate from and to. The `dicts.json` file also contains dictionaries for English-French and German-Italian. If you'd like to add more choices, one possibility is to generate your own `dicts.json` file by getting more bilingual dictionary text files and modify the very simple parser we used in order to extract those words.
- Display the user's score. Have a timed mode so that the user can only enter words during, say, 1 minute. Display a countdown of how much time is left.
- Highlight the matching substring of each word in the list box, so that the user can see at a glance how the word matches the query.
- Use cookies to remember statistics, such as best score across multiple games. This could also be used so that words the user often gets wrong can be prompted more often in order improve memorization.

### What to Hand In

Package your completed assignment as a zip file that contains all of your HTML, JS, and CSS files (including the JSON data).

List your collaborators in the comment at the top of `translate_game.html`. Collaborators are any people you discussed this assignment with. This is an individual assignment, so be aware of the course's [collaboration policy](#).

Here's a checklist of things you should confirm before you hand in:

1. Make a fresh folder and unpack your zip file into it
2. Make sure your collaborators are named in `translate_game.html`
3. Make sure all assets (images, JS files, CSS files, etc.) used by your code are found in the fresh folder and load successfully
4. Make sure that the page renders correctly on Firefox (which we'll be using for evaluating your projects) and in another modern, standards-compliant web browsers (Chrome, Safari or Opera).

Submit your zip file on Stellar.



# Doing PS)

3/4

Translate words 1 lang to another

(This is basically what I do -)

Oh need html from scratch!

Need 3 col layout

- they seem all fixed width

---

Supposed to be width of longest word

How is that done again?  
width: auto

Hmm - never tried before!

---

Do we need font?

TAs: No - but you could

Trebuchet MS - thought it looked familiar!

---

① static part



②

Now Add behavior

---

Running into JS problems -

Grrr

length not working

- returns a word !?!?

Oh its a word in the dictionary!

---

Fixed...

Set focus *focus()* Nice!

---

JS var scope again?

① Done add behavior

---

Now Display answer

- at top

↑ grr

*prepend* not append

① *What* works easily

- since I set it up right



③

Now auto complete

So ~~an~~ eng is the keys

Make a new array (edit)

✓ Nice - functional - but unstyled

And it seems in correct...

Only prefix?

it just says contain - and pic shows all

Now click ✓

And my 2 ✓

But not just prefix

L emailed in /piezzq

---

Step 5

Ah the keyboard enter ...

Did for Baker over the summer...

---

Must clear auto complete ✓

key 13



④

Does not seem to enter twice

It does not really work in IE

L > No - not needed

✓ Think all done

PS1<sup>past</sup> due

6.213 L10  
Input

3/5

GR2 out, due Sun

- generate designs + sketch

Hall of Fame / share Ribbon in Office 07

Large ~~are~~ learnability challenge

Data-driven

Saw which commands used together

Task oriented  $\rightarrow$  review tab

kept icon the same

kept keyboard shortcuts

kinda look

---

Nano Quiz

Antialiasing - <sup>implemented</sup> alpha channel  
~~view tree~~

Z-Order - view tree

Double Buffering - prevent flickering from incomplete drawings



②

Store images in reports as a PNG

---

## Today's Input

What are the surprising things in GUI toolkits

---

Input device is a big state machine

101 key keyboard - 101 bits of state

will get (x,y) mouse state

1, 2, 3 mouse buttons

new: gestures, touch events

button press + release - 2 diff states

→ GUI toolkit translates into higher level events

- Click

- doubleclick

- key press

- mouseenter, mouseleave

(Event diagrams)

③

Discrepancy: If you click in a button, hold, leave button, release  
does the button fire

---

HTML Editing test using Squarespace

Discrepancy: Sometimes up event in click is "cater"  
by double click

Double click does not count

Drag + release does not count in jQuery

---

keyboards

keyboard focus - object in view tree that  
gets these events

HTML els can't have keyboard focus by default  
- need some special tabindex code

---

Properties of keyboard events

- mouse position, button state
- special key
- timestamp
  - since might be queued
  - look at for double click



(4)

key press = key down + key up

App often must do blinking cursor itself

key press fires before key up

- it will fire multiple times if it hold down

JQuery is running a down ~~event~~ even when  
keep button down

makes it hard to program a down

Why diff #s?

down + up refer to physical keyboard  $\rightarrow A$  65  
<sup>capital</sup>  
in ASCII

but press gives you actual letter  $\rightarrow a$  97

If caps lock on

Will get same #s for both

Every key does up + down - (ie Caps Lock)

but not a key press

5

Mouse move can be coalesced into a single event in the queue  
Can simulate app being slow

- Causes straight line b/w corners

Normally each x,y pixel sent

But w/ delay the system drops/coalesces points - get last pt  
So get ~~angles~~ straight lines w/ occasional angles unless you click in the middle  
So ~~you~~ you can still drag items

You can't turn this off in JS

Can build own event queue in JS

Set timeout

Threading for multi threading in JS

High level toolkits do internally

Low level must do it yourself

## ⑥ Design of Dispatch + propagation differs

- sends keyboard to keyboard focus
- mouse events to x, y pos
- mouse capture
  - JavaSwing does not directly support
    - like mouse focus
  - for VM apps
  - or medicine dropper
  - drag/drop / scrollbar - when leave object
  - instead put up big, transparent view object to capture everything

### Propagation

event bubbles around in the hierarchy

### Consumed

event stops propagating at some point



①

Events propagate differently, in diff browser

Netscape Navigator 4 - went downward

- combined propagation w/ selection

~~When~~ Default - went upward

W3C: did both down/capture  $\rightarrow$  up/bubbling

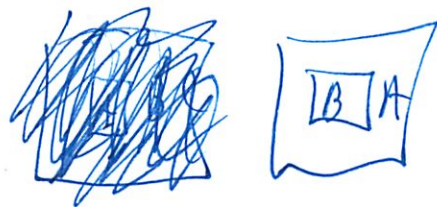
- so can override widgets

### Propagation in jQuery

↪ - only bubbling access

- no capture access

- bubbles up the tree



B mouse down

A mouse down

- could prevent A from getting click when click on B

- Only B gets mouse move events when in B

- when move out, events stop

- it can attach to whole window

⑧ will get all mouse<sup>move</sup> events when in window

~~When~~ if dragging ~~it~~ will get all ~~a~~ mouse move events  
for the entire screen

---

Also fingers (iPhone)

- More complex

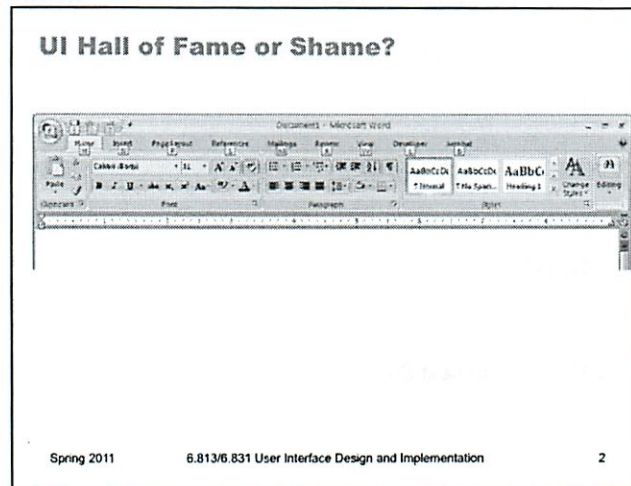
When implementing input → have a state machine

- actually draw it

## L10: Input

- GR2 out, due next Sun





Our Hall of Fame or Shame candidate for today is the command ribbon, which was introduced in Microsoft Office 2007. The ribbon is a radically different user interface for Office, merging the menubar and toolbars together into a single common widget. Clicking on one of the tabs (“Home”, “Insert”, “Page Layout”, etc) switches to a different ribbon of widgets underneath. The metaphor is a mix of menubar, toolbar, and tabbed pane. Notice how UIs have evolved to the point where new metaphorical designs are riffing on existing *GUI* objects, rather than *physical* objects. Expect to see more of that in the future.

Needless to say, strict **external consistency** has been thrown out the window – Office no longer has a menubar or toolbar. But if we were slavishly consistent, we’d never make any progress in user interface design. Despite the radical change, the ribbon *is* still externally consistent in some interesting ways, with other Windows programs and with previous versions of Office. If you look carefully at the interface, they *are* consistent in some important ways: (1) critical toolbar buttons still look the same, like Save, Cut, Copy, and Paste; (2) the command tab buttons resemble menubar menus, in both location and naming; (3) the ribbons look and act like rich toolbars, with familiar widgets and familiar affordances. So even though some new learning has to happen for Office 12 users, the knowledge transfer from other apps or previous Office is likely to be substantial.

One thing Office 12’s developers did very effectively is **task analysis**. In fact, they signed up thousands of Office users to a special program that collected statistics on how frequently they used Office commands and in which order – huge amounts of data that directly drove how commands were grouped into the command tabs, and which commands appear on command tabs as opposed to being buried in deeper dialogs. When a user interface designer can get this kind of data, you can do a lot to improve the usability for an average user. Web site designers are lucky, in this sense, because server logs give it to them for free! Microsoft had to do a lot more work to get it.

Office 2007 also provides more **feedback** about what a command will do, by showing a preview of its effect right in the document while you’re mousing over the command. So if you hover over the Heading 2 option, your document will reformat to show you what the selection would look like with that new style. As long as your computer is fast enough to do it within 100ms, this would be a tremendous improvement to the visibility and feedback of the interface.

## Today's Topics

- Input events
- Event dispatch & propagation
- State machines

Spring 2011

6.813/6.831 User Interface Design and Implementation

5

Today's lecture finishes our look into the mechanics of implementing user interfaces, by examining **input** in more detail. We'll look mainly at keyboard and mouse input, but also multitouch interfaces like those on modern smartphones and tablets. This lecture has two key ideas for thinking about input. First, that **state machines** are a great way to think about and implement tricky input handling (like direct manipulation operations). Second, that events **propagate** through the view tree, and by understanding this process, you can make good design choices about where to attach the listeners that handle them.

## Raw Input Events

- The usual input hardware has state:
  - ~100 keys on the keyboard (down or up)
  - (x,y) mouse cursor position on the screen
  - one, two, or three mouse buttons (down or up)
- A “raw” input event occurs when this state changes
  - jQuery event
  - key pressed or released      keydown, keyup
  - mouse moved                  mousemove
  - button pressed or released    mousedown, mouseup

Spring 2011

6.813/6.831 User Interface Design and Implementation

6

There are two major categories of input events: raw and translated.

A raw event comes from a state transition in the input hardware. Mouse movements, mouse button down and up, and keyboard key down and up are the raw events seen in almost every capable GUI system. A toolkit that does not provide separate events for down and up is poorly designed, and makes it difficult or impossible to implement input effects like drag-and-drop or game controls. And yet some toolkits like that did exist at one time, particularly in the bad old days of handheld and mobile phone programming.



## Translated Events

- Raw events are translated into higher-level events

	<u>jQuery event</u>
– Clicking	click
– Double-clicking	dblclick
– Character typed	keypress
– Entering or exiting an object's bounding box	mouseenter, mouseleave

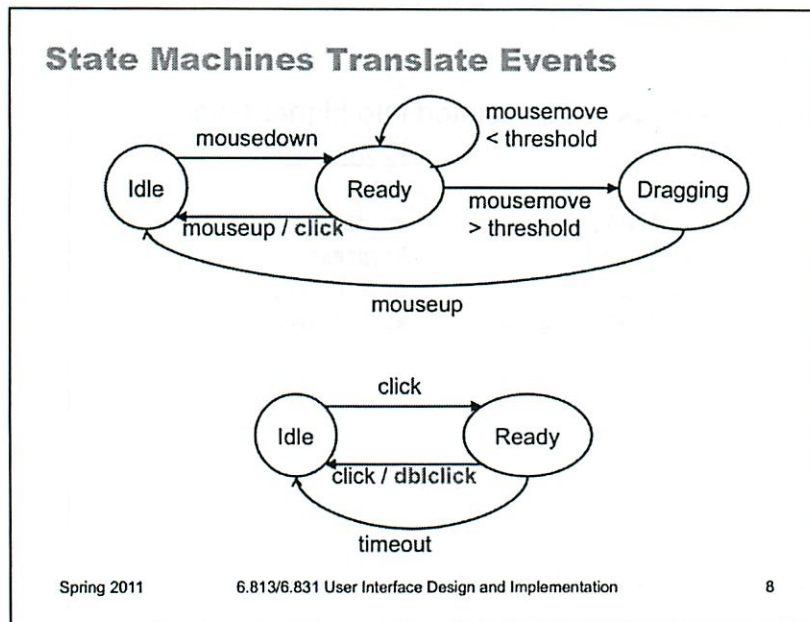
Spring 2011

6.813/6.831 User Interface Design and Implementation

7

For many GUI components, the raw events are too low-level, and must be translated into higher-level events. For example, a mouse button press and release is translated into a mouse click event -- assuming the mouse didn't move much between press and release -- if it did, these events would be interpreted as a drag rather than a click, so a click event isn't produced.

Key down and up events are translated into character-typed events, which take modifiers (Shift/Ctrl/Alt) and input methods (e.g. entering for Chinese characters on a standard keyboard) into account to produce a Unicode character rather than a physical keyboard key. In addition, if you hold a key down, multiple character-typed events may be generated by an autorepeat mechanism (usually built into the operating system or GUI toolkit). When a mouse movement causes the mouse to enter or leave a component's bounding box, entry and exit events are generated, so that the component can give feedback -- e.g., visually highlighting a button, or changing the mouse cursor to a text I-bar or a pointing finger.



Here's our first example of using state machines for input handling. Inside the GUI toolkit, a state machine is handling the translation of raw events into higher-level events. Here's how the click event is generated – after a mousedown and mouseup, as long as the mouse hasn't moved (much) between those two events. Question for you: what is the threshold on your favorite GUI toolkit? If it's measured in pixels, how large is it? Does the mouse exiting the bounding box of the graphical object trigger the threshold regardless of pixel distance?

In this case, the raw events (down, up, move) are still delivered to your application, along with the translated event (click). This means that if your application is handling both the raw events and the translated events, it has to be prepared to expect this. This often comes up with double-click, for example: your application will see two click events before it sees the double-click event. As a result, you can't make click do something incompatible with double-click.

But occasionally, low-level events are **consumed** in the process of translating them to higher-level events. It's a difference you have to pay attention to in your particular toolkit.

## Exercise: Mouse Event Translation

- Try this code at [htmledit.squarefree.com](http://htmledit.squarefree.com)
- Code at <http://pastebin.com/jQzhLZiS>

```
<script src="http://code.jquery.com/jquery-1.7.min.js"></script>
<div id="A" style="width:100px; height:100px; background:lightYellow; padding:5px"></div>
<script>
$(function() {
  $("#A").mousedown(function() { console.log("A down") })
  $("#A").mouseup(function() { console.log("A up") })
  $("#A").click(function() { console.log("A clicked") })
  $("#A").dblclick(function() { console.log("A double-clicked") })
})
</script>
```

- What sequence of events do you see when you double click?
- When does a mouse down and up fail to produce a click? Try to deduce the translation rules used by your browser.
- When does two clicks fail to produce a double click?



## Keyboard Focus

- An object in the view tree has the keyboard focus
  - Keyboard focus gained or lost
    - jQuery event  
focusin,  
focusout
- Changing keyboard focus
  - by user input event: e.g. mouse down, Tab
  - programmatically by a method call
- Not all HTML elements can have keyboard focus by default
  - <div tabindex="-1"> to force ability to take focus

Spring 2011

6.813/6.831 User Interface Design and Implementation

10

The keyboard focus is also part of the state of the input system, but it isn't in the input hardware – instead, the keyboard focus is a particular object in the view tree that currently receives keyboard events. On some X Windows window managers, you can configure the keyboard focus to follow the mouse pointer – whatever view object contains the mouse pointer has the keyboard focus as well. On most windowing systems (like Windows and Mac), however, a mouse down is the more common way to change the focus.

## Properties of an Input Event

- Mouse position (X,Y)
- Mouse button state
- Modifier key state (Ctrl, Shift, Alt, Meta)
- Timestamp
  - Why is timestamp important?
- Keyboard key, character, or mouse button that changed
  - jQuery event.which overloads this for mouse events and key events

Spring 2011

6.813/6.831 User Interface Design and Implementation

11

Input events carry with them some or all of these properties, which represent the state of the input hardware immediately after the event occurred.

On most systems, all events include the modifier key state, since some mouse gestures are modified by Shift, Control, and Alt. Some systems include the mouse position and button state on all events; some put it only on mouse-related events.

The timestamp indicates when the input was received, so that the system can time features like autorepeat and double-clicking. It is essential that the timestamp be a property of the event, rather than just read from the clock when the event is handled. Events are stored in a queue, and an event may languish in the queue for an uncertain interval until the application actually handles it, so it's necessary for the time of the event to be captured as close to the event's actual occurrence (the press or release in the event object itself).

Keyboard events can be trickier to handle than mouse events because identifying the key involved in the event is not always easy. Particularly for cross-platform toolkits (HTML, Flash, Java), there may be a variety of different keyboard hardware with different sets of keys, and in HTML/Javascript, different browsers may work differently. There is the further complication that translated key events (the "character typed" event) do not represent a **keystroke** (like Shift or PgUp or the A key), but rather a **character** (like "a" or "A" or "%"). Keystrokes are identified by physical keys on the keyboard; characters are identified by values in a character set (like Unicode or ASCII). In jQuery, do not treat keydown/keyup and keypress as interchangeable; their names may be similar, but the parameters of the events are different.

## Exercise: Keyboard Events

- Try this code at [htmledit.squarefree.com](http://htmledit.squarefree.com)
- Code at <http://pastebin.com/pgW8vTwQ>

```
<script src="http://code.jquery.com/jquery-1.7.min.js"></script>
<textarea id="A"></textarea>
<script>
$(function() {
  $("#A").keydown(function(event) { console.log("A down " + event.which) })
  $("#A").keyup(function(event) { console.log("A up " + event.which) })
  $("#A").keypress(function(event) { console.log("A press " + event.which) })
})
</script>
```
- Type a letter key. What sequence of events do you see? What do the values mean?
- Press a modifier key, like Shift. What events do you see?
- Hold down a letter key. What events do you see?

## Event Queue

- Events are stored in a queue
  - User input tends to be bursty
  - Queue saves application from hard real time constraints (i.e., having to finish handling each event before next one might occur)
- Mouse moves are coalesced into a single event in queue
  - If application can't keep up, then sketched lines have very few points

Spring 2011

6.813/6.831 User Interface Design and Implementation

13

User input tends to be bursty – many seconds may go by while the user is thinking, followed by a flurry of events. The event queue provides a buffer between the user and the application, so that the application doesn't have to keep up with each event in a burst. Recall that perceptual fusion means that the system has 100 milliseconds in which to respond.

Edge events (button down and up events) are all kept in the queue unchanged. But multiple events that describe a continuing state – in particular, mouse movements – may be **coalesced** into a single event with the latest known state. Most of the time, this is the right thing to do. For example, if you're dragging a big object across the screen, and the application can't repaint the object fast enough to keep up with your mouse, you don't want the mouse movements to accumulate in the queue, because then the object will lag behind the mouse pointer, diligently (and foolishly) following the same path your mouse did.

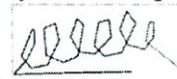
Sometimes, however, coalescing hurts. If you're sketching a freehand stroke with the mouse, and some of the mouse movements are coalesced, then the stroke may have straight segments at places where there should be a smooth curve. If something running in the background causes occasional long delays, then coalescing may hurt even if your application can usually keep up with the mouse.



## Exercise: Mouse Coalescing

- Try this code at [html5edit.squarefree.com](http://html5edit.squarefree.com)
- Code at <http://pastebin.com/xR1hP0f4>

```
<script src="http://code.jquery.com/jquery-1.7.min.js"></script>
<canvas id="A" width="400" height="300" style="border: 1px dashed"></canvas>
<script>
$(function() {
  var ctx = $("#A").get(0).getContext("2d")
  var pos = $("#A").offset()
  ctx.translate(-pos.left, -pos.top) // move origin of canvas to origin of document, so we can
  use event.pageX/pageY directly
  $("#A").mousemove(function(event) { sleep(0); ctx.lineTo(event.pageX, event.pageY);
  ctx.stroke() })
  function sleep(ms) { var wake = now() + ms; while (now() < wake) { } }
  function now() { return new Date().getTime() }
})
</script>
```
- Change the `sleep(0)` to `sleep(100)`. What happens to your scribbling?  
Explain it in terms of event coalescing.



Spring 2012

6.813/6.831 User Interface Design and Implementation

### Example: Without Coalescing

- Can't turn off event coalescing in Javascript
  - So we'll simulate our own event queue
  - Code at <http://pastebin.com/PANDuYmj>
- Try it first, then increase DELAY to 100 ms.  
What happens?



Spring 2011

6.813/6.831 User Interface Design and Implementation

15

Event coalescing can't be easily disabled in HTML/Javascript, so we'll simulate what life would be like without coalescing by creating our own queue of mouse points (queue and pushEvent() in the code below) and an event handler that reads from it (eventLoop()). Try the code as given first, and then try increasing DELAY to 50 milliseconds. What happens?

## Event Loop

- While application is running
  - Block until an event is ready
  - Get event from queue
  - Translate raw event into higher-level events
    - Generates double-clicks, characters, focus, enter/exit, etc.
    - Translated events are put into the queue
  - Dispatch event to target component
- Who provides the event loop?
  - High-level GUI toolkits do it internally (Java Swing, VB, C#, HTML)
  - Low-level toolkits require application to do it (MS Win, Palm, Java SWT)

Spring 2011

6.813/6.831 User Interface Design and Implementation

16

The event loop reads events from the queue and dispatches them to the appropriate components in the view hierarchy. On some systems (notably Microsoft Windows), the event loop also includes a call to a function that translates raw events into higher-level ones. On most systems, however, translation happens when the raw event is added to the queue, not when it is removed.

Every GUI program has an event loop in it somewhere. Some toolkits require the application programmer to write this loop (e.g., Win32); other toolkits have it built-in (e.g., Java Swing).

## Event Dispatch & Propagation

- Dispatch: choose target component for event
  - Key event: component with keyboard focus
  - Mouse event: component under mouse (**hit testing**)
    - **Mouse capture**: any component can grab mouse temporarily so that it receives all mouse events (e.g. for drag & drop)
- Propagation: event bubbles up hierarchy
  - If target component doesn't handle event, the event passes up to its parent, and so on up the tree
- Consumption: event stops propagating
  - May be automatic (because some component finally handles it) or manual (keeps going unless explicitly stopped)

Spring 2011

6.813/6.831 User Interface Design and Implementation

17

Event dispatch chooses a component to receive the event. Key events are sent to the component with the keyboard focus, and mouse events are generally sent to the component under the mouse, using hit testing to determine the visible component that contains the mouse position and is topmost (in z order).

An exception is **mouse capture**, which allows any component to grab all mouse events after a mouse button was pressed over that component, for as long as the button is held down. This is essentially a mouse analogue for keyboard focus. Mouse capture is done automatically by Java when you hold down the mouse button to drag the mouse. Other UI toolkits give the programmer the ability to turn it on or off – in the Windows API, for example, you'll find a `SetCapture` function.

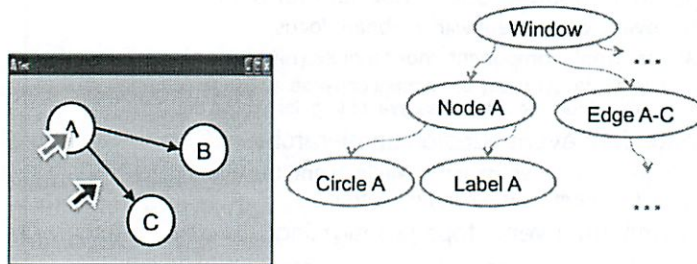
If the target component has no handler for the event, the event propagates up the view hierarchy looking for some component able to handle it. If an event bubbles up to the top without being handled, it is discarded.

In many GUI toolkits, the event stops propagating automatically after reaching a component that handles it; none of that component's ancestors see the event. Java Swing behaves this way; an event propagates up through the tree until it finds a component with at least one listener registered for the event, and then propagation stops automatically. (Note that this doesn't necessarily mean that only one *listener* sees the event. The component that finally handles the event may have more than one listener attached to it, and all of those listeners will receive the event, in some arbitrary order. But no listeners attached to components higher in the tree will see it.)

In some toolkits, however, event propagation is under the control of the programmer, and events continue propagating up the tree unless explicitly stopped. HTML/Javascript behaves this way, as does Adobe Flex. In these toolkits, an event can be stopped by calling `stopPropagation()` on its event object.



## Hit Testing and Event Propagation



Spring 2011

6.813/6.831 User Interface Design and Implementation

18

Here are some examples of how mouse events are dispatched and propagated. The window on the left has the view hierarchy shown on the right, in which each graph node is represented by a Node component with two children, a Circle (displaying a filled white circle with a black outline) and a text Label (displaying a text string, such as “A” or “B”).

First consider the green mouse cursor; suppose it just arrived at this point. Then a mouse-move event is created and dispatched to the topmost component whose bounding box contains that point, which is Label A. If Label A doesn’t handle the mouse-move event, then the event is propagated up to Node A; if that doesn’t handle the event either, it’s propagated to Window, and then discarded. Notice that Circle A never sees the event, because event propagation goes up the tree, not down through z-order layers.

Now consider the blue mouse cursor. What component will be the initial target for a mouse-move event for this point? The answer depends on how hit-testing is done by the toolkit. Some toolkits support only rectangular bounding-box hit testing, in which case Edge A-C (whose bounding box contains the mouse point) will be the event target. Other toolkits allow hit testing to be overridden and controlled by components themselves, so that Edge A-C could test whether the point actually falls on (or within some small threshold of) the actual line it draws. Java Swing supports this by overriding `Component.contains()`. If Edge A-C rejects the point, then the next component in z-order whose bounding box contains the mouse position is the window itself, so the event would be dispatched directly to the window.

## Javascript Event Models

- Events propagate in different directions on different browsers
  - Netscape 4: downwards from root to target
  - Internet Explorer: upwards from target to root
  - W3C standardized by combining them: first downwards ("capturing"), then upwards ("bubbling")
    - Firefox, Opera, Safari

Spring 2011

6.813/6.831 User Interface Design and Implementation

19

The previous slides describe how virtually all desktop toolkits do event dispatch and propagation. Alas, the Web is not so simple.

Early versions of Netscape propagated events *down* the view hierarchy, not up. On the Web, the view hierarchy is a tree of HTML elements. Netscape would first determine the target of the event, using mouse position or keyboard focus, as we explained earlier. But instead of sending the event directly to the target, it would first try sending it to the root of the tree, and so forth down the ancestor chain until it reached the target. Only if none of its ancestors wanted the event would the target actually receive it.

Alas, Internet Explorer's model was exactly the opposite – like the conventional desktop event propagation, IE propagated events upwards. If the target had no registered handler for the event (and no default behavior either, like a button or hyperlink has for click events), then the event would propagate upwards through the tree.

The W3C consortium, in its effort to standardize the Web, combined the two models, so that events first propagate downwards to the target (a phase called "event capturing", not to be confused with mouse capture), and then back upwards again ("event bubbling"). You can register event handlers for either or both phases if you want. Modern standards-compliant browsers, like Firefox and Opera, support this model; so does Adobe Flex.

One advantage of this two-phase event propagation model is that it gives you a lot more flexibility as a programmer to override the behavior of other components. By attaching a capturing listener high up in the component hierarchy, you can handle the events yourself and prevent other components from even seeing them. For example, if you want to implement an "edit mode" for your UI, in which the user can click and drag around standard widgets like buttons and textboxes, you can do that easily with a single capturing listener attached to the top of your UI tree. In the traditional desktop event propagation model, it would be harder to prevent the buttons and textboxes from trying to interpret the click and drag events themselves, and you would have to add listeners to every single widget.

## Exercise: Event Propagation

- Try this code at [htmledit.squarefree.com](http://htmledit.squarefree.com)

- Code at <http://pastebin.com/pF3w9nw0>

```
<script src="http://code.jquery.com/jquery-1.7.min.js"></script>
<div id="A" style="width:100px;height:100px;background:lightYellow;padding:5px">A
<div id="B" style="margin:20px; padding:5px; background:lightPink;">B</div>
</div>
<script>
$(function() {
  $("#B").on("mousedown mouseup", function(event) { console.log(this.id + " got " +
    event.target.id + " " + event.type) })
  $("#A").on("mousedown mouseup", function(event) { console.log(this.id + " got " +
    event.target.id + " " + event.type) })
  //$(window).on("mousedown mouseup", function(event) { console.log("window got " +
    event.target.id + " " + event.type) })
})
</script>
```

- Click on the B node. Which nodes get the events?
  - Try consuming the event in B's listener with `event.stopPropagation()`.

## Exercise: Mouse Capture

- Use the same code from the previous slide
- Add mousemove to the "mousedown mouseup" list. Then click and drag the B node.
  - Can you implement drag-and-drop or a scrollbar with this behavior?
- Now uncomment the window event binding, and drag the B node again -- in particular, drag it outside the entire browser window. What happens?



### Multitouch Dispatch (iPhone)

- Multitouch input events have more than one (x,y) point (fingers on screen)
  - Touch-down event dispatches to the component containing it (which also captures future touch-moves and touch-up for that finger)
  - Touch events carry information about all fingers currently touching
  - A component can turn on “exclusive touch” to receive all touch-down events even if they fall outside it

Spring 2011

6.813/6.831 User Interface Design and Implementation

22

Multitouch interfaces like the Apple iPhone introduce a few wrinkles into the event dispatch story. Instead of having a single mouse position where the event occurs, a multitouch interface may have multiple points (fingers) touching the screen at once. Which of these points is used to decide which component gets the event?

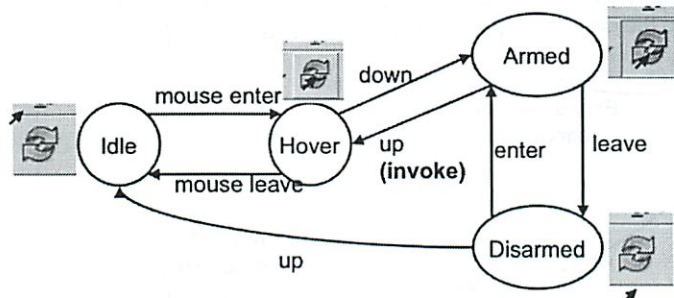
Here's how the iPhone does it. Each time a finger touches down on the screen, the location of the new touch-down is used to dispatch the touch-down event. All events carry along information about all the fingers that are currently touching the screen, so that the component can recognize multitouch gestures like pinching fingers together or rotating the fingers. (This is a straightforward extension of keyboard and mouse events, in fact – most input events carry along information about what keyboard modifiers are currently being held down, and often the current mouse position and mouse button state as well.)

Two kinds of event capture are used in the iPhone. First, after a touch-down event is dispatched to the component that it touched first, that component automatically captures the events about all future moves of that finger, even if it strays outside the bounds of the component, until the finger finally leaves the screen (touch-up). This is similar to the automatic mouse capture used by Java Swing when the mouse is dragged.

Second, a component can also turn on its “exclusive touch” property, which means that if the first touch on the screen (after a period of no fingers touching) is dispatched to that component, then all future touch events are captured by that component, until all fingers are released again. (Apple, “Event Handling”, *iPhone Application Programming Guide*, 2007).

## Designing a Controller

- A controller is a finite state machine
- Example: push button



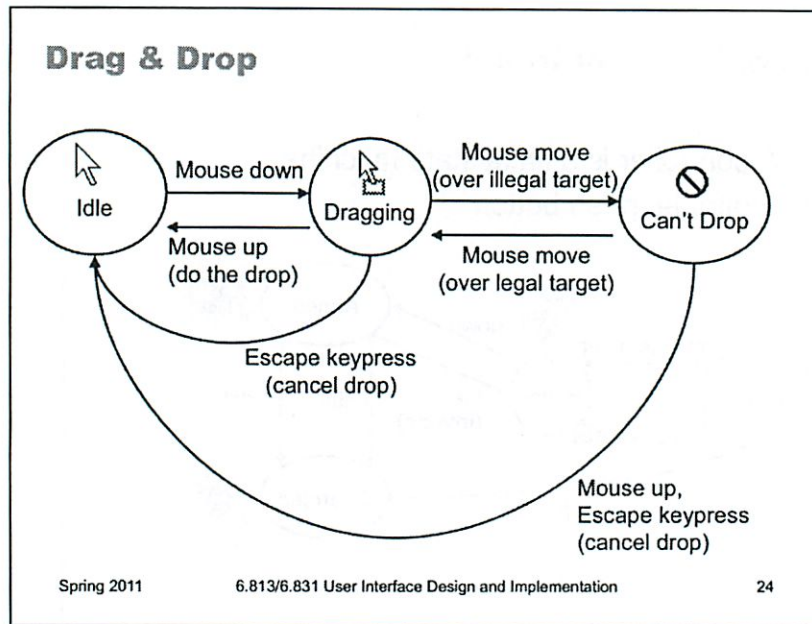
Spring 2011

6.813/6.831 User Interface Design and Implementation

23

Now let's look at how components that handle input are typically structured. A controller in a direct manipulation interface is a **state machine**. Here's an example of the state machine for a push button's controller. **Idle** is the normal state of the button when the user isn't directing any input at it. The button enters the **Hover** state when the mouse enters it. It might display some feedback to reinforce that it affords clickability. If the mouse button is then pressed, the button enters the **Armed** state, to indicate that it's being pushed down. The user can cancel the button press by moving the mouse away from it, which goes into the **Disarmed** state. Or the user can release the mouse button while still inside the component, which invokes the button's action and returns to the **Hover** state.

Transitions between states occur when a certain input event arrives, or sometimes when a timer times out. Each state may need different feedback displayed by the view. Changes to the model or the view occur on transitions, not states: e.g., a push button is actually invoked by the release of the mouse button.

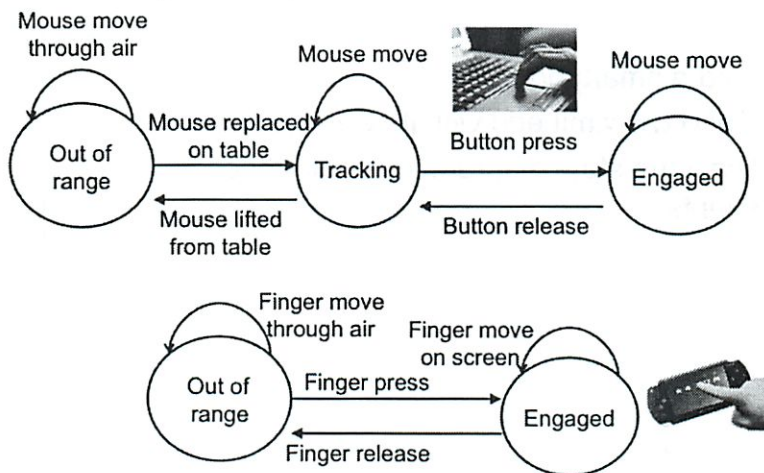


Here's a state machine suitable for drag & drop.

Notice how each state of the machine produces different visual feedback, in this case the shape of the cursor. (The pushbutton on the last page had the same property.) This is a common case in input implementation, since different states of an input controller often represent different **modes** from the user's point of view, and distinguishing those modes with visual feedback helps reduce mode errors.

Visual feedback can also happen on the transitions, but it may have to be animated to be effective, because the transitions are very brief (like pressing or releasing a button).

## Modeling the Input Device Itself



Spring 2012

6.813/6.831 User Interface Design and Implementation

25



### **Exercise: Touch Scrolling**

- Find a smartphone
- Go to [www.mit.edu](http://www.mit.edu) with its web browser
- Draw the state diagram for single-touch events

## Summary

- Input events are raw and translated
- Events are **dispatched** to a target view and **propagated** up (or down then up) the view hierarchy
- State machines are a useful pattern for thinking about input

GR2 due next Sun

GR3 due Mon, the next Sun

PS2/AS2 at Mon, due at end of Spring Break  
- must be a user study participant for others

Next week

1-2PM lectures ~~only~~ only for Grad students

3-5PM for building paper prototypes Mon

- paper, scissors, markers available

Wed-Fri can test prototype

Walker Gym

UI Hall of Fame/shame ~~UI~~ ~~Storyboard~~ / Acrobat

- No scrollbars
  - can click + drag
    - drag down, means img moves up
  - Acrobat moves the other way
  - Common in UI - that 2 things done opposite
- next pg
- like dragging glass  
page  
or adjusting scrollbar
- iPhone does this

②

drag downing ↑

Scrollwheel Windows  
like dragging glass  
like adjusting scrollbar thumb  
Old OS X scrollwheel

drag & imgl

iPhone  
like dragging page in Acrobat  
Mac OS Lion scroll wheel  
about half turn it off



hard to tell the difference

made errors common w/ people that use OS X Lion

Early scrollbars → Xerox Star

more complex

button for jump 1 pg

↳ now if click in blank area → moves 1pg

Now scrollbar height is ~~average~~ for dragging and  
tells you length of document



③ Flash designers try to reinvent scrollbar

- but they do implement all features
- even though looks visually uniform

Or what is Thumb and what is track?

- do thumb bright

## Nano Quiz

### Consuming events

Q1

Must delay events so waiting for double click time period - ~~no~~ must wait for double click

So (A) efficiency

(B) Perceptual fusion - events take a long time to appear

$\tau < 100$  ms appears instantaneous

## View tree

3 diff possible orders

1 common, 2 specific to web

no X - not in path on tree

B. ZX  $\in$  st bubbling order UI, JQuery, UI

D. XZX  $\in$  W3C

E. ZXZ  $\in$  no!

9

## Coalescing

- not mouse capture
  - better for responsive drag & drop
  - keeps your item up w/ the cursor
- 

## Today's Prototyping

- paper
  - computer
  - Wizard of Oz
- 

## Prototyping

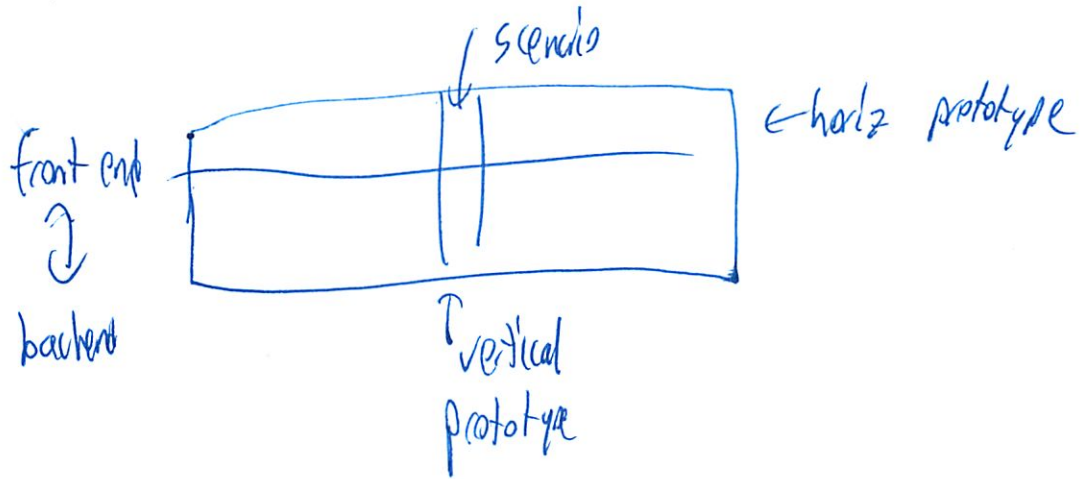
### Why?

Get feedback easier & cheaper  
Easier to change / throw away  
Code is hard to build

5

## Fidelity is Multidimensional

- Breadth - % features covered
- Depth - degree of functionality



So scenario - narrow and shallow  
only a few features  
(ie no password recovery)  
don't fully work

Look - hand drawn

Feed - input method might be different  
except for tablets

6

## Paper prototype

like a sketch

but supercharged

- using a human to move around pieces
- "pop up" pop up boxes

build it quickly

- don't make it too fancy
- kindergarten skills
- not a work of art
- "emotionally cheap"

easy to make changes

- might just be mental change

focus on the big picture

- not details - like the font

Non programmers can help

- designers on your team



## ② Materials

Post it note glue

- make anything into a post it

Transparency

---

Make it larger than life

- Unless smartphone

- even then - big so you can see it too

Monocrone of course

No animation

- describe orally

Can use printouts of real screenshots

- Or use a real photo

Make it big!

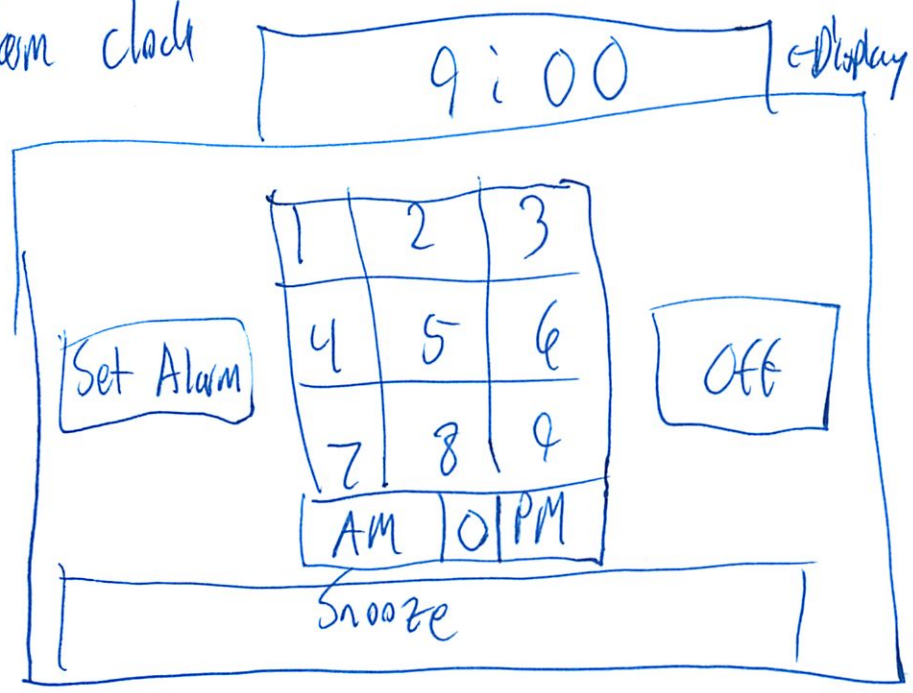
- 8x11 is even too small for a monitor

---

Can cut features if people don't want

8

Alarm clock



9

## Alarm Clock Assignment

move off button to side corner

People will have diff ways

Tell him what display says

0:00

0:09

8:00

4:00

What it do alt way

Set alarm → 9 → 0 → 0 → Set alarm

---

When you change prototype its very visible

## L11: Prototyping

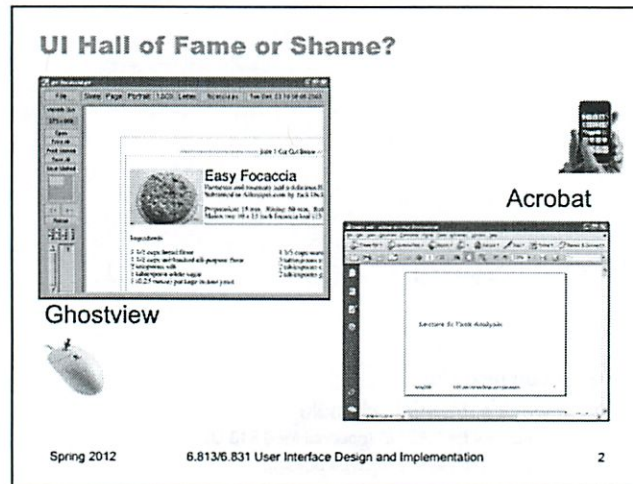
- GR2 due next Sun
- Next week's unusual schedule
  - lectures for 6.831 G (optional for 6.813 U)
  - M 3-5 pm paper prototype building
  - WF 3-5 pm prototype testing & RS2 testing

Spring 2012

6.813/6.831 User Interface Design and Implementation

1





On the left is Ghostview, a Unix program that displays Postscript files. Ghostview has no scrollbars; instead, it scrolls by **direct manipulation** of the page image. Clicking and dragging on the page scrolls it around, and it turns out that dragging downward moves the page image upward. That is, when you drag the mouse down, more of the page image come into view at the bottom of the window. Does this make sense? What mental model, or physical analogy, does this correspond to?

On the right is Acrobat, which displays PDF files. Acrobat has scrollbars, but you can also directly manipulate the page image, as in Ghostview – only now clicking and dragging downward moves the page downward, revealing more page image at the top of the window. What mental model does this correspond to?

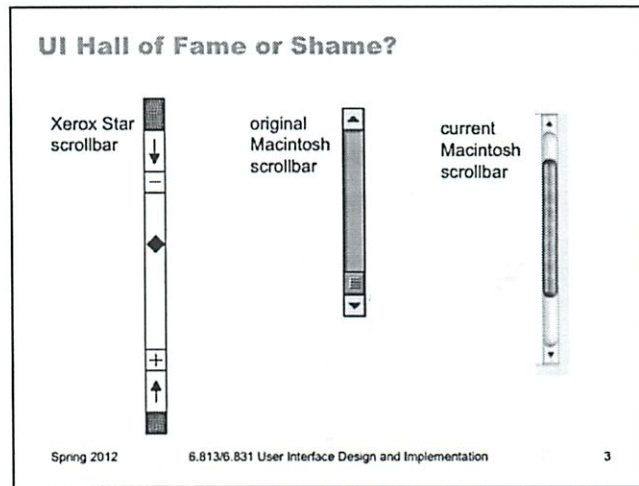
What if you used both Acrobat and Ghostview frequently (one for PDF, say, and the other for Postscript)?

The Ghostview model is like a glass window that you're dragging around on top of the page. You're pressing on the glass and pushing it to the left should indeed make more of the page become visible on the left.

The Acrobat model treats the window like an empty frame, and you're actually reaching through and pushing the page itself around. Pushing it to the left should make more of the page visible on the right.

So two different physical analogies are possible here. Principles of direct manipulation alone don't help us decide. And Ghostview at least uses a **consistent** model – the light gray rectangle in the leftside toolbar (underneath the Save Marked button) is a miniature representation of the position of the glass window over the page, and you push it around using the same dragging direction that you use on the page image itself.

If you had to use both Acrobat and Ghostview frequently, however, the **inconsistency** between them would be very painful. Yet we do have to use **\*both\*** models these days: direct-touch scrolling (smartphones and tablets) uses one model, and scrollbars and mouse scrollwheels use the opposite model. The directness vs. indirectness of the interaction and differences in the low-level muscle interaction may give enough cues to keep the model straight. But touchpads on laptops may be the hardest point of overlap, since they are indirect, and yet use the same finger swiping motion as direct touch. Apple decided to make a significant transition recently with Mac OS X Lion, switching its default touchpad scrolling from one mental model (the scrollwheel) to the other (direct touch).



Let's look at scrolling some more. Scrollbars have evolved considerably over the history of graphical user interfaces.

The Xerox Star offered the earliest incarnation of a graphical user interface, and its scrollbar already had many of the features we recognize in modern scrollbars, such as a **track** containing a scrollbar **thumb** (the handle that you can drag up and down). Even its arrow buttons do the same thing – e.g., pushing on the top button makes more of the page appear at the top of the window, just like the modern scrollbar. But they're *labeled* the opposite way – the top button has a *down* arrow on it. Why is that? What mental model would lead you to call that button *down*? Is that consistent with the mental model of the scrollbar thumb?

Another interesting difference between the Star scrollbar and modern scrollbars are the - and + buttons, which move by whole pages. This functionality hasn't been eliminated from the modern scrollbar; instead, the modern scrollbar just drops the obvious affordance for it. You can still click on the track above or below the thumb in order to jump by whole pages.

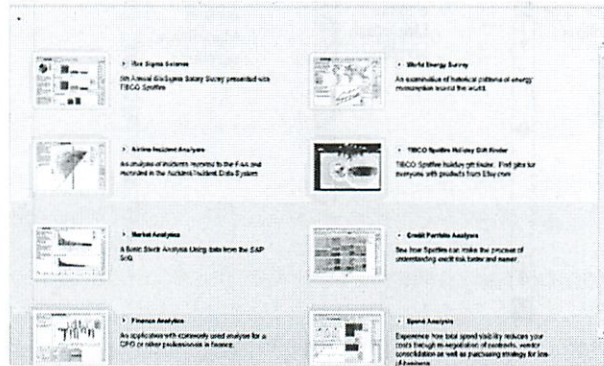
The button labels actually refer to the page-pushing model of scrolling – when you click on the top button, the page itself moves down. Alas, this is completely inconsistent with the thumb, which uses the glass-pane model of scrolling – the thumb represents the glass pane, and you can drag it up and down the document. So the Xerox Star scrollbar couldn't make up its mind about which model to represent.

Removing the +/- buttons makes the scrollbar **simpler**, by forcing the track to do double-duty. But it also makes the page-jumping functionality less **visible**, so new users are unlikely to discover it on their own.

We can also say something about **natural mapping** here – the arrow buttons are on opposite sides of the scrollbar, so their positions map appropriately to the direction that they move the thumb. (But, alas, the Xerox Star's arrow labels disagree with the natural mapping.)

Another improvement in the scrollbar is that the height of the thumb now indicates what fraction of the whole document is visible.

## UI Hall of Fame or Shame?



Spring 2012

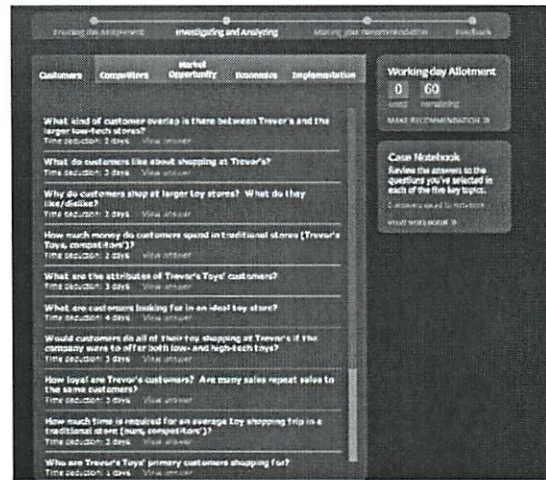
6.813/6.831 User Interface Design and Implementation

4

Flash designers love to reimplement scrollbar widgets – and they do it wrong. Today we'll pick on <http://spotfire.tibco.com/Demo/>. What parts of the scrollbar did they get right, and what parts did they get wrong? The problems are associated with learnability (i.e. consistency with other scrollbars), efficiency, and visibility.



## UI Hall of Fame or Shame?



Spring 2012

6.813/6.831 User Interface Design and Implementation

5

Here's another example of a Flash scrollbar that lacks clear affordances (from [http://www.bcg.com/join\\_bcg/interview\\_prep/interactive\\_case/default.aspx](http://www.bcg.com/join_bcg/interview_prep/interactive_case/default.aspx)). It turns out that the bright, highly salient part of this scrollbar is NOT the thumb, but the track! The thumb is a dark color – in fact, it's the same dark color that's used as a background color of some of the widgets on the right (Used and Remaining). (example suggested by Eryn Maynard)



## Today's Topics

- Paper prototypes
- Computer prototypes
- Wizard of Oz prototypes

Spring 2012

6.813/6.831 User Interface Design and Implementation

8

Today we're going to talk about prototyping: producing cheaper, less accurate renditions of your target interface. Prototyping is essential in the early iterations of a spiral design process, and it's useful in later iterations too.

## Why Prototype?

- Get feedback earlier, cheaper
- Experiment with alternatives
- Easier to change or throw away

Spring 2012

6.813/6.831 User Interface Design and Implementation

9

We build prototypes for several reasons, all of which largely boil down to cost.

First, prototypes are much faster to build than finished implementations, so we can evaluate them sooner and get early feedback about the good and bad points of a design.

Second, if we have a design decision that is hard to resolve, we can build multiple prototypes embodying the different alternatives of the decision.

Third, if we discover problems in the design, a prototype can be changed more easily, for the same reasons it could be built faster. Prototypes are more malleable. Most important, if the design flaws are serious, a prototype can be **thrown away**. It's important not to commit strongly to design ideas in the early stages of design. Unfortunately, writing and debugging a lot of code creates a psychological sense of commitment which is hard to break. You don't want to throw away something you've worked hard on, so you're tempted to keep some of the code around, even if it really should be scrapped. (Alan Cooper, "The Perils of Prototyping", 1994. [http://www.cooper.com/journal/1999/09/the\\_perils\\_of\\_prototyping.html](http://www.cooper.com/journal/1999/09/the_perils_of_prototyping.html) )

Most of the prototyping techniques we'll see in this lecture actually force you to throw the prototype away. For example, a paper mockup won't form any part of a finished software implementation. This is a good mindset to have in early iterations, since it maximizes your creative freedom.

## Prototype Fidelity

- Low fidelity: omits details
- High fidelity: more like finished product

Spring 2012

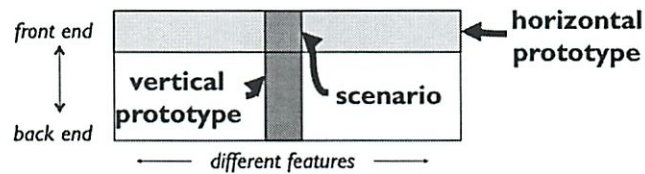
6.813/6.831 User Interface Design and Implementation

10

An essential property of a prototyping technique is its **fidelity**, which is simply how similar it is to the finished interface. Low-fidelity prototypes omit details, use cheaper materials, or use different interaction techniques. High-fidelity prototypes are very similar to the finished product.

## Fidelity is Multidimensional

- Breadth: % of features covered
  - Only enough features for certain tasks
- Depth: degree of functionality
  - Limited choices, canned responses, no error handling



Spring 2012

6.813/6.831 User Interface Design and Implementation

11

Fidelity is not just one-dimensional, however. Prototypes can be low- or high-fidelity in various different ways (Carolyn Snyder, *Paper Prototyping*, 2003).

**Breadth** refers to the fraction of the feature set represented by the prototype. A prototype that is low-fidelity in breadth might be missing many features, having only enough to accomplish certain specific tasks. A word processor prototype might omit printing and spell-checking, for example.

**Depth** refers to how deeply each feature is actually implemented. Is there a backend behind the prototype that's actually implementing the feature? Low-fidelity in depth may mean limited choices (e.g., you can't print double-sided), canned responses (always prints the same text, not what you actually typed), or lack of robustness and error handling (crashes if the printer is offline).

A diagrammatic way to visualize breadth and depth is shown (following Nielsen, *Usability Engineering*, p. 94). A **horizontal prototype** is all breadth, and little depth; it's basically a frontend with no backend. A **vertical prototype** is the converse: one area of the interface is implemented deeply. The question of whether to build a horizontal or vertical prototype depends on what risks you're trying to mitigate. In user interface design, horizontal prototypes are more common, since they address usability risk. But if some aspect of the application is a risky implementation – you're not sure if it can be implemented to meet the requirements – then you may want to build a vertical prototype to test that.

A special case lies at the intersection of a horizontal and a vertical prototype. A **scenario** shows how the frontend would look for a single concrete task.



## More Dimensions of Fidelity

- Look: appearance, graphic design
  - Sketchy, hand-drawn
- Feel: input method
  - Pointing & writing feels very different from mouse & keyboard

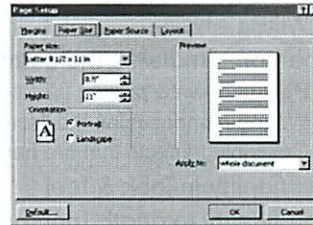
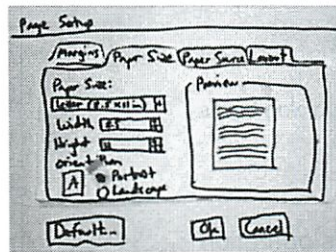
Spring 2012

6.813/6.831 User Interface Design and Implementation

12

Two more crucial dimensions of a prototype's fidelity are, loosely, its look and its feel. **Look** is the appearance of the prototype. A hand-sketched prototype is low-fidelity in look, compared to a prototype that uses the same widget set as the finished implementation. **Feel** refers to the physical methods by which the user interacts with the prototype. A user interacts with a paper mockup by pointing at things to represent mouse clicks, and writing on the paper to represent keyboard input. This is a low-fidelity feel for a desktop application (but it may not be far off for a tablet PC application).

## Comparing Fidelity of Look & Feel



Spring 2012

6.813/6.831 User Interface Design and Implementation

13

Here's the same dialog box in both low-fi and high-fi versions. How do they differ in the kinds of things you can test and get feedback about?

## Paper Prototype

- Interactive paper mockup
  - Sketches of screen appearance
  - Paper pieces show windows, menus, dialog boxes
- Interaction is natural
  - Pointing with a finger = mouse click
  - Writing = typing
- A person simulates the computer's operation
  - Putting down & picking up pieces
  - Writing responses on the "screen"
  - Describing effects that are hard to show on paper
- Low fidelity in look & feel
- High fidelity in depth (person simulates the backend)

Spring 2012

6.813/6.831 User Interface Design and Implementation

14

**Paper prototypes** are an excellent choice for early design iterations. A paper prototype is a physical mockup of the interface, mostly made of paper. It's usually hand-sketched on multiple pieces, with different pieces showing different menus, dialog boxes, or window elements.

The key difference between mere sketches and a paper prototype is **interactivity**. A paper prototype is brought to life by a design team member who simulates what the computer would do in response to the user's "clicks" and "keystrokes", by rearranging pieces, writing custom responses, and occasionally announcing some effects verbally that are too hard to show on paper. Because a paper prototype is actually interactive, you can actually user-test it: give users a task to do and watch how they do it.

A paper prototype is clearly low fidelity in both look and feel. But it can be arbitrarily high fidelity in breadth at very little cost (just sketching, which is part of design anyway). Best of all, paper prototypes can be **high-fidelity in depth** at little cost, since a human being is simulating the backend.

Much of the material about paper prototyping in this lecture draws on the classic paper by Rettig et al, "Prototyping for tiny fingers" (CACM 1994), and Carolyn Snyder's book *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces* (Morgan Kaufmann, 2003).

## Why Paper Prototyping?

- Faster to build
  - Sketching is faster than programming
- Easier to change
  - Easy to make changes between user tests, or even *during* a user test
  - No code investment– everything will be thrown away (except the design)
- Focuses attention on big picture
  - Designer doesn't waste time on details
  - Customer makes more creative suggestions, not nitpicking
- Nonprogrammers can help
  - Only kindergarten skills are required

Spring 2012

6.813/6.831 User Interface Design and Implementation

15

But why use paper? And why hand sketching rather than a clean drawing from a drawing program?

Hand-sketching on paper is faster. You can draw many sketches in the same time it would take to draw one user interface with code. For most people, hand-sketching is also faster than using a drawing program to create the sketch.

Paper is easy to change. You can even change it during user testing. If part of the prototype was a problem for one user, you can scratch it out or replace it before the next user arrives. Surprisingly, paper is more malleable than digital bits in many ways.

Hand-sketched prototypes in particular are valuable because they focus attention on the issues that matter in early design without distracting anybody with details. When you're sketching by hand, you aren't bothered with details like font, color, alignment, whitespace, etc. In a drawing program, you would be faced with all these decisions, and you might spend a lot of time on them – time that would clearly be wasted if you have to throw away this design. Hand sketching also improves the feedback you get from users. They're less likely to nitpick about details that aren't relevant at this stage. They won't complain about the color scheme if there isn't one. More important, however, a hand-sketch design seems less finished, less set in stone, and more open to suggestions and improvements. Architects have known about this phenomenon for many years. If they show clean CAD drawings to their clients in the early design discussions, the clients are less able to discuss needs and requirements that may require radical changes in the design. In fact, many CAD tools have an option for rendering drawings with a "sketchy" look for precisely this reason.

A final advantage of paper prototyping: no special skills are required. So graphic designers, usability specialists, and even users can help create prototypes and operate them.



## Tools for Paper Prototyping

- White poster board (11"x14")
  - For background, window frame
- Big (unlined) index cards (4"x6", 5"x8")
  - For menus, window contents, and dialog boxes
- Restickable glue
  - For keeping pieces fixed
- White correction tape
  - For text fields, checkboxes, short messages
- Overhead transparencies
  - For highlighting, user "typing"
- Photocopier
  - For making multiple blanks
- Pens & markers, scissors, tape

Spring 2012

6.813/6.831 User Interface Design and Implementation

16

Here are the elements of a paper prototyping toolkit.

Although standard (unlined) paper works fine, you'll get better results from sturdier products like **poster board** and **index cards**. Use poster board to draw a static background, usually a window frame. Then use index cards for the pieces you'll place on top of this background. You can cut the index cards down to size for menus and window internals.

**Restickable Post-it Note glue**, which comes in a roll-on stick, is a must. This glue lets you make all of your pieces sticky, so they stay where you put them. You can find this glue at Pearl Arts in Central Square; it's not found in the Coop.

**Post-it correction tape** is another essential element. It's a roll of white tape with Post-it glue on one side. Correction tape is used for text fields, so that users can write on the prototype without changing it permanently. You peel off a length of tape, stick it on your prototype, let the user write into it, and then peel it off and throw it away. Correction tape comes in two widths, "2 line" and "6 line". The 2-line width is good for single-line text fields, and the 6-line width for text areas. You can get correction tape at the Office Max in East Cambridge.

**Overhead transparencies** are useful for two purposes. First, you can make a selection highlight by cutting a piece of transparency to size and coloring it with a transparency marker. Second, when you have a form with several text fields in it, it's easier to just lay a transparency over the form and let the users write on that, rather than sticking a piece of correction tape in every field. Pearl Arts in Central Square sells **colored transparencies** that you can use for selection highlighting.

If you have many similar elements in your prototype, a **photocopier** can save you time.

And, of course, the usual kindergarten equipment: pens, markers, scissors, tape.

### Tips for Good Paper Prototypes

- Make it larger than life
- Make it monochrome
- Replace tricky visual feedback with audible descriptions
  - Tooltips, drag & drop, animation, progress bar
- Keep pieces organized
  - Use folders & open envelopes

Spring 2012

6.813/6.831 User Interface Design and Implementation

17

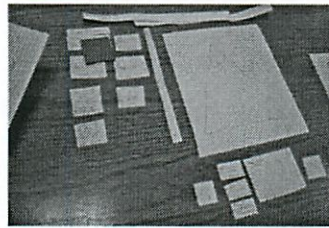
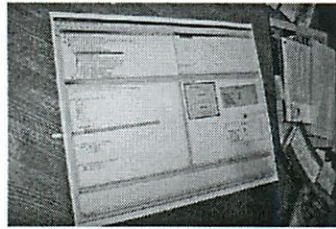
A paper prototype should be larger than life-size. Remember that fingers are bigger than a mouse pointer, and people usually write bigger than 12 point. So it'll be easier to use your paper prototype if you scale it up a bit. It will also be easier to see from a distance, which is important because the prototype lies on the table, and because when you're testing users, there may be several observers taking notes who need to see what's going on. **Big is good.**

Don't worry too much about color in your prototype. Use a single color. It's simpler, and it won't distract attention from the important issues.

You don't have to render every visual effect in paper. Some things are just easier to say aloud: "the basketball is spinning." "A progress bar pops up: 20%, 50%, 75%, done." If your design supports tooltips, you can tell your users just to point at something and ask "What's this?", and you'll tell them what the tooltip would say. If you actually want to test the tooltip messages, however, you should prototype them on paper.

Figure out a good scheme for organizing the little pieces of your prototype. One approach is a three-ring binder, with different screens on different pages. Most interfaces are not sequential, however, so a linear organization may be too simple. Two-pocket folders are good for storing big pieces, and letter envelopes (with the flap open) are quite handy for keeping menus.

## Hand-Drawn or Not?



Spring 2012

6.813/6.831 User Interface Design and Implementation

18

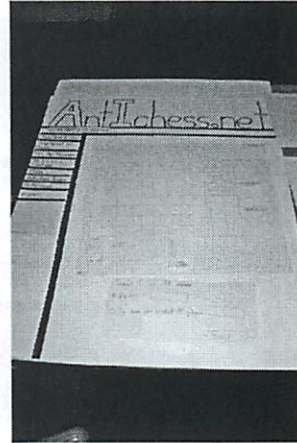
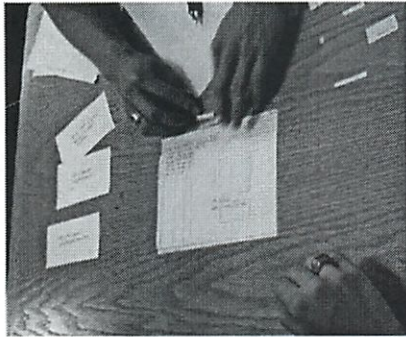
Here are some of the prototypes made by an earlier class. Should a paper prototype be hand-sketched or computer-drawn? Generally hand-sketching is better in early design, but sometimes realistic images can be constructive additions. Top left is a prototype for an interface that will be integrated into an existing program (Eclipse), so the prototype is mostly constructed of modified Eclipse screenshots. The result is very clean and crisp, but also tiny – it's hard to read from a distance. It may also be harder for a test user to focus on commenting about the new parts of the interface, since the new features look just like Eclipse. A hybrid hand-sketched/screenshot interface might work even better.

The top right prototype shows such a hybrid – a interface designed to integrate into a web browser. Actual screenshots of web pages are used, mainly as props, to make the prototype more concrete and help the user visualize the interface better. Since web page layout isn't the problem the interface is trying to solve, there's no reason to hand-sketch a web page.

The bottom photo shows a pure hand-drawn interface that might have benefited from such props -- a photo organizer could use real photographs to help the user think about what kinds of things they need to do with photographs. This prototype could also use a **window frame** – a big posterboard to serve as a static background.



## Size Matters



Spring 2012

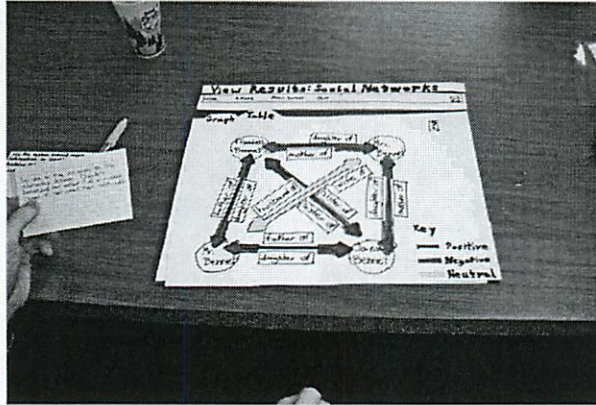
6.813/6.831 User Interface Design and Implementation

19

Both of these prototypes have good window frames, but the big one on the right is easier to read and manipulate.



## The Importance of Writing Big and Dark



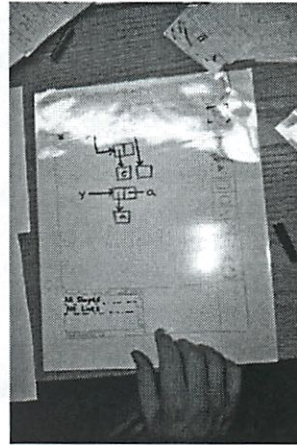
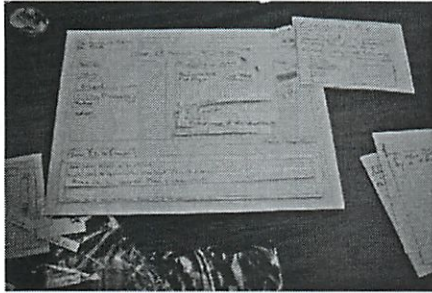
Spring 2012

6.813/6.831 User Interface Design and Implementation

20

This prototype is even easier to read. Markers are better than pencil. (Whiteout and correction tape can fix mistakes as well as erasers can!) Color is also neat, but don't bother unless color is a design decision that needs to be tested, as it is in this prototype. If color doesn't really matter, monochromatic prototypes work just as well.

## Post-it Glue and Transparencies are Good



Spring 2012

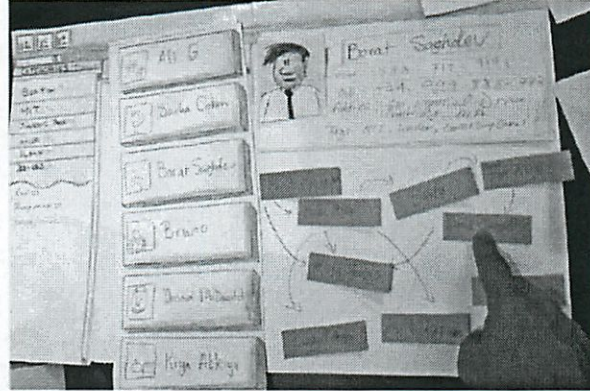
6.813/6.831 User Interface Design and Implementation

21

The prototype on the left has lots of little pieces that have trouble staying put. Post-it glue can help with that.

On the right is a prototype that's completely covered with a transparency. Users can write on it directly with dry-erase marker, which just wipes off – a much better approach than water-soluble transparency markers. With multiple layers of transparency, you can let the user write on the top layer, while you use a lower layer for computer messages, selection highlighting, and other effects.

## Paper Prototypes



Spring 2012

6.813/6.831 User Interface Design and Implementation

22

Paper is great for prototyping features that would be difficult to implement. This project (a contact manager) originally envisioned showing your social network as a graph, but when they prototyped it, it turned out that it wasn't too useful. The cost of trying that feature on paper was trivial, so it was easy to throw it away. Trying it in code, however, would have taken much longer, and been much harder to discard.

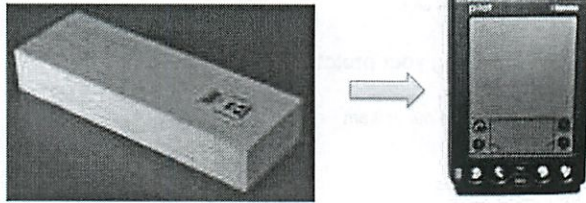
## Make a Paper Prototype

- Make a low-fi prototype of your own alarm clock
  - clock radio, analog clock, phone; whatever wakes you up every day
- Prototype just enough to support these tasks:
  - show & change alarm time
  - turn alarm on/off
- Take turns simulating your prototype on your neighbor with these tasks:
  - set the alarm to wake me at 9 am





## Low-Fidelity Prototypes Aren't Always Paper



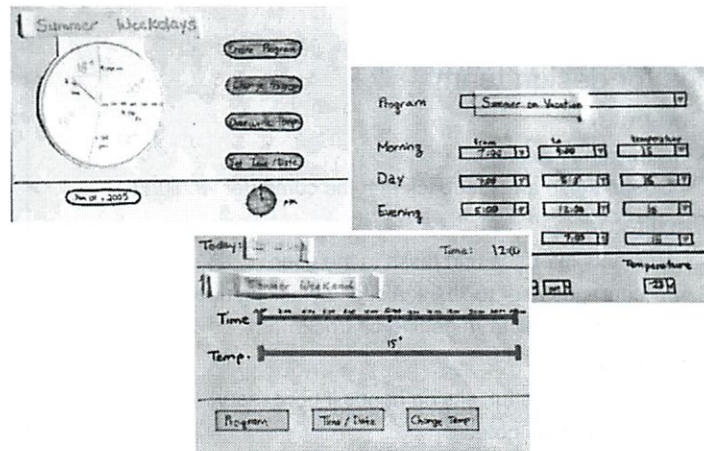
Spring 2012

6.813/6.831 User Interface Design and Implementation

24

The spirit of low-fidelity prototyping is really about using cheap physical objects to simulate software. Paper makes sense for desktop and web UIs because they're flat. But other kinds of UI prototypes might use different materials. Jeff Hawkins carried a block of wood (not this one, but similar) around in his pocket as a prototype for the first PalmPilot. <http://www.designinginteractions.com/interviews/JeffHawkins>

## Multiple Alternatives Generate Better Feedback



Spring 2012

6.813/6.831 User Interface Design and Implementation

25

Doing *several* prototypes and presenting them to the same user is a great idea. When a design is presented with others, people tend to be **more ready to criticize** and offer problems, which is exactly what you want in the early stages of design. These three paper prototypes of a house thermostat were tested against users both singly and as a group of three, and it was found that people offered fewer positive comments when they saw the designs together than when they saw them alone.

Pictures from Tohidi, Buxton, Baecker, Sellen. "Getting the Right Design and the Design Right: Testing Many Is Better Than One." *CHI 2006*.

## How to Test a Paper Prototype

- Roles for design team
  - Computer
    - Simulates prototype
    - Doesn't give any feedback that the computer wouldn't
  - Facilitator
    - Presents interface and tasks to the user
    - Encourages user to "think aloud" by asking questions
    - Keeps user test from getting off track
  - Observer
    - Keeps mouth shut, sits on hands if necessary
    - Takes copious notes

Spring 2012

6.813/6.831 User Interface Design and Implementation

26

Once you've built your prototype, you can put it in front of users and watch how they use it. We'll see much more about user testing in a later lecture, including ethical issues. But here's a quick discussion of user testing in the paper prototyping domain.

There are three roles for your design team to fill:

The **computer** is the person responsible for making the prototype come alive. This person moves around the pieces, writes down responses, and generally does everything that a real computer would do. In particular, the computer should *not* do anything that a real computer wouldn't. Think mechanically, and respond mechanically.

The **facilitator** is the human voice of the design team and the director of the testing session. The facilitator explains the purpose and process of the user study, obtains the user's informed consent, and presents the user study tasks one by one. While the user is working on a task, the facilitator tries to elicit verbal feedback from the user, particularly encouraging the user to "think aloud" by asking probing (but not leading) questions. The facilitator is responsible for keeping everybody disciplined and the user test on the right track.

Everybody else in the room (aside from the user) is an **observer**. The most important rule about being an observer is to keep your mouth shut and watch. Don't offer help to the user, even if they're missing something obvious. Bite your tongue, sit on your hands, and just watch. The observers are the primary note takers, since the computer and the facilitator are usually too busy with their duties.

### **What You Can Learn from a Paper Prototype**

- **Conceptual model**
  - Do users understand it?
- **Functionality**
  - Does it do what's needed? Missing features?
- **Navigation & task flow**
  - Can users find their way around?
  - Are information preconditions met?
- **Terminology**
  - Do users understand labels?
- **Screen contents**
  - What needs to go on the screen?

Spring 2012

6.813/6.831 User Interface Design and Implementation

27

Paper prototypes can reveal many usability problems that are important to find in early stages of design. Fixing some of these problems require large changes in design. If users don't understand the metaphor or conceptual model of the interface, for example, the entire interface may need to be scrapped.



## What You Can't Learn

- Look: color, font, whitespace, etc
- Feel: efficiency issues
- Response time
- Are small changes noticed?
  - Even the tiniest change to a paper prototype is clearly visible to user
- Exploration vs. deliberation
  - Users are more deliberate with a paper prototype; they don't explore or thrash as much

Spring 2012

6.813/6.831 User Interface Design and Implementation

28

But paper prototypes don't reveal every usability problem, because they are low-fidelity in several dimensions. Obviously, graphic design issues that depend on a high-fidelity **look** will not be discovered. Similarly, interaction issues that depend on a high-fidelity **feel** will also be missed. For example, problems like buttons that are too small, too close together, or too far away will not be detected in a paper prototype. The human computer of a paper prototype rarely reflects the speed of an implemented backend, so issues of **response time** – whether feedback appears quickly enough, or whether an entire task can be completed within a certain time constraint -- can't be tested either.

Paper prototypes don't help answer questions about whether subtle feedback will even be *noticed*. Will users notice that message down in the status bar, or the cursor change, or the highlight change? In the paper prototype, even the tiniest change is grossly visible, because a person's arm has to reach over the prototype and make the change. (If many changes happen at once, of course, then some of them may be overlooked even in a paper prototype. This is related to an interesting cognitive phenomenon called **change blindness**.)

There's an interesting qualitative distinction between the way users use paper prototypes and the way they use real interfaces. Experienced paper prototypers report that users are more deliberate with a paper prototype, apparently thinking more carefully about their actions. This may be partly due to the simulated computer's slow response; it may also be partly a social response, conscientiously trying to save the person doing the simulating from a lot of tedious and unnecessary paper shuffling. More deliberate users make fewer mistakes, which is bad, because you want to see the mistakes. Users are also less likely to randomly explore a paper prototype.

These drawbacks don't invalidate paper prototyping as a technique, but you should be aware of them. Several studies have shown that low-fidelity prototypes identify substantially the same usability problems as high-fidelity prototypes (Virzi, Sokolov, & Karis, "Usability problem identification using both low- and hi-fidelity prototypes", CHI '96; Catani & Biers, "Usability evaluation and prototype fidelity", Human Factors & Ergonomics 1998).

## Computer Prototype

- Interactive software simulation
- High-fidelity in look & feel
- Low-fidelity in depth
  - Paper prototype had a human simulating the backend; computer prototype doesn't
  - Computer prototype may be **horizontal**: covers most features, but no backend

Spring 2012

6.813/6.831 User Interface Design and Implementation

29

So at some point we have to depart from paper and move our prototypes into software. A typical computer prototype is a **horizontal** prototype. It's high-fi in look and feel, but low-fi in depth – there's no backend behind it. Where a human being simulating a paper prototype can generate new content on the fly in response to unexpected user actions, a computer prototype cannot.

## **What You Can Learn From Computer Prototypes**

- Everything you learn from a paper prototype, plus:
- Screen layout
  - Is it clear, overwhelming, distracting, complicated?
  - Can users find important elements?
- Colors, fonts, icons, other elements
  - Well-chosen?
- Interactive feedback
  - Do users notice & respond to status bar messages, cursor changes, other feedback
- Efficiency issues
  - Controls big enough? Too close together? Scrolling list is too long?

Spring 2012

6.813/6.831 User Interface Design and Implementation

30

Computer prototypes help us get a handle on the graphic design and dynamic feedback of the interface.

## Why Use Prototyping Tools?

- Faster than coding
- No debugging
- Easier to change or throw away
- Don't let your UI toolkit do your graphic design

Spring 2012

6.813/6.831 User Interface Design and Implementation

31

One way to build a computer prototype is just to program it directly in an implementation language, like Java or C++, using a user interface toolkit, like Swing or MFC. If you don't hook in a backend, or use stubs instead of your real backend, then you've got a horizontal prototype.

But it's often better to use a **prototyping tool** instead. Building an interface with a tool is usually faster than direct coding, and there's no code to debug. It's easier to change it, or even throw it away if your design turns out to be wrong. Recall Cooper's concerns about prototyping: your computer prototype may become so elaborate and precious that it *becomes* your final implementation, even though (from a software engineering point of view) it might be sloppily designed and unmaintainable.

Also, when you go directly from paper prototype to code, there's a tendency to let your UI toolkit handle all the graphic design for you. That's a mistake. For example, Java has layout managers that automatically arrange the components of an interface. Layout managers are powerful tools, but they produce horrible interfaces when casually or lazily used. A prototyping tool will help you envision your interface and get its graphic design right first, so that later when you move to code, you know what you're trying to persuade the layout manager to produce.

Even with a prototyping tool, computer prototypes can still be a tremendous amount of work. When drag & drop was being considered for Microsoft Excel, a couple of Microsoft summer interns were assigned to develop a prototype of the feature using Visual Basic. They found that they had to implement a substantial amount of basic spreadsheet functionality just to test drag & drop. It took two interns their entire summer to build the prototype that proved that drag & drop was useful. Actually adding the feature to Excel took a staff programmer only a week. This isn't a fair comparison, of course – maybe six intern-months was a cost worth paying to mitigate the risk of one fulltimer-week, and the interns certainly learned a lot. But building a computer prototype can be a slippery slope, so don't let it suck you in too deeply. Focus on what you want to test, i.e., the design risk you need to mitigate, and only prototype that.



## Computer Prototyping Techniques

- **Storyboard**
  - Sequence of painted screenshots
  - Sometimes connected by hyperlinks (“hotspots”)
- **Form builder**
  - Real windows assembled from a palette of widgets (buttons, text fields, labels, etc.)
- **Wizard of Oz**
  - Computer frontend, human backend

Spring 2012

6.813/6.831 User Interface Design and Implementation

32

There are two major techniques for building a computer prototype.

A **storyboard** is a sequence (a graph, really) of fixed screens. Each screen has one or more **hotspots** that you can click on to jump to another screen. Sometimes the transitions between screens also involve some animation in order to show a dynamic effect, like mouse-over feedback or drag-drop feedback.

A **form builder** is a tool for drawing real, working interfaces by dragging widgets from a palette and positioning them on a window.

A **Wizard of Oz** prototype is a kind of hybrid of a computer prototype and a paper prototype; the user interacts with a computer, but there's a human behind the scenes figuring out how the user interface should respond.

- Photoshop
- Balsamiq Mockup
- Mockingbird



These wireframe tools strive for some degree of “sketchiness” in their look, so these are really medium-fidelity tools. Not as low fidelity as hand sketch, but still not what the final interface will look like.

## Pros & Cons of Storyboarding

- Pros
  - You can draw anything
- Cons
  - No text entry
  - Widgets aren't active
  - "Hunt for the hotspot"

Spring 2012

6.813/6.831 User Interface Design and Implementation

34

The big advantage of storyboarding is similar to the advantage of paper: you can draw anything on a storyboard. That frees your creativity in ways that a form builder can't, with its fixed palette of widgets.

The disadvantages come from the storyboard's static nature. Some tools let you link the pictures together with hyperlinks, but even then all you can do is click, not really interact. Watching a real user in front of a storyboard often devolves into a game of "**hunt for the hotspot**", like children's software where the only point is to find things on the screen to click on and see what they do. The hunt-for-the-hotspot effect means that storyboards are largely useless for user testing, unlike paper prototypes. In general, horizontal computer prototypes are better evaluated with other techniques, like heuristic evaluation (which we'll discuss in a future lecture).

## **Form Builders**

- FlexBuilder
- Silverlight
- Visual Basic
- Mac Interface Builder
- Qt Designer
  
- Tips
  - Use absolute positioning for now

Spring 2012

6.813/6.831 User Interface Design and Implementation

35

Here are some form builder tools.



## Pros & Cons of Form Builders

- Pros
  - Actual controls, not just pictures of them
  - Can hook in some backend if you need it
    - But then you won't want to throw it away
- Cons
  - Limits thinking to standard widgets
  - Less helpful for rich graphical interfaces

Spring 2012

6.813/6.831 User Interface Design and Implementation

36

Unlike storyboards, form builders use actual working widgets, not just static pictures. So the widgets look the same as they will in the final implementation (assuming you're using a compatible form builder – a prototype in Visual Basic may not look like a final implementation in Java).

Also, since form builders usually have an implementation language underneath them – which may even be the same implementation language that you'll eventually use for your final interface -- you can also hook in as much or as little backend as you want.

On the down side, form builders give you a fixed palette of standard widgets, which limits your creativity as a designer, and which makes form builders far less useful for prototyping rich graphical interfaces, e.g., a circuit-drawing editor. Form builders are great for the menus and widgets that surround a graphical interface, but can't simulate the "insides" of the application window.

## Wizard of Oz Prototype

- Software simulation with a human in the loop to help
- “Wizard of Oz” = “man behind the curtain”
  - Wizard is usually but not always hidden
- Often used to simulate future technology
  - Speech recognition
  - Learning
- Issues
  - Two UIs to worry about: user’s and wizard’s
  - Wizard has to be mechanical

Spring 2012

6.813/6.831 User Interface Design and Implementation

37

Part of the power of paper prototypes is the depth you can achieve by having a human simulate the backend. A **Wizard of Oz prototype** also uses a human in the backend, but the frontend is an actual computer system instead of a paper mockup. The term Wizard of Oz comes from the movie of the same name, in which the wizard was a man hiding behind a curtain, controlling a massive and impressive display.

In a Wizard of Oz prototype, the “wizard” is usually but not always hidden from the user. Wizard of Oz prototypes are often used to simulate future technology that isn’t available yet, particularly artificial intelligence. A famous example was the listening typewriter (Gould, Conti, & Hovanyecz, “Composing letters with a simulated listening typewriter,” *CACM* v26 n4, April 1983). This study sought to compare the effectiveness and acceptability of isolated-word speech recognition, which was the state of the art in the early 80’s, with continuous speech recognition, which wasn’t possible yet. The interface was a speech-operated text editor. Users looked at a screen and dictated into a microphone, which was connected to a typist (the wizard) in another room. Using a keyboard, the wizard operated the editor showing on the user’s screen.

The wizard’s skill was critical in this experiment. She could type 80 wpm, she practiced with the simulation for several weeks (with some iterative design on the simulator to improve her interface), and she was careful to type *exactly* what the user said, even exclamations and parenthetical comments or asides. The computer helped make her responses a more accurate simulation of computer speech recognition. It looked up every word she typed in a fixed dictionary, and any words that were not present were replaced with X’s, to simulate misrecognition. Furthermore, in order to simulate the computer’s ignorance of context, homophones were replaced with the most common spelling, so “done” replaced “dun”, and “in” replaced “inn”. The result was an extremely effective illusion. Most users were surprised when told (midway through the experiment) that a human was listening to them and doing the typing.

Thinking and acting mechanically is harder for a wizard than it is for a paper prototype simulator, because the tasks for which Wizard of Oz testing is used tend to be more “intelligent”. It helps if the wizard is personally familiar with the capabilities of similar interfaces, so that a realistic simulation can be provided. (See Maulsby et al, “Prototyping an intelligent agent through Wizard of Oz”, CHI 1993.) It also helps if the wizard’s interface can intentionally dumb down the responses, as was done in the Gould study.

A key challenge in designing a Wizard of Oz prototype is that you actually have two interfaces to worry about: the user’s interface, which is presumably the one you’re testing, and the wizard’s.

## Summary

- Prototype fidelity
  - Depth, breadth, look, feel
- Kinds of prototypes
  - Paper
  - Computer: storyboard, forms
  - Wizard of Oz
- Don't get attached to a prototype
  - Because it may need to be thrown away

(5 min late)

User testing

UI Hall of Fame / shame Greeting card

Auto count down char remaining

Weird ordering - System ~~WIN~~ IN

Underscores + missing hyp

Likely order → upcoming

no explanation

could show the dates

Quizlearning - not scenarios - you tell them what to do  
computer tests workpaper prototype - users have more time to thinkWizard-of-Oz - front end is a computer  
↳ so looks right  
back end is a human  
high fidelity in depth



②

Today: User testing  
Ethics  
Formative evaluation

---

Formative eval - to find problems  
Want to discover + fix usability problems

Field study

Controlled exp → ask them to do tasks in your lab

---

Ethics

human subjects abused in the past  
(how does this relate to UI?)

- non informed consent
- deception
  - is actually allowed
  - but raises ethical level of scrutiny
  - worth the cost putting on experimenters

③

## Basic Principle (Belmont Report)

Respect for person

- ~~vol~~
- informed consent
- protection of vulnerable participants

Beneficence

- do no harm
- risk vs benefit

Justice

- fair selection of study
  - ~~not~~ mostly medical
  - that available to all

IRB

required at all federally funded  
producing generalizable knowledge

- not on projects

~~MIT~~ MIT's is called COHUS

⑨

## Pressures on a User

- Performance anxiety
- feels like intel test
- comparing self w/ other subject
- ~~the~~ might feel stupid
- Competing w/ other subjects

1. Don't grab someone randomly
2. Didn't say what ~~an~~ study about  
La new UI
3. ~~also~~ Pick someone w/ right domain
4. Don't have boss be observer
5. Pilot test stuff before
6. Don't laugh at user

5

## 5 Big Tenets

Time

- don't waste
- try it out before

Comfort

- allow them to go to BP

Informed Consent

Privacy

- not their boss in room

Control

- can stop at any time

Some things you can do on the slides

- blame the system, not the user
- tell them they can leave
- don't act too emotional

After

- debrief - what you learned
- answer their qu



(6)

Reviel the deceptions

Ask permission to use video/audio

---

## Formative Eval

1. Find users
  - representative sample
2. Give tasks
  - but can settle w/ convenience users
3. Watch the users do the task

## Roles

User

Facilitator

Observer

## User's role

- Should think aloud
  - but this might alter behavior
- or pair of users
- or retrospective
  - review video
  - ask them to narrate play by play

⑦

### Observer

Needs to shut up - no help!

Or be in glass 1-way mirror room  
take notes - including successes

Do put a facilitator in the room

### Recording

Keeping notes + most effective

Video - need to rewatch

## L12: User Testing

- GR2 due Sun
- GR3 out Mon, due next Sun
- PS2/RS2 out Mon, due at end of spring break
- Next week:
  - lectures for 6.831 G (optional for 6.813 U)
  - M 3-5 pm paper prototype building (Walker)
  - WF 3-5 pm prototype testing & RS2 testing (Walker)

Spring 2012

6.813/6.831 User Interface Design and Implementation

1

## UI Hall of Fame or Shame?

**Enter a Card Message**

There is a 210 character limit for your message due to the size of the card. Note: please do not use any special symbols like "&".

At a loss for words? We can help you express your sentiments. [Click here to view](#)

Greeting Type: Click to select..

To:

Message:

From:

210 character maximum

**Click to select..**

- Yom\_Kippur
- Sweetest\_Day
- Rosh\_Hashana
- Mother\_In\_Laws\_Day
- Labor\_Day
- Grandparents\_Day
- Bosss\_Day
- Thank You
- Others
- Anniversary
- Maternity
- Holiday
- Business Gifts
- Birthday
- Illness/Get\_Well

Suggested by Ryan Damico

Spring 2010

6.813/6.831 User Interface Design and Implementation

2

Today's candidate for the Hall of Shame is this entry form from the 1800Flowers web site. The purpose of the form is to enter a message for a greeting card that will accompany a delivered flower arrangement. (So you can see the whole interface, I've moved the Greeting Type drop-down menu to the right. In the real interface, it appears where you'd expect, right under the Greeting Type drop-down box.)

The 210 character limit is probably necessary for backend reasons (e.g. size of the card delivered with the flowers), but hard for a user to check. Suggest a dynamic progress bar showing how much of the quota you've used. (error prevention, flexibility & efficiency)

Special symbols like & is vague. What about asterisk and hyphen – are those special too? What am I allowed to use, exactly? (user control & freedom)

The underscores in the Greeting Type drop-down menu look like technical identifiers, and some even look misspelled because they've omitted other punctuation. Bosss\_Day? (match the real world)

How does Greeting Type actually affect the card? (visibility) This is related to a Hall of Fame we had a little while ago: the Domino's pizza ordering site does a much better job of showing you how your choices actually affect the final product.



## Today's Topics

- User testing
- Ethics
- Formative evaluation

Spring 2010

6.813/6.831 User Interface Design and Implementation

5

In this lecture, we'll talk about **user testing**: putting an interface in front of real users. There are several kinds of user testing, but all of them by definition involve human beings, who are thinking, breathing individuals with rights and feelings. When we enlist the assistance of real people in interface testing, we take on some special responsibilities. So first we'll talk about the **ethics** of user testing, which apply regardless of what kind of user test you're doing.

The rest of the lecture will focus on one particular kind of user test: **formative evaluation**, which is a user test performed during iterative design with the goal of finding usability problems to fix on the next design iteration.

## Kinds of User Tests

- Formative evaluation
  - Find problems for next iteration of design
  - Evaluates prototype or implementation, in lab, on chosen tasks
  - Qualitative observations (usability problems)
- Field study
  - Find problems in context
  - Evaluates working implementation, in real context, on real tasks
  - Mostly qualitative observations
- Controlled experiment
  - Tests a hypothesis (e.g., interface X is faster than interface Y)
  - Evaluates working implementation, in controlled lab environment, on chosen tasks
  - Mostly quantitative observations (time, error rate, satisfaction)

Spring 2010

6.813/6.831 User Interface Design and Implementation

6

Here are three common kinds of user tests.

You'll be doing **formative evaluations** with the prototypes you build in this class. The purpose of formative evaluation is finding usability problems in order to fix them in the next design iteration. Formative evaluation doesn't need a full working implementation, but can be done on a variety of prototypes. This kind of user test is usually done in an environment that's under your control, like an office or a usability lab. You also choose the tasks given to users, which are generally realistic (drawn from task analysis, which is based on observation) but nevertheless fake. The results of formative evaluation are largely **qualitative observations**, usually a list of usability problems.

A key problem with formative evaluation is that you have to control too much. Running a test in a lab environment on tasks of your invention may not tell you enough about how well your interface will work in a real context on real tasks. A **field study** can answer these questions, by actually deploying a working implementation to real users, and then going out to the users' real environment and observing how they use it. We won't say much about field studies in this class.

A third kind of user test is a **controlled experiment**, whose goal is to test a quantifiable hypothesis about one or more interfaces. Controlled experiments happen under carefully controlled conditions using carefully-designed tasks – often more carefully chosen than formative evaluation tasks. Hypotheses can only be tested by quantitative measurements of usability, like time elapsed, number of errors, or subjective ratings. We'll talk about controlled experiments in a future lecture.

## Ethics of User Testing

- **Users are human beings**
  - Human subjects have been seriously abused in the past
    - Nazi concentration camps
    - Tuskegee syphilis study
    - MIT Fernald School study
    - Yale electric shock study
  - Research involving user testing is now subject to close scrutiny
    - MIT Committee on Use of Humans as Experimental Subjects (COUHES) must approve research-related user studies

Spring 2010

6.813/6.831 User Interface Design and Implementation

7

Let's start by talking about some issues that are relevant to all kinds of user testing: ethics. Human subjects have been horribly abused in the name of science over the past century. Here are some of the most egregious cases:

In Nazi concentration camps (1940-1945), doctors used prisoners of war, political prisoners, and Jews as human guinea pigs for horrific experiments. Some experiments tested the limits of human endurance in extreme cold, low pressures, or exposure. Other experiments intentionally infected people with massive doses of pathogens, such as typhus; others tested new chemical weapons or new medical procedures. Thousands of people were killed by these experiments; they were criminal, on a massive scale.

In the Tuskegee Institute syphilis study (1932-1972), the US government studied the effects of untreated syphilis in black men in the rural South. In exchange for their participation in the study, the men were given free health examinations. But they weren't told that they had syphilis, or that the disease was potentially fatal. Nor were they given treatment for the disease, even as proven, effective treatments like penicillin became available. Out of 339 men studied, 28 died directly of syphilis, 100 of related complications. 40 wives were infected, and 19 children were born with congenital syphilis.

In the 1940s and 1950s, MIT researchers cooperated with the Fernald School for mentally disabled children in Waverly, Massachusetts to give radioactive isotopes to some of the children in their milk and cereal, to study how the isotopes were taken up by the body. Permission letters were obtained from their parents, but neither parents nor children were warned that radioactive materials were being used.

In the 1950s, a famous study done at Yale told subjects to give painful electric shocks to another person. The shocks weren't real, and the person they were shocking was just an actor. But subjects weren't told that fact in advance, and many subjects were genuinely traumatized by the experience: sweating, trembling, stuttering.

These cases have led to several reforms. The Nazi-era experiments led to the Nuremberg Code, an international agreement on the rights of human subjects. The Tuskegee study drove the US government to take steps to ensure that all federally-funded institutions follow ethical practices in their use of human subjects. In particular, every experiment involving human subjects must be reviewed and approved by an ethics committee, usually called an institutional review board. MIT's review board is called COUHES.



## Pressures on a User

- Performance anxiety
- Feels like an intelligence test
- Comparing self with other subjects
- Feeling stupid in front of observers
- Competing with other subjects

Spring 2010

6.813/6.831 User Interface Design and Implementation

8

Experiments involving medical treatments or electric shocks are one thing. But what's so dangerous about a computer interface?

Hopefully, nothing – most user testing has minimal physical or psychological risk to the user. But user testing does put psychological pressure on the user. The user sits in the spotlight, asked to perform unfamiliar tasks on an unfamiliar (and possibly bad!) interface, in front of an audience of strangers (at least one experimenter, possibly a roomful of observers, and possibly a video camera). It's natural to feel some performance anxiety, or stage fright. "Am I doing it right? Do these people think I'm dumb for not getting it?" A user may regard the test as a psychology test, or more to the point, an IQ test. They may be worried about getting a bad score. Their self-esteem may suffer, particularly if they blame problems they have on themselves, rather than on the user interface.

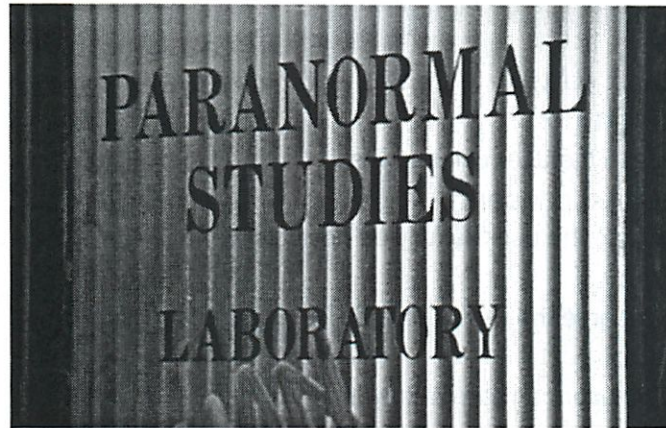
A programmer with an ironclad ego may scoff at such concerns, but these pressures are real. Jared Spool, a usability consultant, tells a story about the time he saw a user cry during a user test. It came about from an accumulation of mistakes on the part of the experimenters:

1. the originally-scheduled user didn't show up, so they just pulled an employee out of the hallway to do the test;
2. it happened to be her first day on the job;
3. they didn't tell her what the session was about;
4. she not only knew nothing about the interface to be tested (which is fine and good), but also nothing about the domain – she wasn't in the target user population at all;
5. the observers in the room hadn't been told how to behave (i.e., shut up);
6. one of those observers was her boss;
7. the tasks hadn't been pilot tested, and the first one was actually impossible.

When she started struggling with the first task, everybody in the room realized how stupid the task was, and burst out laughing – at their own stupidity, not hers. But she thought they were laughing at her, and she burst into tears. (story from Carolyn Snyder, Paper Prototyping)



## **A Case Study of Ethics in User Studies**



Spring 2010

6.813/6.831 User Interface Design and Implementation

9

See Venkman, P. "The Effect of Negative Reinforcement on ESP Ability." Unpublished monograph, 1984.

## Treat the User With Respect

- Time
  - Don't waste it
- Comfort
  - Make the user comfortable
- Informed consent
  - Inform the user as fully as possible
- Privacy
  - Preserve the user's privacy
- Control
  - The user can stop at any time

Spring 2010

6.813/6.831 User Interface Design and Implementation

10

The basic rule for user testing ethics is **respect** for the user as a intelligent person with free will and feelings.

We can show respect for the user in 5 ways:

1. Respecting their **time** by not wasting it. Prepare as much as you can in advance, and don't make the user jump through hoops that you aren't actually testing. Don't make them install the software or load the test files, for example, unless your test is supposed to measure the usability of the installation process or file-loading process.
2. Do everything you can to make the user **comfortable**, in order to offset the psychological pressures of a user test.
3. Give the user as much **information** about the test as they need or want to know, as long as the information doesn't bias the test. Don't hide things from them unnecessarily.
4. Preserve the user's **privacy** to the maximum degree. Don't report their performance on the user test in a way that allows the user to be personally identified.
5. The user is always in **control**, not in the sense that they're running the user test and deciding what to do next, but in the sense that the final decision of whether or not to participate remains theirs, throughout the experiment. Just because they've signed a consent form, or sat down in the room with you, doesn't mean that they've committed to the entire test. A user has the right to give up the test and leave at any time, no matter how inconvenient it may be for you.

## Before a Test

- Time
  - Pilot-test all materials and tasks
- Comfort
  - “We’re testing the system; we’re not testing you.”
  - “Any difficulties you encounter are the system’s fault. We need your help to find these problems.”
- Privacy
  - “Your test results will be completely confidential.”
- Information
  - Brief about purpose of study
  - Inform about audiotaping, videotaping, other observers
  - Answer any questions beforehand (unless biasing)
- Control
  - “You can stop at any time.”

Spring 2010

6.813/6.831 User Interface Design and Implementation

11

Let’s look at what you should do before, during, and after a user test to ensure that you’re treating users with respect.

Long before your first user shows up, you should **pilot-test** your entire test: all questionnaires, briefings, tutorials, and tasks. Pilot testing means you get a few people (usually your colleagues) to act as users in a full-dress rehearsal of the user test. Pilot testing is essential for simplifying and working the bugs out of your test materials and procedures. It gives you a chance to eliminate wasted time, streamline parts of the test, fix confusing briefings or training materials, and discover impossible or pointless tasks. It also gives you a chance to practice your role as an experimenter. Pilot testing is essential for every user test.

When a user shows up, you should brief them first, introducing the purpose of the application and the purpose of the test. To make the user comfortable, you should also say the following things (in some form):

• “Keep in mind that we’re testing the computer system. We’re not testing you.” (comfort)

• “The system is likely to have problems in it that make it hard to use. We need your help to find those problems.” (comfort)

• “Your test results will be completely confidential.” (privacy)

• “You can stop the test and leave at any time.” (control)

You should also inform the user if the test will be audiotaped, videotaped, or watched by hidden observers. Any observers actually present in the room should be introduced to the user.

At the end of the briefing, you should ask “Do you have any questions I can answer before we begin?” Try to answer any questions the user has. Sometimes a user will ask a question that may bias the experiment: for example, “what does that button do?” You should explain why you can’t answer that question, and promise to answer it after the test is over.

## During the Test

- Time
  - Eliminate unnecessary tasks
- Comfort
  - Calm, relaxed atmosphere
  - Take breaks in long session
  - Never act disappointed
  - Give tasks one at a time
  - First task should be easy, for an early success experience
- Privacy
  - User's boss shouldn't be watching
- Information
  - Answer questions (again, where they won't bias)
- Control
  - User can give up a task and go on to the next
  - User can quit entirely

Spring 2010

6.813/6.831 User Interface Design and Implementation

12

During the test, arrange the testing environment to make the user comfortable. Keep the atmosphere calm, relaxed, and free of distractions. If the testing session is long, give the user bathroom, water, or coffee breaks, or just a chance to stand up and stretch.

Don't act disappointed when the user runs into difficulty, because the user will feel it as disappointment in their performance, not in the user interface.

Don't overwhelm the user with work. Give them only one task at a time. Ideally, the first task should be an easy warmup task, to give the user an early success experience. That will bolster their courage (and yours) to get them through the harder tasks that will discover more usability problems.

Answer the user's questions as long as they don't bias the test.

Keep the user in control. If they get tired of a task, let them give up on it and go on to another. If they want to quit the test, pay them and let them go.



## After the Test

- Comfort
  - Say what they've helped you do
- Information
  - Answer questions that you had to defer to avoid biasing the experiment
- Privacy
  - Don't publish user-identifying information
  - Don't show video or audio without user's permission

Spring 2010

6.813/6.831 User Interface Design and Implementation

13

After the test is over, thank the user for their help and tell them how they've helped. It's easy to be open with information at this point, so do so.

Later, if you disseminate data from the user test, don't publish it in a way that allows users to be individually identified. Certainly, avoid using their names.

If you collected video or audio records of the user test, don't show them outside your development group without explicit written permission from the user.

## Formative Evaluation

- Find some users
  - Should be representative of the target user class(es), based on user analysis
- Give each user some tasks
  - Should be representative of important tasks, based on task analysis
- Watch user do the tasks

Spring 2010

6.813/6.831 User Interface Design and Implementation

14

OK, we've seen some ethical rules that apply to running any kind of user test. Now let's look in particular at how to do **formative evaluation**.

Here are the basic steps: (1) find some representative users; (2) give each user some representative tasks; and (3) watch the user do the tasks.

## **Roles in Formative Evaluation**

- User
- Facilitator
- Observers

Spring 2010

6.813/6.831 User Interface Design and Implementation

15

There are three roles in a formative evaluation test: a user, a facilitator, and some observers.

## User's Role

- User should think aloud
  - What they think is happening
  - What they're trying to do
  - Why they took an action
- Problems
  - Feels weird
  - Thinking aloud may alter behavior
  - Disrupts concentration
- Another approach: pairs of users
  - Two users working together are more likely to converse naturally
  - Also called co-discovery, constructive interaction

Spring 2010

6.813/6.831 User Interface Design and Implementation

16

The user's primary role is to perform the tasks using the interface. While the user is actually doing this, however, they should also be trying to **think aloud**: verbalizing what they're thinking as they use the interface. Encourage the user to say things like "OK, now I'm looking for the place to set the font size, usually it's on the toolbar, nope, hmm, maybe the Format menu..." Thinking aloud gives you (the observer) a window into their thought processes, so you can understand what they're trying to do and what they expect.

Unfortunately, thinking aloud feels strange for most people. It can alter the user's behavior, making the user more deliberate and careful, and sometimes disrupting their concentration. Conversely, when a task gets hard and the user gets absorbed in it, they may go mute, forgetting to think aloud. One of the facilitator's roles is to prod the user into thinking aloud.

One solution to the problems of think-aloud is **constructive interaction**, in which two users work on the tasks together (using a single computer). Two users are more likely to converse naturally with each other, explaining how they think it works and what they're thinking about trying. Constructive interaction requires twice as many users, however, and may be adversely affected by social dynamics (e.g., a pushy user who hogs the keyboard). But it's nearly as commonly used in industry as single-user testing.



## Facilitator's Role

- Does the briefing
- Provides the tasks
- Coaches the user to think aloud by asking questions
  - “What are you thinking?”
  - “Why did you try that?”
- Controls the session and prevents interruptions by observers

Spring 2010

6.813/6.831 User Interface Design and Implementation

17

The facilitator (also called the experimenter) is the leader of the user test. The facilitator does the briefing, gives tasks to the user, and generally serves as the voice of the development team throughout the test. (Other developers may be observing the test, but should generally keep their mouths shut.)

One of the facilitator's key jobs is to coax the user to think aloud, usually by asking general questions.

The facilitator may also move the session along. If the user is totally stuck on a task, the facilitator may progressively provide more help, e.g. “Do you see anything that might help you?”, and then “What do you think that button does?” Only do this if you've already recorded the usability problem, and it seems unlikely that the user will get out of the tar pit themselves, and they need to get unstuck in order to get on to another part of the task that you want to test. Keep in mind that once you explain something, you lose the chance to find out what the user would have done by themselves.

## Observer's Role

- Be quiet!
  - Don't help, don't explain, don't point out mistakes
  - Sit on your hands if it helps
- Take notes
  - Watch for critical incidents: events that strongly affect task performance or satisfaction
  - Usually negative
    - Errors
    - Repeated attempts
    - Curses
  - May be positive
    - "Cool!"
    - "Oh, now I see."

Spring 2010

6.813/6.831 User Interface Design and Implementation

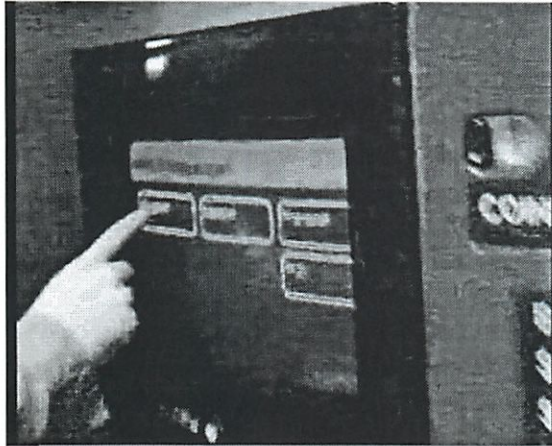
18

While the user is thinking aloud, and the facilitator is coaching the think-aloud, any observers in the room should be doing the opposite: **keeping quiet**. Don't offer any help, don't attempt to explain the interface. Just sit on your hands, bite your tongue, and watch. You're trying to get a glimpse of how a typical user will interact with the interface. Since a typical user won't have the system's designer sitting next to them, you have to minimize your effect on the situation. It may be very hard for you to sit and watch someone struggle with a task, when the solution seems so *obvious* to you, but that's how you learn the usability problems in your interface.

Keep yourself busy by taking a lot of notes. What should you take notes about? As much as you can, but focus particularly on **critical incidents**, which are moments that strongly affect usability, either in task performance (efficiency or error rate) or in the user's satisfaction. Most critical incidents are negative. Pressing the wrong button is a critical incident. So is repeatedly trying the same feature to accomplish a task. Users may draw attention to the critical incidents with their think-aloud, with comments like "why did it do that?" or "@%!@#\$!" Critical incidents can also be positive, of course. You should note down these pleasant surprises too.

Critical incidents give you a list of potential usability problems that you should focus on in the next round of iterative design.

### Example: Think Aloud



Spring 2010

6.813/6.831 User Interface Design and Implementation

19

Let's practice observing a user test, listening to think-aloud, and watching for critical incidents. This isn't really a user test – it's even better, it's a user interacting naturally in the wild! Watch this video of somebody using a NYC subway fare machine:

<http://www.youtube.com/watch?v=mfCQbZR-nhk>

Why is the user thinking aloud? Did you note any critical incidents?

### Example: Watching for Critical Incidents



Spring 2010

6.813/6.831 User Interface Design and Implementation

20

Now here's one from the DC Metro:

<http://www.youtube.com/watch?v=7TOsJCA7DHw>

Note the critical incidents with their timepoints, and we'll talk about them.



## Recording Observations

- Pen & paper notes
  - Prepared forms can help
- Audio recording
  - For think-aloud
- Video recording
  - Usability labs often set up with two cameras, one for user's face, one for screen
  - User may be self-conscious
  - Good for closed-circuit view by observers in another room
  - Generates too much data
  - Retrospective testing: go back through the video with the user, discussing critical incidents
- Screen capture & event logging
  - Cheap and unobtrusive
  - Camtasia, CamStudio

Spring 2010

6.813/6.831 User Interface Design and Implementation

21

Here are various ways you can record observations from a user test. Paper notes are usually best, although it may be hard to keep up. Having multiple observers taking notes helps.

Audio and video recording are good for capturing the user's think-aloud, facial expressions, and body language. Video is also helpful when you want to put observers in a separate room, watching on a closed-circuit TV. Putting the observers in a separate room has some advantages: the user feels fewer eyes on them (although the video camera is another eye that can make users more self-conscious, since it's making a permanent record), the observers can't misbehave, and a big TV screen means more observers can watch. On the other hand, when the observers are in a separate room, they may not pay close attention to the test. It's happened that as soon as the user finds a usability problem, the observers start talking about how to fix that problem – and ignore the rest of the test. Having observers in the same room as the test forces them to keep quiet and pay attention.

Video is also useful for **retrospective testing** – using the videotape to debrief the user immediately after a test. It's easy to fast forward through the tape, stop at critical incidents, and ask the user what they were thinking, to make up for gaps in think-aloud.

The problem with audio and video tape is that it generates too much data to review afterwards. A few pages of notes are much easier to scan and derive usability problems.

Screen capture software offers a cheap and easy way to record a user test, producing a digital movie (e.g. AVI or MPG). It's less obtrusive and easier to set up than a video camera, and some packages can also record an audio stream to capture the user's think-aloud. The course wiki has a page with recommendations for screen capture software.

## Summary

- Formative user testing tries to uncover usability problems to fix in next iteration
- Treat users with respect
- Facilitator and observers should play their roles correctly to maximize the value of the test

## 6.813/6.831 • USER INTERFACE DESIGN AND IMPLEMENTATION

Spring 2012 Massachusetts Institute of Technology  
Department of Electrical Engineering and Computer Science

### GR2: DESIGNS

Due at 11:59 pm on Sunday, March 11, 2012, by updating your group's wiki page.

In this group assignment, you will start designing your term project.

#### Designs

**Scenario.** Write a scenario that involves all three of the tasks you identified in GR1. Where your task descriptions in task analysis were abstract, your scenario should be concrete, complete with imaginary users' names and imaginary details.

**Storyboard designs.** Generate **three different** preliminary designs for your user interface. Explain each design and include a storyboard showing how it works for your scenario. The storyboard should combine words with sketches showing how the interface would look over the course of the scenario. After the storyboard, you should have an analysis that considers the design's good and bad points for learnability, efficiency, and safety.

Your designs will be judged on how well you've described them, how well you analyze them, and how diverse they are. Three designs that differ only in small ways will not receive much credit.

Take time to brainstorm a variety of different interface designs, sketching them by hand on paper or a whiteboard. You should play with many more than three designs, but we only require you to record three on your wiki page.

When you draw your sketches, don't get bogged down in details like wording, graphical appearance, or layout. Keep things simple. Focus on the conceptual model you're trying to communicate to the user, and think about your task analysis: what the user needs to do and how they can do it. Putting too much time into designing low-level details is pointless if big things have to change on the next design iteration.

**Hand-drawn sketches are preferred.** There are a number of ways to get hand-drawn sketches into your wiki page. You can draw on paper and use a scanner to convert it to electronic form. There are scanners located around campus; the Scanners page on the wiki lists the ones we know about; add more if you know of any. Make sure your sketches are readable, and crop them and size them appropriately so that your wiki presentation has good usability.

#### What to Hand In

Update your group's wiki page so that it contains a section **GR2 Designs**, with subsections for:

- **Scenario**
- **Designs**

Put your GR2 as a new page on the wiki, with a link to it from your main project page.

We will grade the version that exists at the moment of the deadline.





# MIT Scenarios

Added by [Michael Plasmeier](#), last edited by [Michael Plasmeier](#) on Mar 10, 2012 13:58

## Poster Creator: Create a Poster

Alice is an administrative assistant in the EECS department at MIT. Alice is in charge of running department events, including the distinguished lecture series. Alice is currently designing a poster for a talk by noted UI researcher, Bob, in 2 weeks. Alice wants people to see the poster and attend the event. In addition, Alice would like to know how many people are planning to attend, in order to order food for the event. Currently estimating is ~~an~~ inexact science because Alice does not know how popular the event will be.

Alice is designing the poster for the event in the program she always uses for designing posters, Adobe Publisher. Alice just heard about a new service, RScanVP. RScanVP allows Alice to put a QR code on her posters to make it easy for people to add the event to her calendar and give Alice statistics on how many people added the event to their calendars. Alice goes to RScanVP.com. She then clicks on "Create a QR code." She enters the title of the event, the location, as well as the start and end times of the event. Alice then hits save, and a QR code is generated. Alice can change the color of the QR code to match the design of her poster. Alice then downloads the QR code as a PNG file and imports it into Adobe Publisher. Alice also receives an email with a link to edit the QR code in the future, as well as a website where she or a colleague can see how many people RSVPed to the event. Alice then prints 100 copies of the poster and posts it around MIT.

## Poster Viewer: Add event to calendar

Charlie is a student in EECS at MIT. He is walking down the hall when a poster catches his eye. It's an advertisement for a talk given by a UI Researcher named Bob next week. Charlie is intrigued; he is interested in Bob's work. Charlie whips out his Android to add the event to his calendar. Charlie notices the QR code on the poster has the label "Add this event to your calendar." Charlie was going to enter all the details manually, but he thinks the QR code might be faster. Charlie launches his Barcode Scanner app and takes a picture of the QR code. Charlie is then brought to a mobile page on RScanVP.com. He clicks a button to add the event to his calendar. The link opens up his phone's Calendar app and adds the event to his calendar.

## View RSVPs

A day before the event, Alice needs to place an order for food. She goes back in her email and finds the link from RScanVP.com. She opens the link and sees that 50 people have added the event to her calendar. She knows that not everyone used RScanVP and that some people who added the event to their calendars might not attend, so Alice decides to order food for 75 people. Alice is happy because she previously had no clue how popular the event would be.

### Get Help

- [User Guide](#)
- [Training](#)
- [Confluence Help](#)
- [Contact the Help Desk](#)
- [Knowledge Base](#)
- [Request a Wiki Space](#)

### Resources

- [Terms of Service](#)
- [Stellar](#)
- [Supported Browsers](#)
- [WebSIS](#)
- [MIT Touchstone](#)

## Review

3/10

Matt

- why type of event
- location of scans
- poster pictures
- email reminder
- RSVP
- login

EJ

QR code for A

OR	View events
----	-------------

Add img to QR code

Colors

Size matters

Live preview

Graphic Design > Event Details

(2)

Scrolling widget to view events

Remind user of picture of QR code

QR code per location

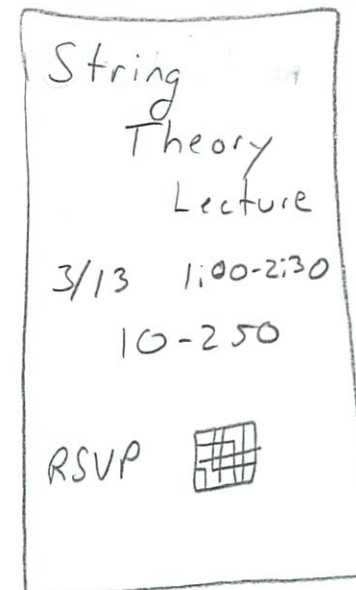
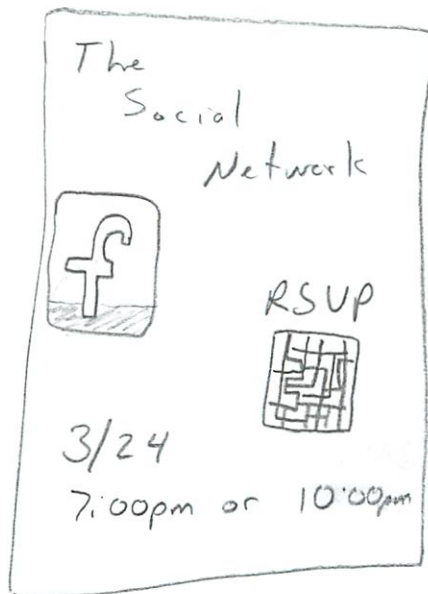
create account login

# RScanVP

Make your posters clickable  
with a scannable QR code.

Make a  
QR Code

Examples (Hover = blow up image)






# Make a QR Code

Step 1: Choose an event... ▾

Lecture
Food Event
Movie
Trip
Other

Step 2: RSVP Required\* Yes ☐ No ☐

Date  

Place

Time  ▾

Step 3: Add to calendar ☐

Select one Email user reminder ☐

Create phone reminder ☐

Step 4:

Create QR Code

Hi, user!

View RSUP's (click an event)

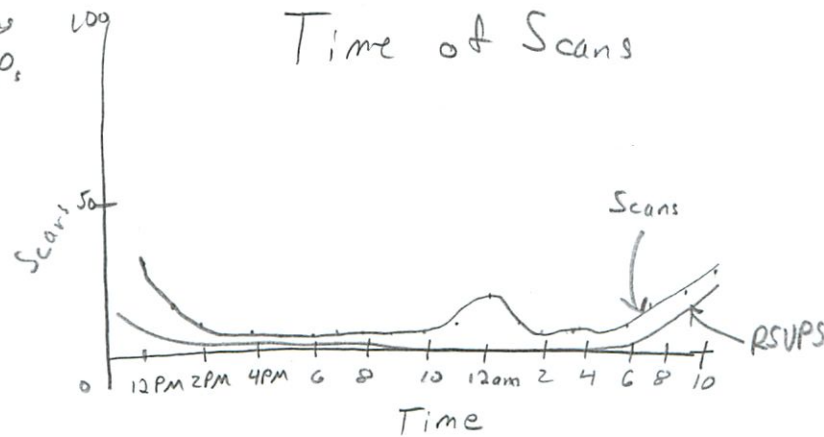
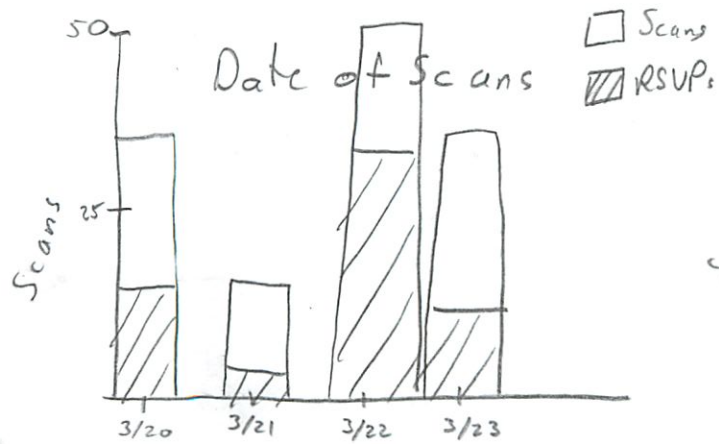
1 The Social Network - Movie
2 2012 BBQ - Food Event
3 String Theory Lecture - Lecture
Create New QR Code

Clicking an event updates the info below.

Event: The Social Network - Movie

Number of Scans: 129  
click for list of emails

Number of RSUPs: 83  
click for list of emails



Location of Scans

Most Popular:

- 1) Infinite Hallway
- 2) Lobby 7
- 3) Lobby 16
- 4) Student Center
- 5) Building 3P

# RScanVP

- ~~Make it easy~~ <sup>for your attendees</sup> to add ~~your~~ <sup>your own</sup> events to their calendars
- Gives you a count on attendees

Homepage  
text

No login

Homepage

# R Scan VP

Text

Text

66%

Make a QR code

33%

Screenshots

3 screenshots

create

Photos

Add

View ASP

Help

About

Contact



R5canVP

Create an Event

Step 1: Enter Details

Title

Location

Start

End

Next →

# RScanVP: Create an Event

Create

## Step 2 Design

You can always change this later

Color select

R

G

B

Size

PM

Live Preview

Download

PNG

JPG

Save Event

Done

< Back

Bob's UI Lecture  
Mon 4/11 10AM-11AM

RScanVP logo in middle

More to next pg

RScanVP: Create an Event

Event Created!

A link has been emailed to you

View Link

~~My RScan VP~~

FB

Twitter

Admin Link

- change details, View RScanVP

~~RS~~

Create Another

Add download here

- We could have the link  
for desktop / email  
Out of Scope for 6.813

Mobile

Ream VP

Bob's VI Talk

9-190

3/10 10AM-11AM

Add to Calendar

All options

Google Calendar

Email Reminders

← ~~Work~~  
ICS  
method

Add to Calendar

RSP ☒ Yes ☐ No

Other ways

Email ☐

to Cal ☐

← add this actually  
iPhone style switcher  
default yes



RScanVP: View RSVs; Bob's lecture

337 RSVs  
430 Scans

Mon 4-7 PM  
4-196  
Edit Details

Edit QR Code

#



Depth



#

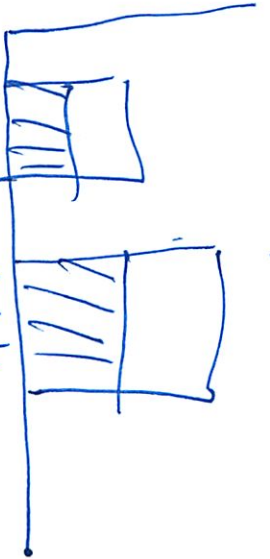


VP

Platform



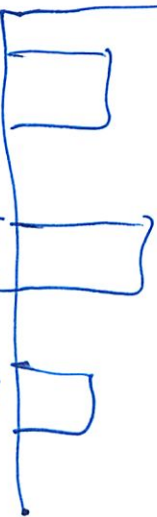
Phone  
Android  
Moto



ICS

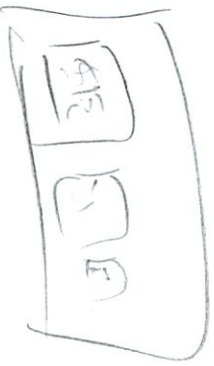
Emul

6.0

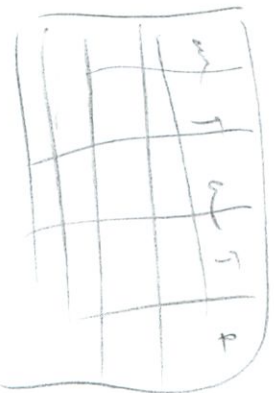


Rescued N Y G

Q R code checker



My Evans



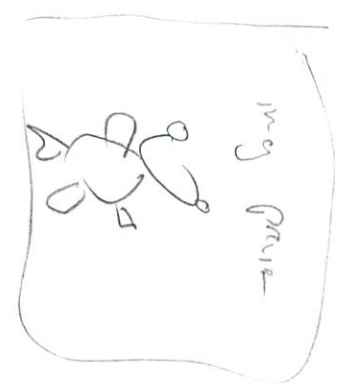
R S C N VP

Phase | Events | Account



②  
Help

Input map



Color wheel / Selector

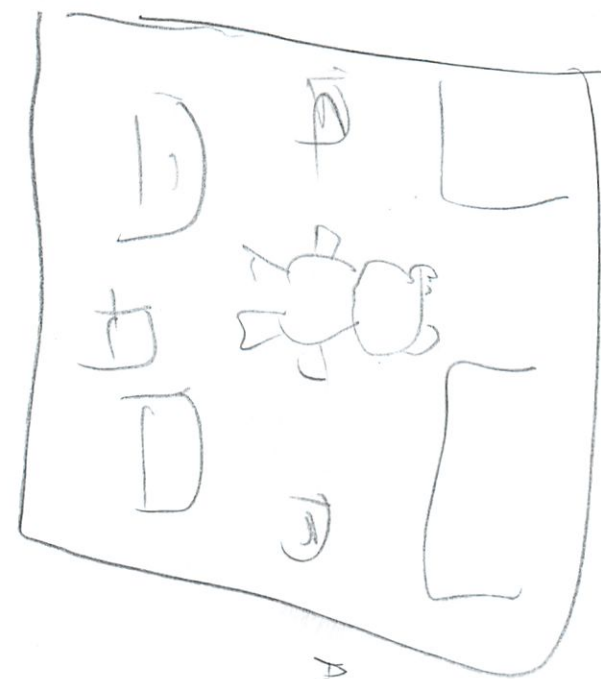


Color  
Size  
Shape  
Texture

Size settings

Export

Save out



Attack and Def.

Porta





Title: \_\_\_\_\_  
Date: \_\_\_\_\_  
Location: \_\_\_\_\_  
Description: \_\_\_\_\_  
Coordinates: \_\_\_\_\_

Tenn

See Yearbook

Add to site

ASUP

R Sc D2 N VP

---

You Have 5 current Elands

---

Horse Race

588 1850

Yacht Race

45 1850

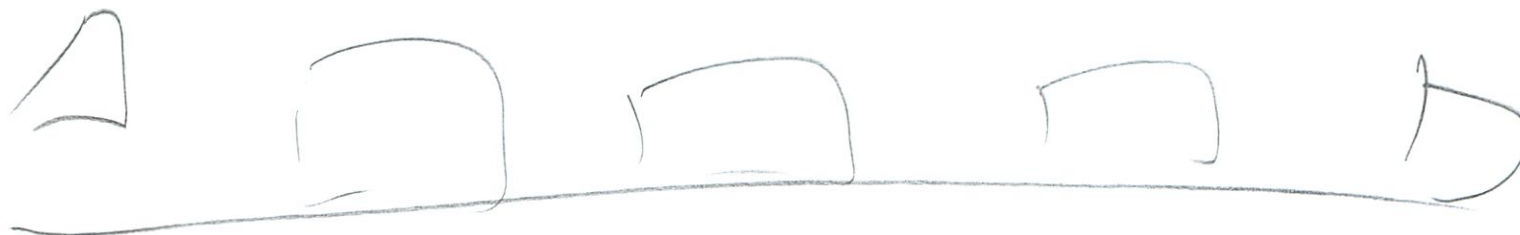
~~R~~ Croquet Tournament

2 1850

---


You Have 17 Previous Elands

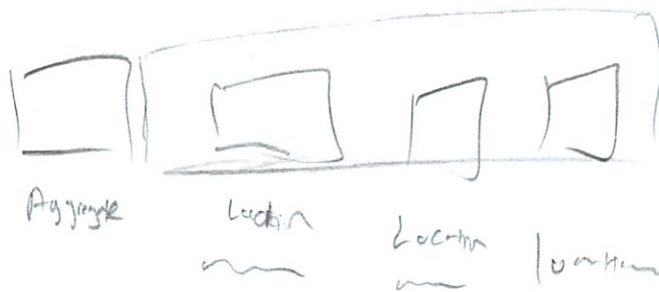
---



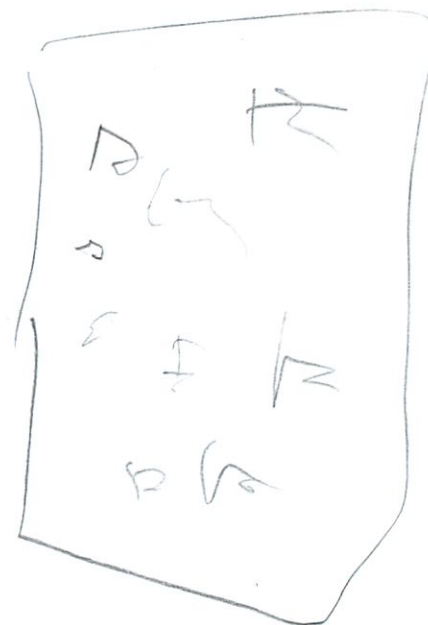
R SC  VP

Statistics

 Total RSVPs  
- 498



Guest Name	13+	13-	Photo
John Pang	j pang	7/1	17 hrs ago
~~~~~	~~~~~		
~~~~~	~~~~~		
~~~~~	~~~~~		
~~~~~	~~~~~		
~~~~~	~~~~~		



# RS c NVP

---

You have 3 current Events

---

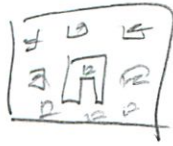


You Have 17 previous events

---



# RSC NVP



Home Make QR Settings

▶ Total RSVPs  
- 498

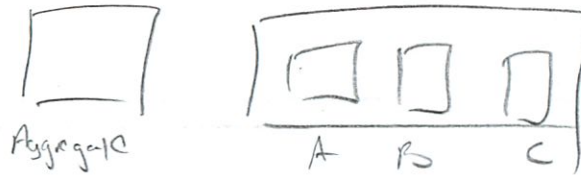


▶ Total Scans  
573

▶ Guest Lists

Name	Email	Time
Jon Pany	jpan8	17 hrs ago
✓	✓	✓
✓	✓	✓
✓	✓	✓
✓	✓	✓

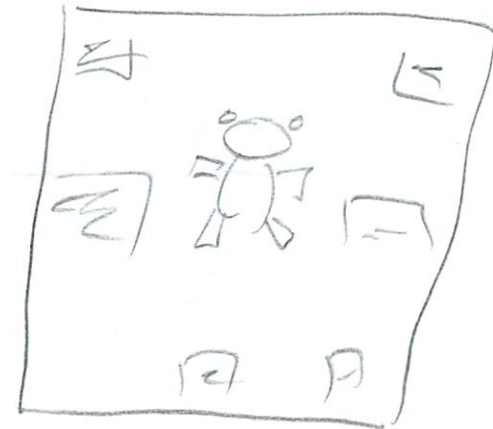
By location



Event  
Date

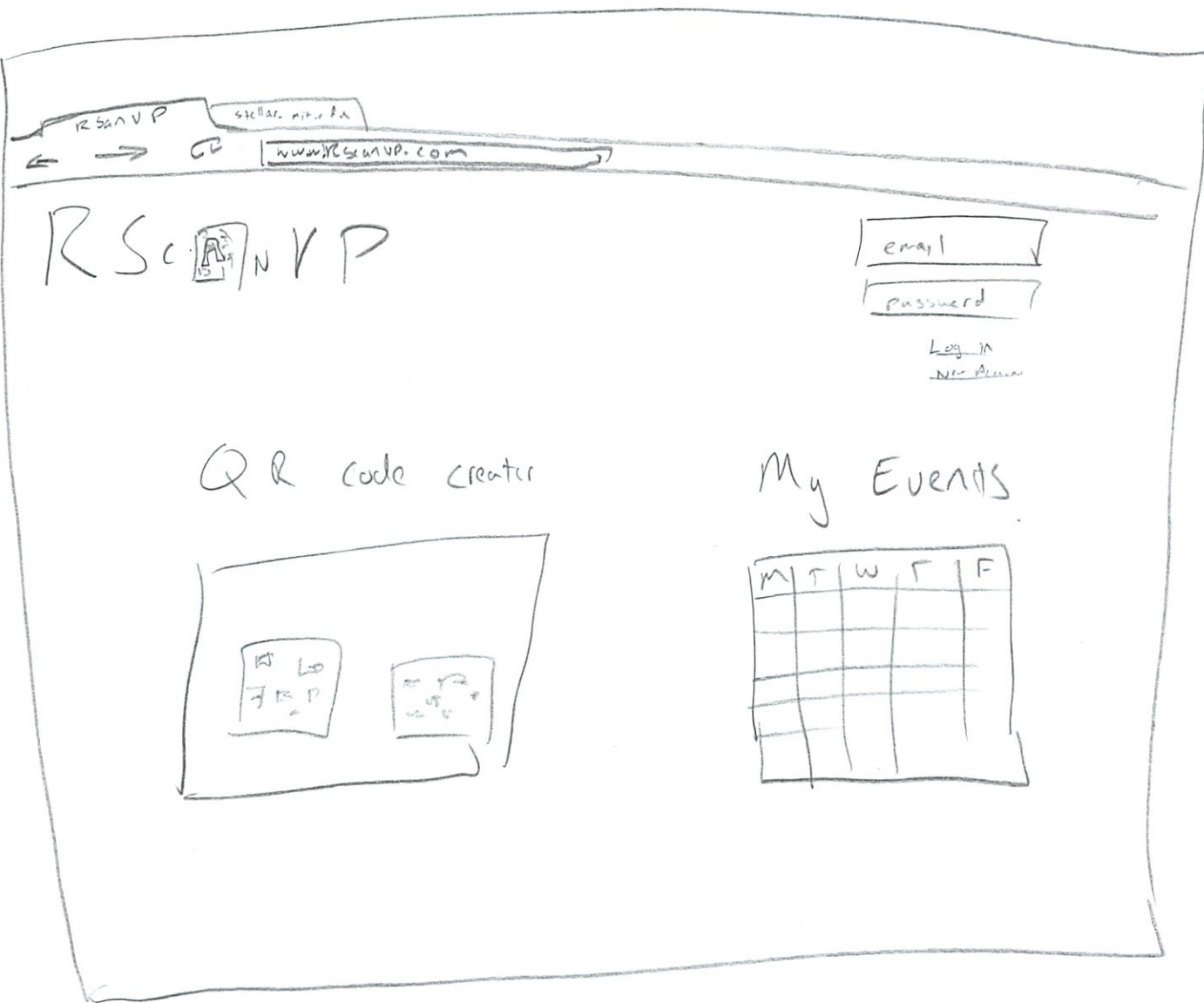
Time

Description



Edit Details





# R S C O N V P

Home

Events

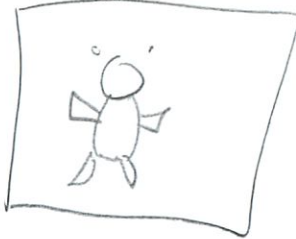
Account

(?)

Help

Import Image

My Picture



Color

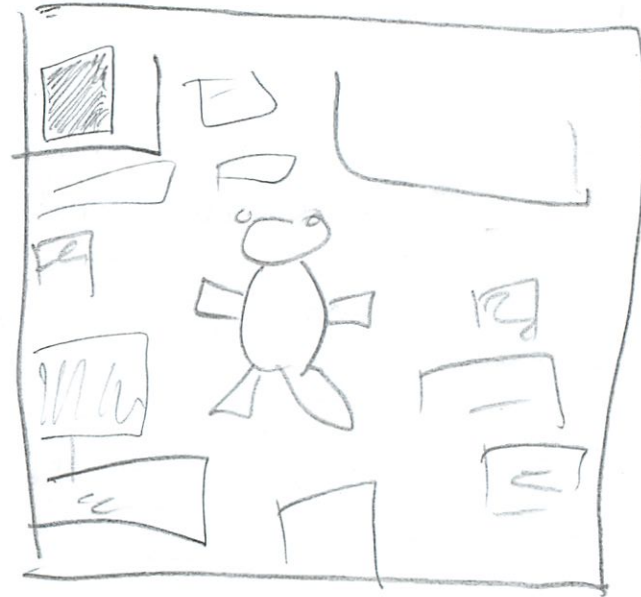
R      
G      
B

Color Selector



3x3 ☐ 1x16 ☐ Size settings

24x24 ☐ ☐ Custom



Export

Save

3/12

6.813 L13

Experimental Design

(optional session)

UI Hall Fare/share: Office 2003 expanding menus

trying to make more efficient  
but changes position  
and stuff hidden

Or Office Font list - split menu

Proved in study

- users learned where to go for frequent things

---

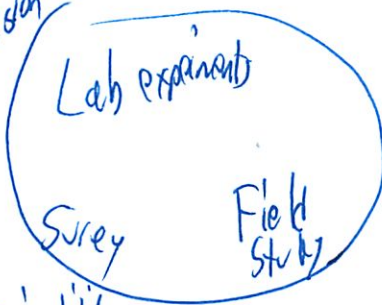
no nano quiz

---

Today's Research Methods

Methods

Precision



Realism

↑ Abstract

↓ Concrete

Generalizability  
(can reach a large amt)

← Obtrusive

→ Unobtrusive

(2) Web experiments are

---

### Quantifying

Efficiency: Time

Learnability: Time 1st time

Safety: Cont errors

---

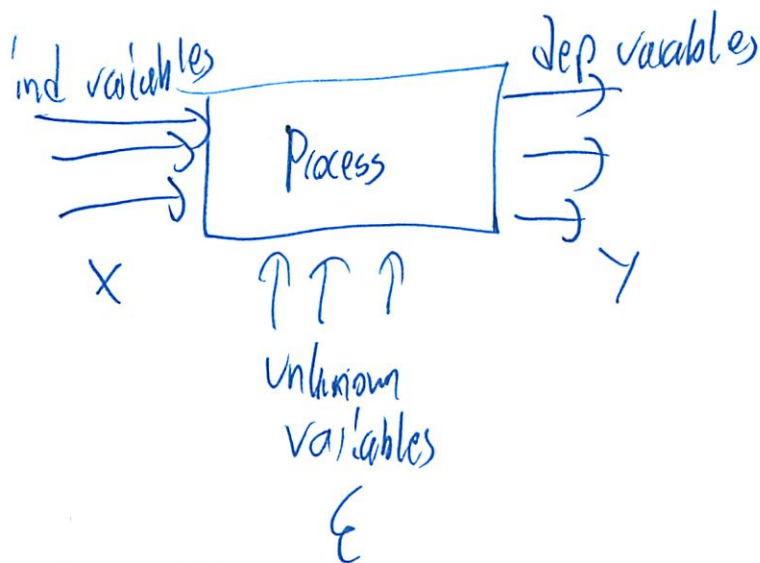
### Controlled Experiment

1. Testable hyp.

2. Ind variables

3. Measure dep variables

4. Use Stat. Tests to accept/reject



$$Y = f(X) + \epsilon$$

③ We hope the  $\epsilon$  terms do not correlate w/  $X$  terms

Who will we recruit?

Age?

Handedness?

Experience of OS

How implement?

Artificial infrastructure or real?

May be some stuff we did not prove

Realistic tasks?

How often do you do task?

↳ More often get more data points

Plus might be different in real life

When start/stop timing?

Concerns

Internal Validity

- are results caused by ind variables?

External Validity

- (can results be generalized to world outside the lab?)

Reliability

- could you repeat + get same results?



④

95% confidence - means  $\frac{1}{20}$  times it will fail

---

(Marbles example in slide)

TARE the boxes

Use ~~same~~ same box

Measure a sample of marbles

---

Common Threats to Validity

- Ordering effects
  - people learn
  - can't unlearn
  - could randomize order
- or selection of people
  - people live w/ friends (homophily)
  - instead randomize
- experimenter bias
  - you want one to win
  - hard to do a double blind

⑤

## Threats to External Validity

### Population

- Usually you pick MIT students
- Reviewers don't like

### Ecological

- lab is like an office
- but it's something else (mobile, etc)

### Training

- mimic has real VT

### Tasks

- does it reflect real tasks?

---

## Threats to Reliability

- Same computer, room, etc
- but can't control everything
- users vary in prior experience
- large difference in users
- other stuff: person stops to blow their nose

(6)

## Ephemeral Adaptation

prompt not really realistic

Users might picture instead

'Designed for novice users'

telling them which menu to go to

learn menu bar over trials

Using mediators

- often a good thing to use for humans

- instead of means

Give details

- shows controlling

- so you can replicate

Want sample to look like population at large

---

Between subjects vs within subjects

Hard to compare

User A to

User B

Hard for user

to unlearn for

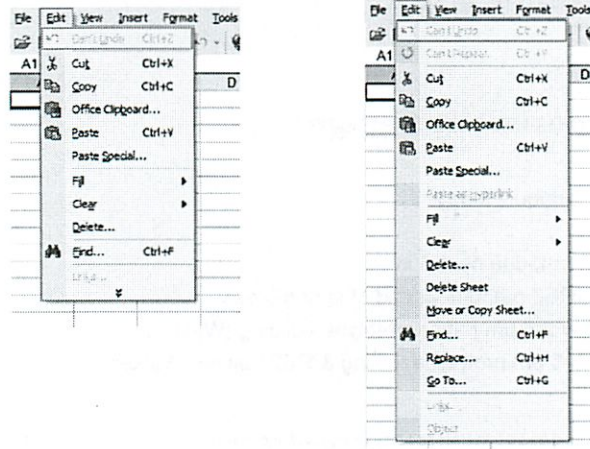
2nd test

## **L13: Experiment Design**

- No nanoquiz today
- GR3 out, due next Sun
- PS2/RS2 out, due at end of spring break
- Today 3-5 pm paper prototype building (Walker)
- WF 3-5 pm prototype testing & RS2 testing (Walker)



## UI Hall of Fame or Shame?



Spring 2012

6.813/6.831 User Interface Design and Implementation

2

Today's candidate for the Hall of Fame or Shame is **adaptive menus**, a feature of Microsoft Office 2003. Initially, a menu shows only the most commonly used commands. Clicking on the arrow at the bottom of the menu expands it to show all available commands. Adaptive menus track how often a user invokes each command, in order to display frequently-used commands and recently-used commands.

Let's discuss the usability of this idea.

This interface is striving for a compromise between **simplicity** (i.e., providing as few features as possible) and **task analysis** (supporting the tasks required by many users, and trying to adapt to the common tasks of each individual user). Both properties are important for usability. Unfortunately they also compete with each other. Adaptive menus are an interesting hybrid technique that's trying to satisfy both.

The downside is lack of predictability. Because the menu may change in complex and unpredictable ways – with new items appearing and underused items disappearing for no visible reason – the user can no longer rely on a lot of useful cues to find menu items. One of these cues that's lost is spatial memory – Paste may be found at different distances down the menu each time the menu appears. Another missing cue is context: Paste's neighbors may change as well.

Another downside is lack of user control. The adaptation happens automatically; the user can't manually fixate or remove items from a menu.

This particular implementation of adaptive menus has some specific usability problems. When the full menu appears, the new items are inserted into place, and there's very little **contrast** in the graphic design to distinguish between the old items and the new items. So the user has to search through the entire menu again.

But this particular implementation addresses other usability problems very well. When the user is hunting through all the menus, looking for a command, the full menu only has to be requested once; then all subsequent menus are fully displayed.



## UI Hall of Fame or Shame?



Spring 2012

6.813/6.831 User Interface Design and Implementation

3

Here's an alternative approach to providing easy access to high-frequency menu choices: a **split menu**. You can see it in Microsoft Office's font drop-down menu. A small number of fonts that you've used recently appear in the upper part of the menu, while the entire list of fonts is available in the lower part of the menu. The upper list is sorted by recency, while the lower part is sorted alphabetically.

Let's discuss the split menu approach too.

These menu approaches are particularly relevant to today's lecture because they've been tested by a couple of excellent controlled experiments. The split menu idea was evaluated by Sears & Shneiderman, "Split menus: effectively using selection frequency to organize menus", *ACM TOCHI*, March 1994. And the adaptive menu was tested by Findlater & McGrenere, "A comparison of static, adaptive, and adaptable menus", *CHI 2004*.

Sears & Shneiderman: compared alphabetic, frequency-ordered, and split menus (with up to 4 high-frequency items at top of menu, **ordered in same way** as rest of menu, **removed** from the rest of menu – so not the same as Office's font split menu). 15-item menus (randomly selected from a dictionary of 1000 common words). Three different frequency distributions across the alphabetic menu (end of menu, middle of menu, top of menu). 36 subjects, within-subjects, each saw 3x3 menus (random unique items) counterbalanced. Pulldown menus, timed from mouse press on menu bar until selection of item. 100 trials per menu, chosen from frequency distribution of menu. Measured time and ranked preference of menu type (1-3). Subjective ranking had split (1.4) > alphabetic (2.0) > frequency (2.6). Selection time for frequent items at end of menu had split (1.4s) > freq (1.5s) > alphabetic (1.7s); for frequent items at start of menu had split, alphabetic (1.4s) > freq (1.5s). Also proposed and fitted a cognitive model that high-frequency menu items take time logarithmic in item position (a la Fitts's Law) while low-frequency items take linear time in position (visual scan).

Findlater & McGrenere: compared split menus that were static (unchanging), adaptive (changed by system), and adaptable (changed by user). Adaptive menu has two most recent and two most frequent items in the split part. Adaptable had arrow buttons next to each item to promote or demote it. Used frequency distributions from three most frequently-used Word menus (File, Format, Insert), collected from one user over 20 weeks, but changed all item names to mask them. Measured time and ranked preference (on several dimensions: overall, efficiency, ease, error, frustration, etc.). 27 subjects, within-subjects 3x3. Selection time had static (1.5), adaptable (1.6) >> adaptive (1.65). Overall preference had adaptable >> adaptive >> static.

## Research Methods in HCI

- Lab experiment
- Field study
- Survey

Spring 2012

6.813/6.831 User Interface Design and Implementation

4

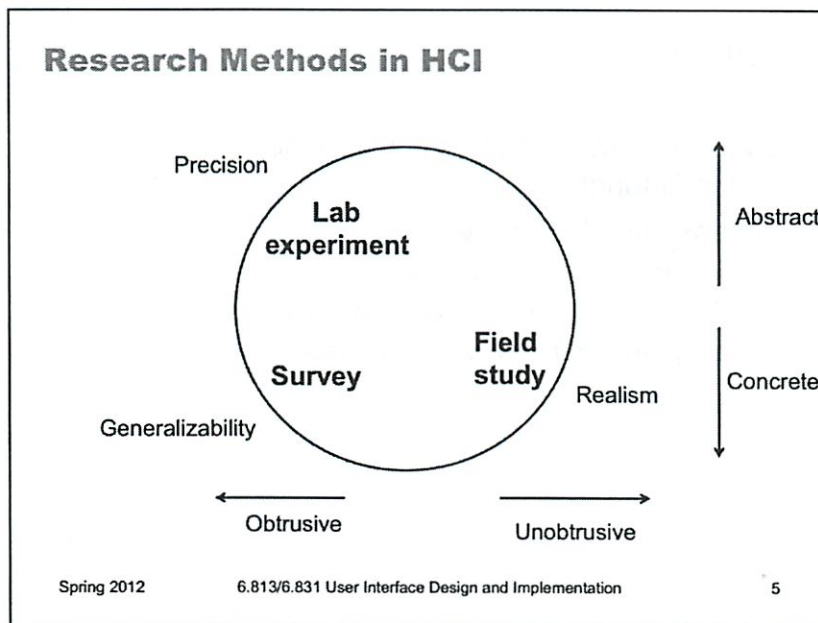
We'll start by talking about the main kinds of research methods in human-computer interaction. Here we mean **empirical methods**, techniques for investigating the world and collecting evidence to prove or disprove our hypotheses about how people interact with computers, and about the usability of interfaces. These methods are widely used across other kinds of research involving people, including psychology, cognitive science, sociology, and economics. When we talk about an HCI research paper and look for the **evaluation** of the paper's claim, we will often find one of these methods. Note that these are not the *only* acceptable kinds of evaluation in human-computer interaction. In particular, this picture (and this lecture) largely ignore evaluation methods that are relevant to the **computer** side of the human-computer interface: issues like performance, reliability, security; proof that a system or toolkit is broadly applicable; etc.

A **lab experiment** is an artificial situation, created by and highly controlled by the experimenter, that typically compares alternative user interfaces or measures how usability varies with some design parameter. One example might be a test of font readability, done by bringing subjects into the experimenter's lab, asking them to read texts displayed with different fonts, and timing their reading speed.

A **field study** is a real situation, happening in the actual environment where people use the interface under consideration, and using real tasks (rather than tasks concocted by the experimenter). Social scientists make distinctions between pure field studies (where the researcher acts like an anthropologist or ethnographer, intervenes as little as possible, hiding in the bushes, so to speak) and field *experiments* (where the researcher introduces something new, like a new system, new process, or new UI). In HCI, which is still a young field and still striving to make something new, these distinctions end up being a matter of *when* the study happens to be done in the evolution of your project. Initial field studies just observe without intervening (contextual inquiry, which we discussed in the task analysis lecture, is a technique like this). Final field studies deliver the new UI and see how it's used.

A **survey** is a questionnaire, conducted by paper, phone, web, or in person.





The picture shows how these methods compare on several interesting dimensions. In the vertical dimension, lab experiments often use highly **abstract** tasks, which are simplified and highly controlled in order to make strong statistical claims, but also farther removed from the real-world. An example is the classic Fitts's Law experiment, which measures the time it takes to hit targets on the screen by using plain rectangles rather than realistic pointing targets like buttons, menus, and hyperlinks. Field studies and surveys, by contrast, can use more concrete, real-world examples.

In the horizontal direction, lab experiments and surveys are **obtrusive**: people need to be interrupted and forced to give their attention to the study. Because they're aware they're being studied, their behavior can change; we'll mention some of the ways that can happen later in this lecture. Field studies, on the other hand, can be far less obtrusive, since the subjects are doing their own tasks in their own environment, and so the likelihood of obtaining natural behavior is much higher.

Finally, the picture also shows how the three methods compare on three criteria which are all desirable but virtually impossible to obtain at once from a single method. **Realism** means whether the phenomena captured are actually what happens in the real world (in real contexts on real tasks); field studies are strongest on that. **Generalizability** concerns whether the results apply to the whole population of *people* relevant to the study; a survey is strongest on that, because it's far cheaper to survey a large number of people, and good statistical sampling techniques exist to make the results generalizable. **Precision** means control over measurement error and extraneous factors that might introduce noise into the results; lab experiments are strongest on that, because they can tightly control the tasks and environment to eliminate as much error as possible.

(This picture actually omits one important class of methods, which is occasionally but not widely used in HCI research: a theoretical model or computer simulation of human behavior. We talked about some theoretical models for HCI, like KLM and GOMS, in the efficiency lecture.)

## Quantifying Usability

- Usability: how well users can use the system's functionality
- Dimensions of usability
  - Learnability: is it easy to learn?
  - Efficiency: once learned, is it fast to use?
  - Safety: are errors few and recoverable?

Spring 2012

6.813/6.831 User Interface Design and Implementation

6

In this lecture, we'll be focusing on controlled experiments – particularly those that try to measure the **usability** of alternative user interface designs. Recall our definition of usability, and how we broke it down into several dimensions. We can **quantify** all these measures of usability, which will be essential to doing an experiment. Just as we can say algorithm X is faster than algorithm Y on some workload, we can say that interface X is more learnable, or more efficient, or less error-prone than interface Y for some set of tasks and some class of users, by designing an experiment that measures the two interfaces.

## Controlled Experiment

- Start with a testable **hypothesis**
  - e.g. Mac menu bar is faster than Windows menu bar
- Manipulate **independent variables**
  - different interfaces, user classes, tasks
  - in this case, y-position of menubar
- Measure **dependent variables**
  - times, errors, # tasks done, satisfaction
- Use statistical tests to accept or reject the hypothesis

Spring 2012

6.813/6.831 User Interface Design and Implementation

7

Here's a high-level overview of a controlled experiment. You start by stating a clear, **testable** hypothesis. By testable, we mean that the hypothesis must be quantifiable and measurable. Here's an example of a hypothesis that we might want to test: that the Macintosh menu bar, which is anchored to the top of the screen, is faster to access than the Windows menu bar, which is separated from the top of the screen by a window title bar.

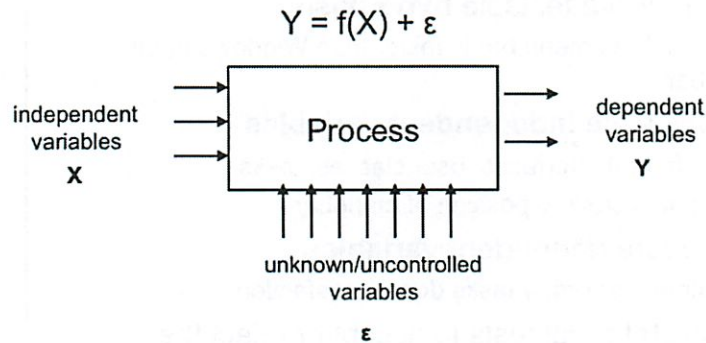
You then choose your **independent variables** – the variables you're going to manipulate in order to test the hypothesis. In our example, the independent variable is the kind of interface: Mac menubar or Windows menubar. In fact, we can make it very specific: the independent variable is the y-position of the menubar (either  $y = 0$  or  $y = 16$ , or whatever the height of the title bar is). Other independent variables may also be useful. For example, you may want to test your hypothesis on different user classes (novices and experts, or Windows users and Mac users). You may also want to test it on certain kinds of tasks. For example, in one kind of task, the menus might have an alphabetized list of names; in another, they might have functionally-grouped commands.

You also have to choose the **dependent variables**, the variables you'll actually measure in the experiment to test the hypothesis. Typical dependent variables in user testing are time, error rate, event count (for events other than errors – e.g., how many times the user used a particular command), and subjective satisfaction (usually measured by numerical ratings on a questionnaire).

Finally, you use statistical techniques to analyze how your changes in the independent variables affected the dependent variables, and whether those effects are **significant** (indicating a genuine cause-and-effect) or not (merely the result of chance or noise). We'll say more about statistical tests in the next lecture.



## Schematic View of Experiment Design



Spring 2012

6.813/6.831 User Interface Design and Implementation

8

Here's a block diagram to help you visualize what we're trying to do with experiment design. Think of the process you're trying to understand (e.g., menu selection) as a black box, with lots of inputs and a few outputs. A controlled experiment twiddles some of the input knobs on this box (the independent variables) and observes some of the outputs (the dependent variables) to see how they are affected.

The problem is that there are many *other* input knobs as well (unknown/uncontrolled variables), that may affect the process we're observing in unpredictable ways. The purpose of experiment design is to eliminate the effect of these unknown variables, or at least render harmless, so that we can draw conclusions about how the independent variables affect the dependent variables.

What are some of these unknown variables? Consider the menubar experiment. Many factors might affect how fast the user can reach the menubar: the pointing device they're using (mouse, trackball, isometric joystick, touchpad); where the mouse pointer started; the surface they're moving the mouse on; the user's level of fatigue; their past experience with one kind of menubar or the other. All of these are unknown variables that might affect the dependent variable (speed of access).

## Design of the Menubar Experiment

- Users
  - Windows users or Mac users?
  - Age, handedness?
  - How to sample them?
- Implementation
  - Real Windows vs. real Mac
  - Artificial window manager that lets us control menu bar position
- Tasks
  - Realistic: word processing, email, web browsing
  - Artificial: repeatedly pointing at fake menu bar
- Measurement
  - When does movement start and end?
- Ordering
  - of tasks and interface conditions
- Hardware
  - mouse, trackball, touchpad, joystick?
  - PC or Mac? which particular machine?

Spring 2012

6.813/6.831 User Interface Design and Implementation

9

Here are some of the issues we'd have to consider in designing the menubar experiment.

First, what user population do we want to sample? Does experience matter? Windows users will be more experienced with one kind of menu bar, and Mac users with the other. On the other hand, the model underlying our hypothesis (Fitts's Law for pointing tasks) is largely independent of long-term memory or practice, so we might expect that experience doesn't matter. But that's another hypothesis we should test, so maybe past experience should be an independent variable that we select when sampling.

How do we sample the user population we want? The most common technique (in academia) is advertising around a college campus, but this biases against older users and less-educated users. *Any* sampling method has biases; you have to collect demographic information, report it, and consider whether the bias influences the generalizability of your results. What implementation should we test? One possibility is to test the menu bars in their real context: inside the Mac interface, and inside the Windows interface, using real applications and real tasks. This is more realistic, but the problem is now the presence of confounding variables -- the *size* of the menu bars might be different, the reading speed of the font, the mouse acceleration parameters, etc. We need to control for as many of these variables as we can. Another possibility is implementing our own window manager that holds these variables constant and merely changes the position of the menu bar.

What tasks should we give the user? Again, having the user use the menu bar in the context of realistic tasks might provide more generalizable results; but it would also be noisier. An artificial experiment that simply displays a menu bar and cues the user to hit various targets on it would provide more reliable results. But then the user's mouse would always be in the menu bar, which isn't at all realistic. We'd need to force the user to move the mouse out of the menu bar between trials, perhaps to some home location in the middle of the screen.

How do we **measure** the dependent variable, time? Maybe from the time the user is given the cue ("click Edit") to the time the user successfully clicks on Edit.

What **order** should we present the tasks and the interface conditions? Using the same order all the time can cause both learning effects (e.g., you do better on the interface conditions because you got practice with the tasks and the study protocol during the first interface condition) and fatigue effects (where you do worse on later interface conditions because you're getting tired).

Finally, the hardware we use for the study can introduce lots of confounding variables. We should use the same computer for the entire experiment -- across both users and interface conditions.



## Concerns Driving Experiment Design

- Internal validity
  - Are observed results actually **caused** by the independent variables?
- External validity
  - Can observed results be **generalized** to the world outside the lab?
- Reliability
  - Will consistent results be obtained by **repeating** the experiment?

Spring 2012

6.813/6.831 User Interface Design and Implementation

10

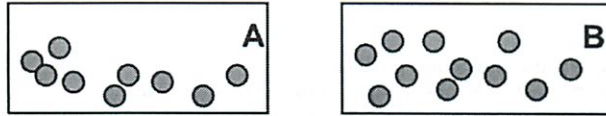
Unknown variation is the bugaboo in experiment design, and here are the three main problems it can cause.

**Internal validity** refers to whether the effect we see on the experiment outputs was actually caused by the changes we made to the inputs, or caused by some unknown variable that we didn't control or measure. For example, suppose we designed the menubar experiment so that the Mac menubar position was tested on an actual Mac, and the Windows menubar position was tested on a Windows box. This experiment wouldn't be internally valid, because we can't be sure whether differences in performance were due to the difference in the position of the menubar, or to some other (unknown) difference in two machines -- like font size, mouse acceleration, mouse feel, even the system timer used to measure the performance! Statisticians call this effect **confounding**; when a variable that we didn't control has a systematic effect on the dependent variables, it's a **confounding variable**.

One way to address internal validity is to hold variables constant, as much as we can: for example, conducting all user tests in the same room, with the same lighting, the same computer, the same mouse and keyboard, the same tasks, the same training. The cost of this approach is **external validity**, which refers to whether the effect we see can be generalized to the world outside the lab, i.e. when those variables are not controlled. But when we try to control *all* the factors that might affect menu access speed -- a fixed starting mouse position, a fixed menubar with fixed choices, fixed hardware, and a single user -- then it would be hard to argue that our lab experiment generalizes to how menus are used in the varying conditions encountered in the real world.

Finally, **reliability** refers to whether the effect we see (the relationship between independent and dependent variables) is repeatable. If we ran the experiment again, would we see the same effect? If our experiment involved only one trial -- clicking the menubar just once -- then even if we held constant every variable we could think of, unknown variations will still cause error in the experiment. A single data sample is rarely a reliable experiment.

## How Many Marbles in Each Box?



- Hypothesis: box A has a different number of balls than box B
- Reliability
  - Counting the balls manually is reliable only if there are few balls
  - Repeated counting improves reliability
- Internal validity
  - Suppose we weigh the boxes instead of counting balls
  - What if an A ball has different weight than a B ball?
  - What if the boxes themselves have different weights?
- External validity
  - Does this result apply to all boxes in the world labeled A and B?

Spring 2012

6.813/6.831 User Interface Design and Implementation

11

Here's a simple example to illustrate internal validity, external validity, and reliability.

Suppose we have two boxes labeled A and B, and each box has some marbles inside. We want to test the **hypothesis** that these two boxes have different numbers of marbles. (For example, maybe each box is from a different manufacturer, but both boxes are the same price, and we want to claim that one box is a better deal than the other.) In this case, the **independent variable** is the identity of the box (the brand of the manufacturer), and the **dependent variable** we're trying to measure is the number of marbles inside the box.

This hypothesis is **testable** because we can do measurements on the boxes to test it. One measurement we might do is open the boxes and count the marbles inside each one. If we counted 3 marbles in one box, and 5 marbles in the other box, then we'd have strong evidence that A has fewer marbles than B. But if we counted 99 marbles in A and 101 marbles in B, would we be just as confident? No, because we know that our counting may have errors -- we may skip marbles, or we may double-count marbles. If we counted A again, we might come up with 102 marbles. The **reliability** of a single measurement isn't good enough. We can fix this reliability problem by better experiment design: counting each box several times, and computing a **statistic** (a summary of the measurements) instead of using a single measurement. The statistic is often the mean of the measurements, but others are possible.

Counting is pretty slow, though, so to make our experiment cheaper to run, suppose we weigh the boxes instead. If box A is lighter than box B, then we'll say that it has fewer marbles in it. It's easy to see the possible flaws in this experiment design: what if each marble in box A is lighter than each marble in box B? Then the whole box A might be lighter than B even if it has more marbles. If this is true, then it threatens the experiment's **internal validity**: the dependent variable we've chosen to measure (total weight) is a function not only of the number of marbles in the box, but also of the weight of each marble, and the weight of the marble also varies by the independent variable (the identity of the box).

Suppose the weight of marbles is regulated by an international standard, so we can argue that A marbles should weigh the same as B marbles. But what if box A is made out of metal, and box B is made out of cardboard? This would also threaten internal validity -- the weight of the box itself is a confounding variable. But now we can imagine a way to **control** for this variable, by pouring the marbles out of their original boxes into box C, a box that we've chosen, and weighing both sets of marbles separately in box C. We've changed our experiment design so that the weight of the box itself is held constant, so that it won't contribute to the measured difference between box A and box B.

But we've only done this experiment with the box A and box B we have in front of us. What confidence do we have in concluding that *all* boxes from manufacturer A have a different number of marbles than those from manufacturer B -- that if we went to the store and picked up new boxes of marbles, we'd see the same difference? That's the question of **external validity**.



## Threats to Internal Validity

- Ordering effects
  - People learn, and people get tired
  - Don't present tasks or interfaces in same order for all users
  - Randomize or counterbalance the ordering
- Selection effects
  - Don't use pre-existing groups (unless group is an independent variable)
  - Randomly assign users to independent variables
- Experimenter bias
  - Experimenter may be enthusiastic about interface X but not Y
  - Give training and briefings on paper, not in person
  - Provide equivalent training for every interface
  - Double-blind experiments prevent both subject and experimenter from knowing if it's condition X or Y
    - Essential if measurement of dependent variables requires judgement

Spring 2012

6.813/6.831 User Interface Design and Implementation

12

Let's look closer at typical dangers to internal validity in user interface experiments, and some solutions to them. You'll notice that the solutions tend to come in two flavors: randomization (which prevents unknown variables from having systematic effects on the dependent variables) and control (which tries to hold unknown variables constant).

**Ordering effects** refer to the order in which different levels of the independent variables are applied. For example, does the user work with interface X first, and then interface Y, or vice versa? There are two effects from ordering: first, people learn. They may learn something from using interface X that helps them do better (or worse) with interface Y. Second, people get tired or bored. After doing many tasks with interface X, they may not perform as well on interface Y. Clearly, holding the order constant threatens internal validity, because the ordering may be responsible for the differences in performance, rather than inherent qualities of the interfaces. One good solution to this threat is **randomization**: present the interfaces, or tasks, or other independent variables in a random order to each user. Another good solution is counterbalancing (see a later slide).

**Selection effects** arise when you use pre-existing groups as a basis for assigning different levels of an independent variable. Giving the Mac menubar to artists and the Windows menubar to engineers would be an obvious selection effect. More subtle selection effects can arise, however. Suppose you let the users line up, and then assigned the Mac menubar to the first half of the line, and Windows menubar to the second half. This procedure seems "random", but it isn't – the users may line up with their friends, and groups of friends tend to have similar activities and interests. The same thing can happen even if people are responding "randomly" to your study advertisement – you don't know how "random" that really is! The only safe way to eliminate selection effects is honest randomization.

A third important threat to internal validity is **experimenter bias**. After all, you have a hypothesis, and you're hoping it works out – you're rooting for interface X. This bias is an unknown variable that may affect the outcome, since you're personally interacting with the user whose performance you're measuring. One way to address experimenter bias is **controlling** the protocol of the experiment, so that it doesn't vary between the interface conditions. Provide equivalent training for both interfaces, and give it on paper, not live.

An even better technique for eliminating experimenter bias is the **double-blind experiment**, in which neither the subject nor the experimenter knows which condition is currently being tested. Double-blind experiments are the standard for clinical drug trials, for example; neither the patient nor the doctor knows whether the pill contains the actual experimental drug, or just a placebo. With user interfaces, double-blind tests may be impossible, since the interface condition is often obvious on its face. (Not always, though! The behavior of cascading submenus isn't obviously visible, for example.)

Experimenter-blind tests are essential if measurement of the dependent variables requires some subjective judgement. Suppose you're developing an interface that's supposed to help people compose good memos. To measure the quality of the resulting memos, you might ask some people to evaluate the memos created with the interface, along with memos created with a regular word processor. But the memos should be presented in random order, and you should hide the interface that created each memo from the judge, to avoid bias.



## Threats to External Validity

- Population
  - Draw a random sample from your real target population
- Ecological
  - Make lab conditions as realistic as possible in important respects
- Training
  - Training should mimic how real interface would be encountered and learned
- Task
  - Base your tasks on task analysis

Spring 2012

6.813/6.831 User Interface Design and Implementation

13

Here are some threats to external validity that often come up in user studies. If you've done a thorough user analysis and task analysis, in which you actually went out and observed the real world, then it's easier to judge whether your experiment is externally valid.

**Population** asks whether the users you sampled are representative of the target user population. Do your results apply to the entire user population, or only to the subgroup you sampled? The best way to ensure population validity is to draw a random sample from your real target user population. But you can't really, because users must choose, of their own free will, whether or not to participate in a study. So there's a **self-selection** effect already in action. The kinds of people who participate in user studies may have special properties (extroversion? curiosity? sense of adventure? poverty?) that threaten external validity. But that's an inevitable effect of the ethics of user testing. The best you can do is argue that on important, measurable variables – demographics, skill level, experience – your sample resembles the overall target user population.

**Ecological validity** asks whether conditions in the lab are like the real world. An office environment would not be an ecologically valid environment for studying an interface designed for the dashboard of a car, for example.

**Training validity** asks whether the interfaces tested are presented to users in a way that's realistic to how they would encounter them in the real world. A test of an ATM machine that briefed each user with a 5-minute tutorial video wouldn't be externally valid, because no ATM user in the real world would watch such a video. For a test of an avionics system in an airplane cockpit, on the other hand, even hours of tutorial may be externally valid, since pilots are highly trained.

Similarly, **task validity** refers to whether the tasks you chose are realistic and representative of the tasks that users actually face in the real world. If you did a good task analysis, it's not hard to argue for task validity.

## Threats to Reliability

- Uncontrolled variation
  - Previous experience
    - Novices and experts: separate into different classes, or use only one class
  - User differences
    - Fastest users are **10 times** faster than slowest users
  - Task design
    - Do tasks measure what you're trying to measure?
  - Measurement error
    - Time on task may include coughing, scratching, distractions
- Solutions
  - Eliminate uncontrolled variation
    - Select users for certain experience (or lack thereof)
    - Give all users the same training
    - Measure dependent variables precisely
  - Repetition
    - Many users, many trials
    - Standard deviation of the mean shrinks like the square root of N (i.e., quadrupling users makes the mean twice as accurate)

Spring 2012

6.813/6.831 User Interface Design and Implementation

14

Once we've addressed internal validity problems by either controlling or randomizing the unknowns, and external validity by sampling and experiment protocol design, reliability is what's left.

Here's a good way to visualize reliability: imagine a bullseye target. The center of the bullseye is the true effect that the independent variables have on the dependent variables. Using the independent variables, you try to aim at the center of the target, but the uncontrolled variables are spoiling your aim, creating a spread pattern. If you can reduce the amount of uncontrolled variation, you'll get a tighter shot group, and more reliable results.

One kind of uncontrolled variation is a user's previous experience. Users enter your lab with a whole lifetime of history behind them, not just interacting with computers but interacting with the real world. You can limit this variation somewhat by screening users for certain kinds of experience, but take care not to threaten external validity when you artificially restrict your user sample.

Even more variation comes from differences in ability – intelligence, visual acuity, memory, motor skills. The best users may be **10 times better** than the worst, an enormous variation that may swamp a tiny effect you're trying to detect.

Other kinds of uncontrolled variation arise when you can't directly measure the dependent variables. For example, the tasks you chose to measure may have their own variation, such as the time to understand the task itself, and errors due to misunderstanding the task, which aren't related to the difficulty of the interface and act only to reduce the reliability of the test. Time is itself a gross measurement which may include lots of activities unrelated to the user interface: coughing, sneezing, asking questions, responding to distractions.

One way to improve reliability eliminates uncontrolled variation by holding variables constant: e.g., selecting users for certain experience, giving them all identical training, and carefully controlling how they interact with the interface so that you can measure the dependent variables precisely. If you control too many unknowns, however, you have to think about whether you've made your experiment externally invalid, creating an artificial situation that no longer reflects the real world.

The main way to make an experiment reliable is **repetition**. We run many users, and have each user do many trials, so that the mean of the samples will approach the bullseye we want to hit. As you may know from statistics, the more trials you do, the closer the sample mean is likely to be to the true value. (Assuming the experiment is internally valid of course! Otherwise, the mean will just get closer and closer to the *wrong* value.) Unfortunately, the standard deviation of the sample mean goes down slowly, proportionally to the square root of the number of samples N. So you have to *quadruple* the number of users, or trials, in order to double reliability.

### Example: Ephemeral Adaptation Study

#### Task

The experimental task was a sequence of menu selections from an experimental system. A prompt across the top of the screen displayed the name of the item to be selected and the menu in which it was located. Three menus were positioned just below the prompt. Once the participant had correctly selected the target item, the prompt for the next trial would be displayed.

To mitigate the effect of an individual selection sequence, the same underlying sequence was used for all conditions and task blocks for a given participant, but the location of the menus was permuted for each condition to reduce learning across conditions. For example, if the first selection in the first condition was Menu 1, Item 3, then in the second condition it would be Item 3 of either Menu 2 or Menu 3. The underlying selection sequences were then masked with different menu item names in each task block and condition. Each menu was generated by randomly selecting 4 groups of 4 semantically related items from a set of 72 such groups, so that each group appeared only once.

Spring 2012

6.813/6.831 User Interface Design and Implementation

15

To help understand threats to internal validity, external validity, and reliability, let's talk about a study from the HCI research literature. These quotes are from L. Findlater, K. Moffat, J. McGrenere, J. Dawson. "Ephemeral Adaptation: The Use of Gradual Onset to Improve Menu Selection Performance". CHI 2009.



#### *Quantitative and Qualitative Measures*

Speed was measured using the median selection time, calculated as the time from opening the menu to selecting the correct item. The median was used to reduce the influence of outlier trials. We used an implicit error penalty in the speed measures; that is, participants could not advance to the next trial until they correctly completed the current trial. For completeness, we also recorded the error rate. Finally, subjective data was collected using 7-point Likert scales on difficulty, satisfaction, efficiency and frustration. At the end of the study, a questionnaire asked for comparative rankings of the menu conditions.

#### *Apparatus*

A 2.0GHz Pentium M laptop with 1.5 GB of RAM and Microsoft Windows XP was used for the experiment. The system was connected to an 18" LCD monitor with 1280x1024 resolution and the experiment was coded in Java 1.5. The system recorded all timing and error data.

#### *Participants*

Twenty-four participants (12 females) were recruited through on-campus advertising. All were regular computer users, were between the ages of 19–45 ( $M = 25.5$ ) and were reimbursed \$10 per hour to participate.



#### *Procedure*

The procedure was designed to fit into a single 1-hour session. Participants were first given a background questionnaire to collect demographic information. Then, for each condition participants completed a short 8-trial practice block of selections to familiarize themselves with the behavior of the menus before completing two longer 126-trial task blocks. Short breaks were given in the middle of each block and between blocks. After both task blocks, participants completed a questionnaire with the subjective Likert scale questions for that condition. Once all experimental tasks were complete, a comparative questionnaire was given.

Before each adaptive menu condition, participants were given a brief description of the adaptive behavior: they were told that some of the items would appear sooner than others, and that these were the items the system predicted would be most likely needed by the user. However, participants were not told the level of prediction accuracy.

## Blocking

- Divide samples into subsets which are more homogeneous than the whole set
  - Example: testing wear rate of different shoe sole material
  - Lots of variation between feet of different kids
  - But the feet on the same kid are far more homogeneous
  - Each child is a block
- Apply all conditions within each block
  - Put material A on one foot, material B on the other
- Measure difference within block
  - $\text{Wear(A)} - \text{Wear(B)}$
- Randomize within the block to eliminate internal validity threats
  - Randomly put A on left foot or right foot

Spring 2012

6.813/6.831 User Interface Design and Implementation

19

**Blocking** is another good way to eliminate uncontrolled variation, and therefore increase reliability. The basic idea is to divide up your samples up into blocks that are more homogeneous than the whole set. In other words, even if there is lots of uncontrolled variation between blocks, the blocks should be chosen so that there is little variation within a block. Once you've blocked your data, you apply **all** the independent variable conditions **within** each block, and compare the dependent variables only within the block.

Here's a simple example of blocking. Suppose you're studying materials for the soles of childrens' shoes, and you want to see if material A wears faster or slower than material B. There's much uncontrolled variation between feet of different children – how they behave, where they live and walk and play – but the two feet of the same child both see very similar conditions by comparison. So we treat each child as a block, and apply one sole material to one foot, and the other sole material to the other foot. Then we measure the difference between the sole wear as our dependent variable. This technique prevents inter-child variation from swamping the effect we're trying to see.

In agriculture, blocking is done in space. A field is divided up into small plots, and all the treatments (pesticides, for example) are applied to plants in each plot. Even though different parts of the field may differ widely in soil quality, light, water, or air quality, plants in the same plot are likely to be living in very similar conditions.

Blocking helps solve reliability problems, but it doesn't address internal validity. What if we always assigned material A to the left foot, and material B to the right foot? Since most people are right-handed and left-footed, some of the difference in sole wear may be caused by footedness, and not by the sole material. So you should still randomize assignments within the block.

## Between Subjects vs. Within Subjects

- "Between subjects" design
  - Users are divided into two groups:
    - One group sees only interface X
    - Other group sees only interface Y
  - Results are compared **between** different groups
    - Is  $\text{mean}(x_i) > \text{mean}(y_j)$ ?
  - Eliminates variation due to ordering effects
    - User can't learn from one interface to do better on the other
- "Within subjects" design
  - Each user sees both interface X and Y (in random order)
  - Results are compared **within** each user
    - For user  $i$ , compute the difference  $x_i - y_i$
    - Is  $\text{mean}(x_i - y_i) > 0$ ?
  - Eliminates variation due to user differences
    - User only compared with self

Spring 2012

6.813/6.831 User Interface Design and Implementation

20

The idea of blocking is what separates two commonly-used designs in user studies that compare two interfaces. Looking at these designs also gives us an opportunity to review some of the concepts we've discussed in this lecture.

A **between-subjects** design is unblocked. Users are randomly divided into two groups. These groups aren't blocks! Why? First, because they aren't more homogeneous than the whole set – they were chosen randomly, after all. And second, because we're going to apply only one independent variable condition within each group. One group uses only interface X, and the other group uses only interface Y. The performance of the X group is then compared with the performance of the Y group. This design eliminates variation due to interface ordering effects. Since users only see one interface, they can't transfer what they learned from one interface to the other, and they won't be more tired on one interface than the other. But it suffers from reliability problems, because the differences between the interfaces may be swamped by the innate differences between users. As a result, you need more repetition – more users – to get reliable and significant results from a between subjects design.

A **within-subjects** design is blocked by user. Each user sees both interfaces, and the differential performance (performance on X – performance on Y) of all users is averaged and compared with 0. This design eliminates variation due to user differences, but may have reliability problems due to ordering effects.

Which design is better? User differences cause much more variation than ordering effects, so the between-subjects design typically needs more users than the within-subjects design. But the between-subjects design may be more externally valid, because users in the real world will probably end up using only one interface (X or Y), not both.



## Counterbalancing

- Defeats ordering effects by varying order of conditions systematically (not randomly)
- Latin Square designs
  - randomly assign subjects to equal-size groups
  - A,B,C,... are the experimental conditions
  - Latin Square ensures that each condition occurs in every position in the ordering for an equal number of users

G1	G2
A	B
B	A

2x2

G1	G2	G3
A	C	B
B	A	C
C	B	A

3x3

G1	G2	G3	G4
A	D	C	B
B	A	D	C
C	B	A	D
D	C	B	A

4x4

... etc.

Spring 2012

6.813/6.831 User Interface Design and Implementation

21

Within-subjects designs suffer from ordering effects (particularly learning, which makes people get better, and fatigue, which makes them do worse). Randomizing the order of tasks and experimental conditions is one way to deal with these effects. Another way, particularly when the number of users is small, is **counterbalancing**, which ensures that every experimental condition occurs the same number of times at each position in the order. You counterbalance your experiment with the help of a **Latin square**, which is an  $N \times N$  matrix with the property that a symbol occurs exactly once in each row and each column. A few Latin squares are shown here; it's not hard to recognize the pattern and reproduce it for higher  $N$ .

To use counterbalancing, you first determine  $N$ , the number of experimental conditions you have, which is done by taking the product of the different values of each independent variable you are using. For example, the Windows vs. Mac menubar experiment has one independent variable with two values, hence two experimental conditions. If we also decided to test how the speed of access varied with starting distance from the menu (say, 50 pixels, 500 pixels, and 1000 pixels), then we'd have 6 experimental conditions (2 menubars  $\times$  3 distances). Given the number of conditions  $N$ , we divide the users randomly into  $N$  equal groups (G1...GN), and present each group with the conditions in the order of a different column from the Latin square.

Note that it's important for the number of users to be a multiple of  $N$ , so that the groups are equal in size. Otherwise the conditions won't occur the same number of times at each position in the order.

The simple Latin squares shown here have a flaw – *pairwise* learning effects are not controlled. In the 3x3 matrix, for example, B appears after A two-thirds of the time. In the 4x4 matrix, B follows A three-quarters of the time. So high performance on B may be due to practice on A, rather than inherent to B. This problem can be fixed by a **balanced Latin square**, left as an exercise for the reader. (Or see I. Scott MacKenzie, "Research Note: Within-subjects vs. Between-subjects Designs: Which to Use?", 2008, <http://www.yorku.ca/mack/RN-Counterbalancing.html>).



## Example: Ephemeral Adaptation Paper

### *Design*

A 2-factor mixed design was used: adaptive accuracy (Low or High) was a between-subjects factor and menu type (Control, Short-Onset or Long-Onset) was a within-subjects factor. Order of presentation was fully counterbalanced and participants were randomly assigned to conditions.

## Kinds of Measures

- Self-report
- Observation
  - Visible observer
  - Hidden observer
- Archival records
  - Public records
  - Private records
- Trace

Spring 2012

6.813/6.831 User Interface Design and Implementation

23

One thing we've glossed over a bit is how to **measure** the dependent variables we're interested in. Different measurement approaches have different kinds of noise, and some have biases induced by the user's awareness that their behavior is being studied. These biases are generally called **reactivity effects** by the social science community. Here's a simple taxonomy from the McGrath paper ("Methodology Matters", cited earlier), ranked by the obtrusiveness (and hence susceptibility to reactivity) of the method.

**Self-report** means asking the user to provide the data. Examples might be "How many times a day do you check email?" or "How much did you like this interface?" Self-reports are applicable to many different questions, and very cheap to collect. They are also noisy (particularly where quantitative data is requested), and may be biased by a variety of reactive effects, like politeness (trying to please the experimenter) and social desirability (saying what they think they *should* say, rather than the truth). Surveys generally exclusively use self-reported measures, though carefully-designed surveys can eliminate some of the biases. Lab experiments must resort to self-reports for some variables, particularly satisfaction, but better alternatives can often be found.

**Observation** means either the experimenter or an instrument (stopwatch, logging, screen capture, camera) is capturing the data. Observation is more expensive, but also more objective and controlled. A key distinction is whether the observation is visible (known to the subject) or hidden. Visible observation can produce distorting effects, like the classic "Hawthorne effect", in which people perform better simply because they know they're being studied. Hidden observation raises ethical questions of informed consent. Usually in lab experiments, the dilemma is resolved by using visible observation (or at least informed consent about observation), but using it on *all* conditions, and making it as unobtrusive as possible so that users stop thinking about it.

The last two measures are not used in lab experiments, because they use data that was not created solely for the purpose of the experiment, so they cannot be controlled. **Archival records** are records of past behavior, not made expressly for the purpose of the current research. Like observation, archival records can be distinguished by whether the user was aware that the records might be studied or read by another person (public) or not (private). Comments on a blog are public archival records; files on your hard drive are private. Finally, **traces** are laid down by behavior without the people involved even being *aware* that they are leaving something behind. The classic example of a trace is "read wear", the dogears and damage to books that indicate where and how they've been used. Web server access logs probably also fall into this category, since the vast majority of users are probably unaware that their page visits are being recorded.

## Triangulation

- Any given research method has advantages and limitations
  - lab experiment not externally valid
  - field study not controlled
  - survey biased by self-report
- For higher confidence, researchers **triangulate**
  - try several different methods to attack the same question; see if they concur or conflict
- Triangulation is rarely seen in a single HCI paper
  - More important that triangulation be happening **across the field**
  - encourage a diversity of research methods from different researchers aimed at the same question

Spring 2012

6.813/6.831 User Interface Design and Implementation

24

Let's conclude the lecture by returning to the main kinds of research methods in HCI (lab experiments, field studies, and surveys). We're focusing on lab experiments in this course, because there's much to say about them that isn't covered by any other course in the EECS curriculum. But frankly, each of these methods has advantages and disadvantages. Lab experiments are abstract and obtrusive, and may not be representative of the real world. Field studies can't be controlled, so it's hard to make strong, precise claims about comparative usability. Surveys are biased by reactivity.

So a research program that depends entirely on only one of these methods as a source of data and validation is likely to be biased and incomplete. One way to deal with this problem is by **triangulation** – using multiple methods (usually at least three, so that they can vote) to tackle the same research question. If they all support your claim, then you have much stronger evidence, without as many biases.

Triangulation is rarely seen in a single paper, but is not uncommon in a single researcher's whole research program (the series of papers generated by their work), and is frankly essential for the HCI field as a whole. A diversity of research methods is necessary for producing reliable knowledge. There's an active debate right now in the HCI community about this question, because of a sense that the flagship conference (CHI) has become dominated by papers using controlled lab experiments.

## Summary

- Research methods in HCI include lab experiments, field studies, and surveys
- Controlled experiments manipulate independent variables and measure dependent variables
- Must consider and defend against threats to internal validity, external validity, and reliability
- Blocking, randomization, and counterbalancing techniques help



(Day 2 of 3 of optional Experiment Analysis classes)  
 Today + Fri Walker testing

---

VI Hall Fano/shamp The Password Engine

---

Very dense VI

- don't know where to start

Must explicitly push save

- no expectation of saving on mobile

Make it simpler!

---

NanoQuiz (I don't need to do)

Test for # errors

formative - what doing today

qualitative, not quantitative

field survey

More concrete

closer to real world

harder to measure exact errors

small scale

② Survey - non exact since self reported

lab studies - precise as you can get

Deciding between subjects + within subjects

Pop validity (X) No external validity

learning effects (✓)

individual diff (✓) - than within subjects is better

fatigue (✓) order effect

Between-subject study in order they enter

reliability - Mes b/c internal validity

External validity (X)

internal validity (✓) people come in biased orders

---

Today's Topics Hypothesis testing

Graphing + error bars

T-test

ANOVA test

(3)

So same experiment as last time

We have some data now

Data does not look conclusive

1st = Find mean  
- Summarize statistic

2nd Apply a Statistical test

t-test - are the two means different?

ANOVA - are three or more means different?

3rd Tests produces a p value

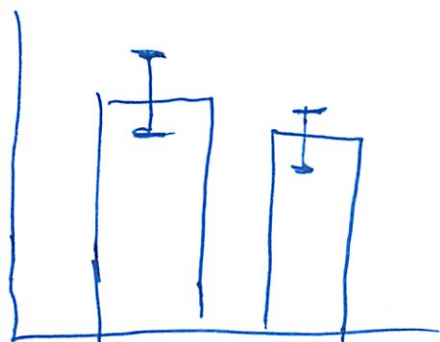
- So when repeat experiment it will be same

\* Prob that result we got is by chance  
    { we were fooled

So if  $p < .05$  are 95% confident

(4)

First plot error bars (Confidence interval  $\pm$  standard error of mean)  
- if bars overlap  $\rightarrow$  can't conclude <sup>related to st dev</sup>  
(I should research)



We have a weak estimate of the true mean  
<sup>if  $n = \text{world pop}$</sup>   
Confidence interval our mean is inside true mean

Hc recommends R

- stat package
  - ~~more~~ quirky language
  - can compute st error, mean, sd
- $$\frac{sd}{\sqrt{\#rows}}$$



⑤ Also hard to claim they are exactly the same  
I think he  
m's got my q

Box plots show 25th, 75th percentile

Not a substitute for stats for publishing

---

### Statistical Tests

- "Alternative Hypothesis"  $H_1$
- Null Hyp  $\text{mean}(\text{win}) = \text{mean}(\text{mac})$   $H_0$ 
  - non publishable
  - since its default state of world absent other into

- We can't really disprove null hyp

Instead we argue that chance of seeing  $p$   
diff is at least as @xtrem is very small  
if null hyp is true.

^ This is why it should be faster  
Lie Fitts Law

⑥

## Statistical Significance

$$X = \text{mean (win)} - \text{mean (loss)}$$

$$P(X = x \mid H_0)$$

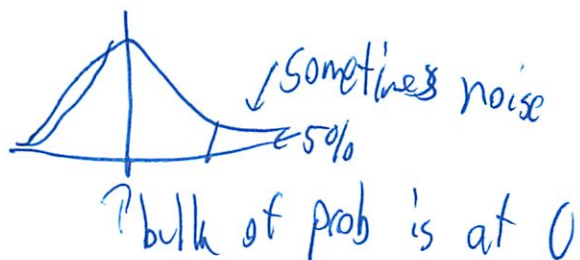
? prob if we assume  $H_0$  is true

Measure prob of getting same or greater difference

$$P(X \geq x_0 \mid H_0) \quad \text{one sided test}$$

$$2P(X \geq |x_0| \mid H_0) \quad \text{2 sided test}$$

Basically if  $H_0$  was true



If prob that null hyp is true is <sup>so</sup> low (25%)  
that it can't be true

① Run a t-test

↳ its essentially the difference /  $\sigma$

n affects degrees of freedom

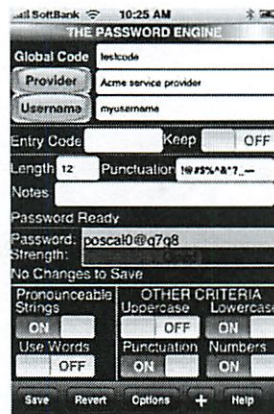
p-value - prob null hyp correct despite experiment

## **L14: Experiment Analysis**

- Today & Friday 3-5 pm prototype testing & RS2 testing (Walker)



## UI Hall of Fame or Shame?



Spring 2011

6.813/6.831 User Interface Design and Implementation

2

This is Password Engine, an iPhone app (<http://appshopper.com/utilities/password-engine>). Its purpose is to generate passwords

This UI has some simplicity challenges. The screen is packed and complex. Are all these features necessary? Do we need “Other Criteria”? The very name suggests unimportance. Why are these features on the main screen, instead of bundled off to an Options screen? And once you bundle a feature off to Options, where many users won’t even find it, it’s important to ask whether you need it at all. Aggressive removal of features that are unnecessary to the user’s task is a great way to improve usability.

The UI also has some learnability challenges. What should be done first? There’s no natural ordering in a screen as complex as this.

What about the Save and Revert buttons? This app has taken the interesting approach that the user has to explicitly save any changes to options—and yet it isn’t using the conventional dialog pattern to do that (i.e., a temporary window with OK/Cancel buttons). This is likely to lead to lapses on the part of the user – failures to remember to save – so the app goes to extra effort to make changes visible (see the message “No Changes to Save”). Unfortunately this message is hardly salient – it’s unlikely to be in the user’s attention. Fewer things on the screen would make it more salient; but even better would be either automatic save or a less error-prone dialog structure.

## Today's Topics

- Hypothesis testing
- Graphing with error bars
- T test
- ANOVA test

Spring 2011

6.813/6.831 User Interface Design and Implementation

5

This lecture continues the stream on research methods. Our last lecture in the stream concerned experiment design -- how to design controlled experiments to answer a research question. Today's lecture is about the second part of that process, how to analyze the data from the experiment to determine the answer to the question. We'll discuss the principles of **hypothesis testing**, which is the basis for analysis. We'll talk about a cheap and easy way to get a feel for your data, by graphing it with error bars, which is not hypothesis testing but is always good practice to do anyway. And we'll discuss two statistical tests commonly used in HCI research: the t test and the ANOVA (Analysis of Variance) test.

This is only a very brief introduction to statistical methods and experiment analysis. There's much more to be said on this topic, which is outside the scope of this class. There are other good MIT classes that cover it in much more depth, particularly 9.07 Statistical Methods in Brain & Cognitive Sciences and 16.470/ESD.756 Statistical Methods in Experimental Design. Also see <http://statistics.mit.edu/>, a clearinghouse site for classes and research in statistics at MIT.

## Experiment Analysis

- Hypothesis: Mac menubar is faster to access than Windows menubar
  - Design: between-subjects, randomized assignment of interface to subject

Windows	Mac
625	647
480	503
621	559
633	586

Spring 2011

6.813/6.831 User Interface Design and Implementation

6

Let's return to the example we used in the experiment design lecture. Suppose we've conducted an experiment to compare the position of the Mac menubar (flush against the top of the screen) with the Windows menubar (separated from the top by a window title bar).

For the moment, let's suppose we used a **between-subjects** design. We recruited users, and each user used only one version of the menu bar, and we'll be comparing different users' times. For simplicity, each user did only one trial, clicking on the menu bar just once while we timed their speed of access. (Doing only one trial is a very unreliable experiment design, and an expensive way to use people, but we'll keep it simple for the moment.)

The results of the experiment are shown above (times in milliseconds; note that this is fake, randomly-generated data, and the actual experiment data probably wouldn't look like this). Mac *seems* to be faster (574 ms on average) than Windows (590 ms). But given the **noise** in the measurements – some of the Mac trials are actually much slower than some of the Windows trials -- how do we know whether the Mac menubar is really faster?

This is the fundamental question underlying statistical analysis: estimating the amount of evidence in support of our hypothesis, even in the presence of noise.

## Statistical Testing

- Compute a statistic summarizing the experimental data
  - `mean(Win)`
  - `mean(Mac)`
- Apply a statistical test
  - *t* test: are two means different?
  - ANOVA (ANalysis Of VAriance): are three or more means different?
- Test produces a *p* value
  - *p* value = probability that the observed difference happened purely by chance
  - If  $p < 0.05$ , then we are 95% confident that there is a difference between Windows and Mac

Spring 2011

6.813/6.831 User Interface Design and Implementation

7

Here's the basic process we follow to determine whether the measurements we made support the hypothesis or not.

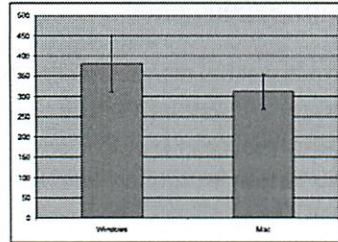
We summarize the data with a **statistic** (which, by definition, is a function computed from a set of data samples). A common statistic is the mean of the data, but it's not necessarily the only useful one. Depending on what property of the process we're interested in measuring, we may also compute the variance (or standard deviation), or median, or mode (i.e., the most frequent value). Some researchers argue that for human behavior, the median is a better statistic than the mean, because the mean is far more distorted by outliers (people who are very slow or very fast, for example) than the median.

Then we apply a **statistical test** that tells us whether the statistics support our hypothesis. Two common tests for means are the **t test** (which asks whether the mean of one condition is different from the mean of another condition) and **ANOVA** (which asks the same question when we have the means of three or more conditions).

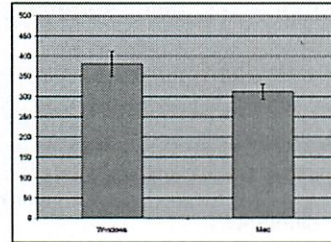
The statistical test produces a **p value**, which is the probability that the difference in statistics that we observed happened purely by chance. Every run of an experiment has random noise; the *p* value is basically the probability that the means were different *only* because of these random factors. Thus, if the *p* value is less than 0.05, then we have a 95% confidence that there really is a difference. (There's a more precise meaning for this, which we'll get to in a bit.)



## Standard Error of the Mean



N = 4 :  
Error bars overlap, so can't  
conclude anything



N=10:  
Error bars are disjoint, so  
Windows may be different  
from Mac

Spring 2011

6.813/6.831 User Interface Design and Implementation

8

Let's talk about a simple, rough method for judging whether an experiment might support its hypothesis or not, if the statistics you're using are **means**.

The **standard error of the mean** is a statistic that measures how close the mean statistic you computed is likely to be to the true mean. The standard error is computed by taking the standard deviation of the measurements and dividing by the square root of  $n$ , the number of measurements. (This is derived from the Central Limit Theorem of probability theory: that the sum of  $N$  samples from a distribution with mean  $\mu$  and variance  $V$  has a probability distribution that approaches a *normal* distribution, i.e. a bell curve, whose mean is  $N\mu$  and whose variance is  $NV$ . Thus, the *average* of the  $N$  samples would have a normal distribution with mean  $\mu$  and variance  $V/n$ . Its standard deviation would be  $\sqrt{V/N}$ , or equivalently, the standard deviation of the underlying distribution divided by  $\sqrt{n}$ .)

The standard error is like a region of likelihood around the computed mean – the region around the computed mean in which the *true* mean of the process probably lies. Think of the computed mean as a random selection from a normal distribution (bell curve) around the true mean; it's randomized because of all the uncontrolled variables and intentional randomization that you did in your experiment. With a normal distribution, 68% of the time your random sample will be within  $\pm 1$  standard deviation of the mean; 95% of the time it will be within  $\pm 2$  standard deviations of the mean. The standard error is the standard deviation of the mean's normal distribution, so what this means is that if we draw an **error bar** one standard error above our computed mean, and one standard error below our computed mean, then that interval will have the true mean in it 68% of the time. It is therefore a 68% confidence interval for the mean.

To use the standard error technique, draw a bar chart of the means for each condition, with error bars (whiskers) stretching 1 standard error above and below the top of each bar. If we look at whether those error whiskers **overlap** or are substantially different, then we can make a rough judgement about whether the true means of those conditions are likely to be different. Suppose the error bars overlap – then it's possible that the true means for both conditions are actually the *same* – in other words, that whether you use the Windows or Mac menubar design makes no difference to the speed of access. But if the error bars do *not* overlap, then it's likely that the true means are different.

The error bars can also give you a sense of the reliability of your experiment, also called the **statistical power**. If you didn't take enough samples – too few users, or too few trials per user – then your error bars will be large relative to the size of the data. So the error bars may overlap even though there really is a difference between the conditions. The solution is more repetition – more trials and/or more users – in order to increase the reliability of the experiment.

## Quick Intro to R

- R is an open source programming environment for data manipulation
  - includes statistics & charting
- Get the data in

```
data1 = read.csv(file.choose())
```
- Compute with it

```
means = mean(data1)
stderrs = sd(data1)/sqrt(nrow(data1))
```
- Graph it

```
x = barplot(means, ylim=c(0,800))
arrows(x, means-stderrs, x, means+stderrs, code=3, angle=90, length=.1)
```

Spring 2011

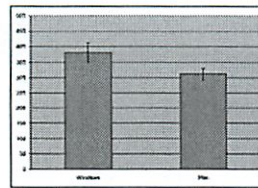
6.813/6.831 User Interface Design and Implementation

9

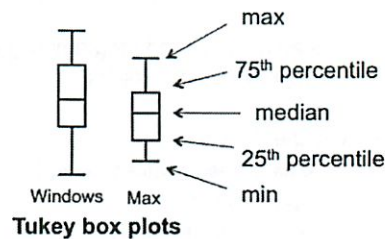
R is a good choice for a lot of statistical processing, because it's free and very powerful. A good introduction to R is online at <http://cran.r-project.org/doc/manuals/R-intro.html>.

Here's how you can use R to create simple bar charts with error bars.

## Graphing Techniques



Error bars



Tukey box plots

- Pros
  - Easy to compute
  - Give a feel for your data
- Cons
  - Not a substitute for statistical testing

Spring 2011

6.813/6.831 User Interface Design and Implementation

10

Plotting your data is the first step you should do after every experiment, to eyeball your results and judge whether statistical testing is worthwhile, or whether you need more data. It's said that John Tukey, the Stanford statistician who gave his name to one of the statistical tests we'll be talking about, refused to help anybody who hadn't first gone off and plotted their data on a big piece of paper. Tukey's excellent book *Exploratory Data Analysis* introduced the notion of a "box plot" (shown here on the right) which is even richer than a simple bar with error whiskers, showing 5 useful statistics about the spread of each data set in a single graphic. Don't discount the value of your perceptual system for detecting patterns and really *appreciating* the size of effects.

If you want to publish the results of your experiment, you'll need to do some statistical tests as well, like the t tests or ANOVAs we'll talk about in the rest of this lecture. But your paper should still have plots with error bars in it. Some researchers even argue that the error-bar plots are more valuable and persuasive than the statistical tests (G.R. Loftus, "A picture is worth a thousand p values: On the irrelevance of hypothesis testing in the microcomputer age," *Behavior Research Methods, Instruments & Computers*, 1993, 25(2), 250–256), though this view is far from universally held.

Be warned that nonoverlapping error bars is only a rough indicator; it does *not* imply a statistically significant difference (i.e.,  $p < 0.05$ ). For that, you have to actually do the t test or ANOVA test, which is what we'll turn to now. (For more explanation, see Harvey Motulsky, "The link between error bars and statistical significance", <http://www.graphpad.com/articles/errorbars.htm>, 2002; and Dave Munger, "Most researchers don't understand error bars," [http://scienceblogs.com/cognitivedaily/2008/07/most\\_researchers\\_dont\\_understa\\_1.php#more](http://scienceblogs.com/cognitivedaily/2008/07/most_researchers_dont_understa_1.php#more), March 2007.)



## Hypothesis Testing

- Our hypothesis: position of menubar matters
  - i.e.,  $\text{mean}(\text{Mac times}) < \text{mean}(\text{Windows times})$
  - This is called the alternative hypothesis (also called H1)
- If we're wrong: position of menu bar makes no difference
  - i.e.,  $\text{mean}(\text{Mac}) = \text{mean}(\text{Win})$
  - This is called the null hypothesis (H0)
- We can't really disprove the null hypothesis
  - Instead, we argue that the chance of seeing a difference **at least as extreme** as what we saw is very small if the null hypothesis is true

Spring 2011

6.813/6.831 User Interface Design and Implementation

11

Our hypothesis is that the position of the menubar makes a difference in time. Another way of putting it is that the (noisy) process that produced the Mac access times is **different** from the process that produced the Windows access times. Let's make the hypothesis very specific: that the mean access time for the Mac menu bar is less than the mean access time for the Windows menu bar.

In the presence of randomness, however, we can't really *prove* our hypothesis. Instead, we can only present evidence that it's the best conclusion to draw from all possible other explanations. We have to argue against a skeptic who claims that we're wrong. In this case, the skeptic's position is that the position of the menu bar makes *no* difference; i.e., that the process producing Mac access times and Windows access times is the same process, and in particular that the mean Mac time is equal to the mean Windows time. This hypothesis is called the **null hypothesis**. In a sense, the null hypothesis is the "default" state of the world; our own hypothesis is called the **alternative hypothesis**.

Our goal in hypothesis testing will be to accumulate enough evidence – enough of a difference between Mac times and Windows times – so that we can **reject the null hypothesis** as very unlikely.



## Statistical Significance

- Compute a statistic from our experimental data  
 $X = \text{mean}(\text{Win}) - \text{mean}(\text{Mac})$
- Determine the probability distribution of the statistic assuming  $H_0$  is true  
 $\Pr(X=x \mid H_0)$
- Measure the probability of getting the same or greater difference  
 $\Pr(X > x_0 \mid H_0)$  *one-sided test*  
 $2 \Pr(X > |x_0| \mid H_0)$  *two-sided test*
- If that probability is less than 5%, then we say
  - “We reject the null hypothesis at the 5% significance level”
  - equivalently: “difference between menubars is statistically significant ( $p < .05$ )”
- Statistically significant does not mean scientifically important

Spring 2011

6.813/6.831 User Interface Design and Implementation

12

Here’s the basic idea behind statistical testing. We boil all our experimental data down to a single statistic (in this case, we’d want to use the difference between the average Mac time and the average Windows time). If the null hypothesis is true, then this statistic has a certain probability distribution. (In this case, if  $H_0$  is true and there’s no difference between Windows and Mac menu bars, then our difference in averages should be distributed around 0, with some standard deviation).

So if  $H_0$  is really true, we can regard our entire experiment as a single random draw from that distribution. If the statistic we computed turned out to be a typical value for the  $H_0$  distribution – really near 0, for example – then we don’t have much evidence for arguing that  $H_0$  is false. But if the statistic is extreme – far from 0 in this case – then we can **quantify** the likelihood of getting such an extreme result. If only 5% of experiments would produce a result that’s at least as extreme, then we say that we reject the null hypothesis – and hence accept the alternative hypothesis  $H_1$ , which is the one we wanted to prove – at the 5% significance level.

The probability of getting at least as extreme a result given  $H_0$  is called the **p value** of the experiment. Small p values are better, because they measure the likelihood of the null hypothesis. Conventionally, the p value must be 5% to be considered **statistically significant**, i.e. enough evidence to reject. But this convention depends on context. An experiment with very few trials ( $n < 10$ ) may be persuasive even if its p value is only 10%. (Note that a paper reviewer would expect you to have a good reason for running so few trials that the standard 5% significance wasn’t enough...) Conversely, an experiment with thousands of trials won’t be terribly convincing unless its p value is 1% or less.

Keep in mind that **statistical significance does not imply importance**. Suppose the difference between the Mac menu bar and Windows menu bar amounted to only 1 millisecond (out of hundreds of milliseconds of total movement time). A sufficiently large experiment, with enough trials, would be able to detect this difference at the 5% significance level, but the difference is so small that it simply wouldn’t be relevant to user interface design.

## T test

- T test compares the means of two samples A and B
- Two-sided:
  - $H_0: \text{mean}(A) = \text{mean}(B)$
  - $H_1: \text{mean}(A) \neq \text{mean}(B)$
- One-sided:
  - $H_0: \text{mean}(A) = \text{mean}(B)$
  - $H_1: \text{mean}(A) < \text{mean}(B)$
- Assumptions:
  - samples A & B are independent (between-subjects, randomized)
  - normal distribution
  - equal variance

Spring 2011

6.813/6.831 User Interface Design and Implementation

13

Let's look at some of the more common statistical tests that are used in user interface experiments.

The T test is what you'd use to compare two means in a between-subjects experiment, like the hypothetical Mac/Windows menubar experiment we've been discussing. The T statistic computes the difference between the Mac average and the Windows average, divided by an estimate of the standard deviation. If the null hypothesis is true, then this statistic follows a T distribution (which looks very similar to a normal distribution, a hump centered at 0). You can look up the value of the T statistic you computed in a table of the T distribution to find out the probability of getting a more extreme value.

There are two forms of the T test, with different alternative hypotheses. In the more conservative, **two-sided** T test, your alternative hypothesis is merely that the means are different, so an extreme t value (either positive or negative) counts as evidence against the null hypothesis. The other form is the **one-sided** test, in which your alternative hypothesis expects the difference to go one way or the other – e.g., if there's any difference between Mac and Windows at all, the Mac should be faster. It's conventional to use the two-sided test unless you (and the skeptic you're arguing against) are completely certain which way the difference should go, if the difference exists at all.

Using the T test requires a few assumptions. First, your samples should be independent, so you need to use good experiment design with randomization and controls to prevent inadvertent dependence between samples. Second, the T test also assumes that the underlying probability distribution of the samples (e.g., the access times) is a normal distribution, and that even if the alternative hypothesis is true, both samples have equal variance. Fortunately the T test is not too sensitive to the normality and equal-variance assumptions: if your sample is large enough ( $N > 20$ ), deviations don't affect it much. There's also an "unequal variances" version of the T test, which uses a slightly different statistic, but with weaker assumptions come less power (i.e., it takes a larger N to reach the same level of significance with the unequal variances T test).



## Running a T Test

- `t.test(data1$win, data1$mac)`

Welch Two Sample t-test

```
data: win and mac
t = 2.1322, df = 17.623, p-value = 0.04733
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.9992905 151.0007095
sample estimates:
mean of x mean of y
    584      508
```

Spring 2011

6.813/6.831 User Interface Design and Implementation

14

The actual calculation of the t statistic and the proof that it's a valid test are beyond the scope of this course; the statistics courses mentioned earlier cover it. In practice, nobody computes their own t statistic; they use a statistical package to do the computations. So for the sake of this class, we'll focus on understanding **when to use** a t test (that was the last slide), and then **how to read** the report of the test produced by a stats package.

Running a t test in R to compare two conditions, represented as vectors of numbers, is almost trivial. For more information, look at <http://www.statmethods.net/stats/ttest.html>

Here's the result of applying the t test (assuming equal variances) to the 10-observation Windows/Mac menubar experiment we've been using.

The most important numbers are highlighted in yellow. **T** is the actual t statistic value computed from your data. **df (degrees of freedom)** is a measure of the power of the test; it's directly related to the number of observations you have (n-2 in the case of the t test, but other statistical tests have different ways to calculate their degrees of freedom).

Finally, the **p value** for the t test is 0.047, which means that the observed difference between the Windows and Mac menubar is only 4.7% likely to happen purely by chance. Assuming we decided to use a 5% significance level from the outset, we would now say that the difference is **statistically significant** (two-tailed  $t = 2.13$ ,  $df = 18$ ,  $p < 0.05$ ). Often researchers will write just " $p < 0.05$ " or " $p < 0.01$ " instead of giving the actual p value.

## Summary

- Use statistical tests to establish significance of observed differences
- Graphing with error bars is cheap and easy, and great for getting a feel for data



6.813 L15

3/16

GR3 due Sun

-  $\geq 6$  users

- can still prototype after that

## Experimental Analysis

MS Office

Live preview

Hall Fane/Shamp

- quickly preview and back track  
- but jarring

## Nanoquiz

If error bars don't overlap - good  
but not provable

Null hyp - skeptic might assume  
- trying to disprove

Added data might  $\uparrow \downarrow$  mean

but will likely  $\downarrow$  st error since  $\sqrt{\# \text{ empts}}$   
and  $\downarrow$  p-value of stat test

~~revenue~~ in lots of people

- even small differences w/ very low ps

② Mac vs Windows title bar

So ~~the~~  $H_0$  no difference

$$\text{mean } w = \text{mean } m$$

$H_1$  alt hyp

$$1. \text{ mean } w \neq \text{mean } m$$

↳ (two tail)

(or) (from + or - side dist)  
more conservative, twice the p value

$$2. \text{ mean } w > \text{mean } m$$

↳ Since Fitts tells us that  
(one tailed)

T-statistic

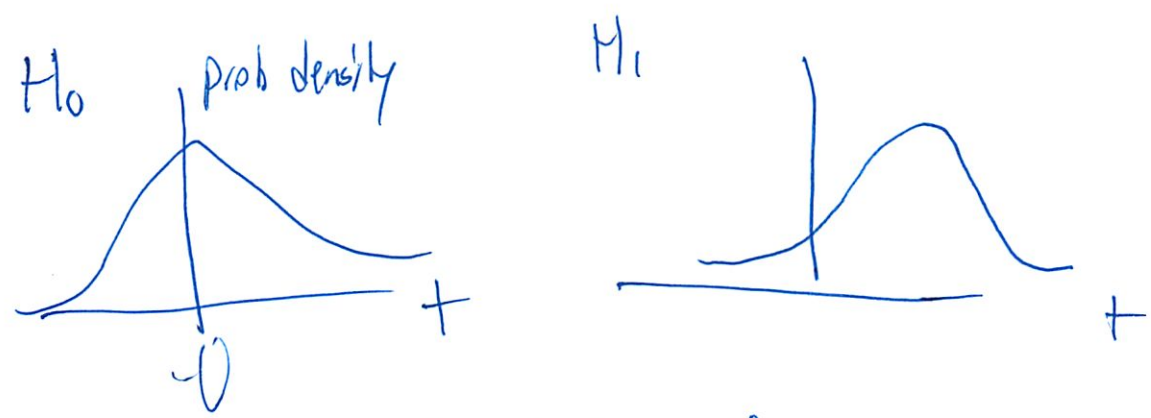
Diff of 2 means divided by normalization

$$t = \frac{\text{mean } w - \text{mean } m}{\sigma / \sqrt{n}}$$

Fitts who size testing

will have one prob dist if  $H_0$   
diff " " if  $H_1$

(3)



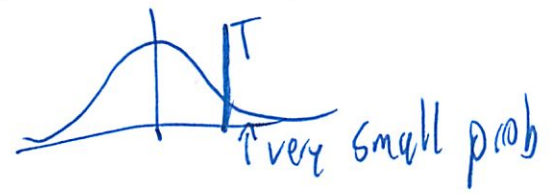
↑ skeptic says is centered on 0

large  $t$ -stats good  $\rightarrow$  big difference

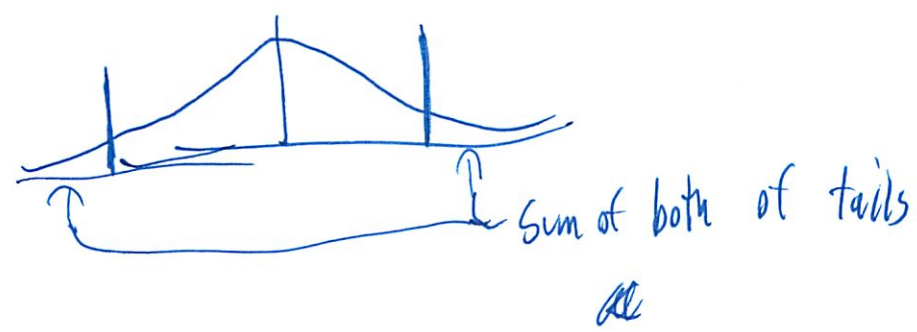
Can calc p-dist of possibility of

Still only have one  $t$ -stat value

Then prob of having a large  $t$  value is small



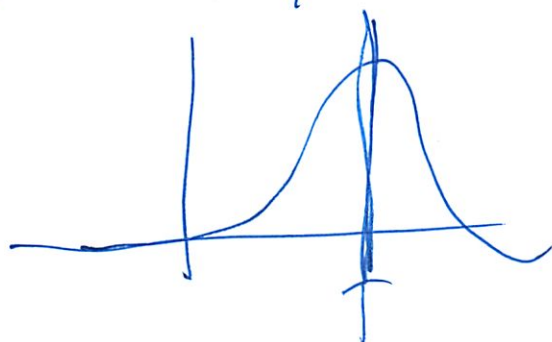
When two tailed



If  $< 5\%$  - tell skeptic - only 5% chance you are right

④

It's far more likely it was drawn from this dist



Alt hyp can't be close to center

Skeptics does not want them to count

## T-Test

### One Sided

$$H_0 \text{ mean}(A) = \text{mean}(B)$$

$$H_1 \text{ mean}(A) < \text{mean}(B)$$

### Two Sided

$$H_0 \text{ mean}(A) = \text{mean}(B)$$

$$H_1 \text{ mean}(A) \neq \text{mean}(B)$$

Assumes samples ind

and are ~~of~~ = variance

(can be a ~~different~~ weaker version) or  
in alt t-test



5

~~10~~ 10  
# degrees of freedom = (df)  
↓  
↓ another 10 degrees

fuzzy + loose  
actually is much  
more precise

$$t = \frac{\text{mean}_w - \text{mean}_w}{\sqrt{n}}$$

not from data pts  
subtracts  
2 degrees  
of freedom

t will be smaller w/ less #s  
df smaller as well

hard to make strong argument against critic

---

Factors

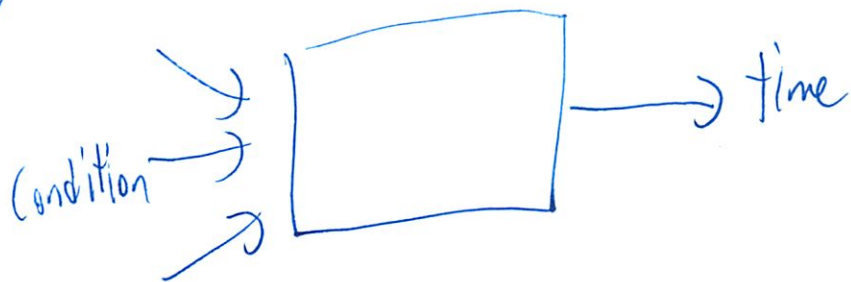
better to have 1 table w/ 1 col of the condition

R auto treats strings as categories

Can then run a T-test on that

↳ testing a model of the process

(e)



Some  $H_s$  (just negative since thing flipped)

---

## Paired t-test

within subject studies

difference of users w/ themselves

Tell R paired = TRUE

---

## ANOVA (Variance)

Compares more than 2 means ( $k \geq 2$ )

$H_0 \rightarrow$  all  $k$  means are =

$H_1 \rightarrow$  the means are diff (so ind variables matter)

between If  $F$  high ( $> 1$ )  $\Rightarrow$  stat. sig. result

within Add a col w/ user who did trial  
Include it as a special error term

⑦

We've shown at least 1 cond. diff than  
the others - but not which one

So Tukey HSD

- can get p-value b/w comparisons

98%  $\rightarrow$  almost sure

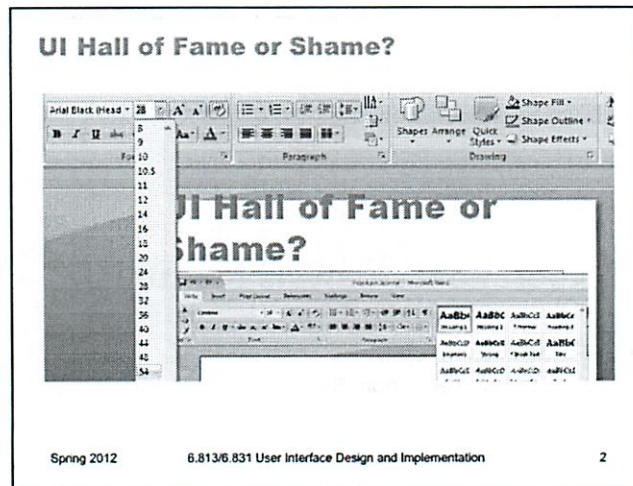
2%  $\rightarrow$  border line significant

( $\hat{=}$ ) check

## **L15: Experiment Analysis (continued)**

- GR3 due Sun
- RS2 due two weeks from Sun





Today's Hall of Fame or Shame example is a feature of Microsoft Office 2007 that gives a preview of what a style command will do to the document while you're mousing over it. Here, the mouse is hovering over the 54-point choice on the font-size drop-down, and PowerPoint is showing what the selection would look like with that new style.

Let's discuss the pros and cons of this approach from a usability point of view.

## Today's Topics

- T test
- ANOVA test

Spring 2012

6.813/6.831 User Interface Design and Implementation

5

This lecture continues the stream on research methods. Our last lecture in the stream concerned experiment design -- how to design controlled experiments to answer a research question. Today's lecture is about the second part of that process, how to analyze the data from the experiment to determine the answer to the question. We'll discuss the principles of **hypothesis testing**, which is the basis for analysis. We'll talk about a cheap and easy way to get a feel for your data, by graphing it with error bars, which is not hypothesis testing but is always good practice to do anyway. And we'll discuss two statistical tests commonly used in HCI research: the t test and the ANOVA (Analysis of Variance) test.

This is only a very brief introduction to statistical methods and experiment analysis. There's much more to be said on this topic, which is outside the scope of this class. There are other good MIT classes that cover it in much more depth, particularly 9.07 Statistical Methods in Brain & Cognitive Sciences and 16.470/ESD.756 Statistical Methods in Experimental Design. Also see <http://statistics.mit.edu/>, a clearinghouse site for classes and research in statistics at MIT.

## Experiment Analysis

- Hypothesis: Mac menubar is faster to access than Windows menubar
  - Design: between-subjects, randomized assignment of interface to subject

Windows	Mac
625	647
480	503
621	559
633	586

Spring 2012

6.813/6.831 User Interface Design and Implementation

6

Let's return to the example we used in the experiment design lecture. Suppose we've conducted an experiment to compare the position of the Mac menubar (flush against the top of the screen) with the Windows menubar (separated from the top by a window title bar).

For the moment, let's suppose we used a **between-subjects** design. We recruited users, and each user used only one version of the menu bar, and we'll be comparing different users' times. For simplicity, each user did only one trial, clicking on the menu bar just once while we timed their speed of access. (Doing only one trial is a very unreliable experiment design, and an expensive way to use people, but we'll keep it simple for the moment.)

The results of the experiment are shown above (times in milliseconds; note that this is fake, randomly-generated data, and the actual experiment data probably wouldn't look like this). Mac *seems* to be faster (574 ms on average) than Windows (590 ms). But given the **noise** in the measurements – some of the Mac trials are actually much slower than some of the Windows trials -- how do we know whether the Mac menubar is really faster?

This is the fundamental question underlying statistical analysis: estimating the amount of evidence in support of our hypothesis, even in the presence of noise.

## Hypothesis Testing

- Our hypothesis: position of menubar matters
  - i.e.,  $\text{mean}(\text{Mac times}) < \text{mean}(\text{Windows times})$
  - This is called the alternative hypothesis (also called H1)
- If we're wrong: position of menu bar makes no difference
  - i.e.,  $\text{mean}(\text{Mac}) = \text{mean}(\text{Win})$
  - This is called the null hypothesis (H0)
- We can't really disprove the null hypothesis
  - Instead, we argue that the chance of seeing a difference at **least as extreme** as what we saw is very small if the null hypothesis is true

Spring 2012

6.813/6.831 User Interface Design and Implementation

7

Our hypothesis is that the position of the menubar makes a difference in time. Another way of putting it is that the (noisy) process that produced the Mac access times is **different** from the process that produced the Windows access times. Let's make the hypothesis very specific: that the mean access time for the Mac menu bar is less than the mean access time for the Windows menu bar.

In the presence of randomness, however, we can't really *prove* our hypothesis. Instead, we can only present evidence that it's the best conclusion to draw from all possible other explanations. We have to argue against a skeptic who claims that we're wrong. In this case, the skeptic's position is that the position of the menu bar makes *no* difference; i.e., that the process producing Mac access times and Windows access times is the same process, and in particular that the mean Mac time is equal to the mean Windows time. This hypothesis is called the **null hypothesis**. In a sense, the null hypothesis is the "default" state of the world; our own hypothesis is called the **alternative hypothesis**.

Our goal in hypothesis testing will be to accumulate enough evidence – enough of a difference between Mac times and Windows times – so that we can **reject the null hypothesis** as very unlikely.



## Statistical Significance

- Compute a statistic from our experimental data  
 $X = \text{mean}(\text{Win}) - \text{mean}(\text{Mac})$
- Determine the probability distribution of the statistic assuming  $H_0$  is true  
 $\Pr(X=x \mid H_0)$
- Measure the probability of getting the same or greater difference  
 $\Pr(X > x_0 \mid H_0)$  *one-sided test*  
 $2 \Pr(X > |x_0| \mid H_0)$  *two-sided test*
- If that probability is less than 5%, then we say
  - “We reject the null hypothesis at the 5% significance level”
  - equivalently: “difference between menubars is statistically significant ( $p < .05$ )”
- Statistically significant does not mean scientifically important

Spring 2012

6.813/6.831 User Interface Design and Implementation

8

Here’s the basic idea behind statistical testing. We boil all our experimental data down to a single statistic (in this case, we’d want to use the difference between the average Mac time and the average Windows time). If the null hypothesis is true, then this statistic has a certain probability distribution. (In this case, if  $H_0$  is true and there’s no difference between Windows and Mac menu bars, then our difference in averages should be distributed around 0, with some standard deviation).

So if  $H_0$  is really true, we can regard our entire experiment as a single random draw from that distribution. If the statistic we computed turned out to be a typical value for the  $H_0$  distribution – really near 0, for example – then we don’t have much evidence for arguing that  $H_0$  is false. But if the statistic is extreme – far from 0 in this case – then we can **quantify** the likelihood of getting such an extreme result. If only 5% of experiments would produce a result that’s at least as extreme, then we say that we reject the null hypothesis – and hence accept the alternative hypothesis  $H_1$ , which is the one we wanted to prove – at the 5% significance level.

The probability of getting at least as extreme a result given  $H_0$  is called the **p value** of the experiment. Small p values are better, because they measure the likelihood of the null hypothesis. Conventionally, the p value must be 5% to be considered **statistically significant**, i.e. enough evidence to reject. But this convention depends on context. An experiment with very few trials ( $n < 10$ ) may be persuasive even if its p value is only 10%. (Note that a paper reviewer would expect you to have a good reason for running so few trials that the standard 5% significance wasn’t enough...) Conversely, an experiment with thousands of trials won’t be terribly convincing unless its p value is 1% or less.

Keep in mind that **statistical significance does not imply importance**. Suppose the difference between the Mac menu bar and Windows menu bar amounted to only 1 millisecond (out of hundreds of milliseconds of total movement time). A sufficiently large experiment, with enough trials, would be able to detect this difference at the 5% significance level, but the difference is so small that it simply wouldn’t be relevant to user interface design.

## T test

- T test compares the means of two samples A and B
- Two-sided:
  - $H_0: \text{mean}(A) = \text{mean}(B)$
  - $H_1: \text{mean}(A) \neq \text{mean}(B)$
- One-sided:
  - $H_0: \text{mean}(A) = \text{mean}(B)$
  - $H_1: \text{mean}(A) < \text{mean}(B)$
- Assumptions:
  - samples A & B are independent (between-subjects, randomized)
  - normal distribution
  - equal variance

Spring 2012

6.813/6.831 User Interface Design and Implementation

9

Let's look at some of the more common statistical tests that are used in user interface experiments.

The T test is what you'd use to compare two means in a between-subjects experiment, like the hypothetical Mac/Windows menubar experiment we've been discussing. The T statistic computes the difference between the Mac average and the Windows average, divided by an estimate of the standard deviation. If the null hypothesis is true, then this statistic follows a T distribution (which looks very similar to a normal distribution, a hump centered at 0). You can look up the value of the T statistic you computed in a table of the T distribution to find out the probability of getting a more extreme value.

There are two forms of the T test, with different alternative hypotheses. In the more conservative, **two-sided** T test, your alternative hypothesis is merely that the means are different, so an extreme t value (either positive or negative) counts as evidence against the null hypothesis. The other form is the **one-sided** test, in which your alternative hypothesis expects the difference to go one way or the other – e.g., if there's any difference between Mac and Windows at all, the Mac should be faster. It's conventional to use the two-sided test unless you (and the skeptic you're arguing against) are completely certain which way the difference should go, if the difference exists at all.

Using the T test requires a few assumptions. First, your samples should be independent, so you need to use good experiment design with randomization and controls to prevent inadvertent dependence between samples. Second, the T test also assumes that the underlying probability distribution of the samples (e.g., the access times) is a normal distribution, and that even if the alternative hypothesis is true, both samples have equal variance. Fortunately the T test is not too sensitive to the normality and equal-variance assumptions: if your sample is large enough ( $N > 20$ ), deviations don't affect it much. There's also an "unequal variances" version of the T test, which uses a slightly different statistic, but with weaker assumptions come less power (i.e., it takes a larger N to reach the same level of significance with the unequal variances T test).



## Running a T Test

- `t.test(data1$win, data1$mac)`

Welch Two Sample t-test

```
data: win and mac
t = 2.1322, df = 17.623, p-value = 0.04733
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.9992905 151.0007095
sample estimates:
mean of x mean of y
    584      508
```

Spring 2012

6.813/6.831 User Interface Design and Implementation

10

The actual calculation of the t statistic and the proof that it's a valid test are beyond the scope of this course; the statistics courses mentioned earlier cover it. In practice, nobody computes their own t statistic; they use a statistical package to do the computations. So for the sake of this class, we'll focus on understanding **when to use** a t test (that was the last slide), and then **how to read** the report of the test produced by a stats package.

Running a t test in R to compare two conditions, represented as vectors of numbers, is almost trivial. For more information, look at <http://www.statmethods.net/stats/ttest.html>

Here's the result of applying the t test (assuming equal variances) to the 10-observation Windows/Mac menubar experiment we've been using.

The most important numbers are highlighted in yellow. **T** is the actual t statistic value computed from your data. **df (degrees of freedom)** is a measure of the power of the test; it's directly related to the number of observations you have (n-2 in the case of the t test, but other statistical tests have different ways to calculate their degrees of freedom).

Finally, the **p value** for the t test is 0.047, which means that the observed difference between the Windows and Mac menubar is only 4.7% likely to happen purely by chance. Assuming we decided to use a 5% significance level from the outset, we would now say that the difference is **statistically significant** (two-tailed  $t = 2.13$ ,  $df = 18$ ,  $p < 0.05$ ). Often researchers will write just " $p < 0.05$ " or " $p < 0.01$ " instead of giving the actual p value.

## Running a T Test

- `smalldata = data1[1:4,]`
- `t.test(smalldata$win, smalldata$mac)`

Welch Two Sample t-test

```
data: smalldata$win and smalldata$mac
t = 0.3381, df = 5.767, p-value = 0.7473
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-100.9362  132.9362
sample estimates:
mean of x mean of y
 589.75   573.75
```

Spring 2012

6.813/6.831 User Interface Design and Implementation

11

Now let's look at the subset of the data we graphed earlier – just the first four observations for each condition. This will let us see an example of a failing statistical test.

In this case, the two-tailed t test had p value 0.75, which means that the difference in means between the Windows sample and the Mac sample was 75% likely to happen by pure chance even if the Windows and Mac conditions actually had the same true mean (the null hypothesis). That's way too high a chance, so we say that this data showed **no significant difference** between the Windows and Mac menubars (**two-tailed t=0.336, df=6, p = 0.75**). The part in parentheses is important when you're writing this result in a paper – it gives the type of statistical test used (two-tailed t), the actual value of the statistic (0.336), the degrees of freedom, and the resulting p value. (Many researchers would omit it for a failing test like this one, but it's essential to include it when the test succeeds).



## Using Factors in R

- Instead of representing the win/mac conditions as columns, it's better to represent them by a factor (categorical variable)
- `data2 = read.csv(file.choose())`
- `t.test(data2$time ~ data2$condition)`

```
Welch Two Sample t-test

data: all by allf
t = -2.1322, df = 17.623, p-value = 0.04733
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-151.0007095 -0.9992905
sample estimates:
mean in group mac mean in group win
508                    584
```

condition	time
win	625
win	480
win	621
win	633
win	694
win	599
win	505
win	527
win	651
win	505
mac	647
mac	503
mac	559
mac	586
mac	458
mac	380
mac	477
mac	409
mac	589
mac	472

Spring 2012

6.813/6.831 User Interface Design and Implementation

12

There's another way to run this t test in R, which we'll look at because it introduces an important concept that we'll need for more sophisticated tests in a bit: a **factor**. A factor is a vector of values of a categorical independent variable. In this case, the condition can be either *win* or *mac*, so we first construct a vector of strings (10 "win" and 10 "mac", matching the 20 measurements in the vector *time*), and then convert it from a vector of strings into a factor of enumerated values.

Once we've used a factor to identify the two groups in our t test, we can run the t test against an explicit **model** of the process. That's what *time ~ condition* means: that we believe that the dependent variable *time* is a function of the (two-valued variable) *condition*, and we want the t test to test this model against the null hypothesis that *time* is independent of *condition*.

## Paired T Test

- For within-subject experiments with two conditions
- Uses the mean of the differences (each user against themselves)
- $H_0: \text{mean}(A_i - B_i) = 0$
- $H_1: \text{mean}(A_i - B_i) \neq 0$  (two-sided test)  
or  $\text{mean}(A_i - B_i) > 0$  (one-sided test)

Spring 2012

6.813/6.831 User Interface Design and Implementation

13

What if we had run a **within-subjects** experiment instead? Then we would need to compare each subject with themselves, by computing the difference between each subject's Macintosh access time and the same subject's Windows access time. We would then use a t test for the hypothesis that the mean of these differences is nonzero, against the null hypothesis that the mean of the differences is zero. This test is called a **paired t test**.

Why is a paired t test more powerful? Because by computing the difference within each user, we're canceling out the contribution that's unique to the user. That means that individual differences between users are no longer contributing to the noise (variance) of the experiment.

## Running a Paired T Test (in R)

- `t.test(data2$times ~ data2$condition, paired=TRUE)`

```
data: win and mac
t = 2.6758, df = 9, p-value = 0.02538
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 11.74872 140.25128
sample estimates:
mean of the differences
              76
```

Spring 2012

6.813/6.831 User Interface Design and Implementation

14

Here's an analysis of a within-subjects menubar experiment. Each subject did a trial on each menubars (counterbalanced to control for ordering effects, so half the subjects used the Windows menubar first and half used the Mac menubar first). The data is ordered by subject, so subject #1's times were 625ms for the Windows menubar and 647ms for the Mac menubar. The t test is actually applied to the differences (e.g.,  $625 - 647 = -22$  for subject 1). The p value for the two-tailed t test is now 0.025, which means that the observed difference between the Windows and Mac menubar is only 2.5% likely to happen purely by chance. So we would be justified in concluding that the difference is statistically significant.

Note the `paired=TRUE` parameter to `t.test`; that's what makes R pair up the observations. See <http://www.statmethods.net/stats/ttest.html>

## Analysis of Variance (ANOVA)

- Compares more than 2 means
- One-way ANOVA
  - 1 independent variable with  $k \geq 2$  levels
  - $H_0$ : all  $k$  means are equal
  - $H_1$ : the means are different (so the independent variable matters)

Spring 2012

6.813/6.831 User Interface Design and Implementation

15

So far we've only looked at one independent variable (the menu bar position) with only two levels tested (Mac position and Windows position). If you want to test means when you have more than one independent variable, or more than two levels, you can use ANOVA (short for Analysis of Variance).

One-way ANOVA (also called "single factor ANOVA") addresses the case where you have more than two levels of the independent variable that you're testing. For example, suppose we wanted to test a third menu bar position at the bottom of the screen. Then we'd have three samples: top (Mac), below title (Windows), and bottom. One-way ANOVA can simultaneously compare all three means against the null hypothesis that all the means are equal.

ANOVA works by weighing the variation between the independent variable conditions (Mac vs. Windows vs. bottom) against the variation within the conditions (which is due to other factors like individual differences and random noise). If the null hypothesis is true, then the independent variable doesn't matter, so dividing up the observations according to the independent variable is merely an arbitrary labeling. Thus, assuming we randomized our experiment properly, the variation **between** those arbitrary groups should be due entirely to chance, and identical to the random variation **within** each group. So ANOVA takes the ratio of the between-group variation and the within-group variation, and if this ratio is significantly greater than 1, then that's sufficient evidence to argue that the null hypothesis is false and the independent variable actually **does** matter.

Like the  $t$  test, ANOVA also assumes that the samples are independent, normally distributed, and have equal variance.



## Running ANOVA (in R)

```
data3 = read.csv(file.choose())
• fit = aov(data3$time ~ data3$condition)
• summary(fit)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
menubar	2	41553	20776.5	3.6909	0.03828
Residuals	27	151984	5629.1		

Spring 2012

6.813/6.831 User Interface Design and Implementation

16

Here's an example of running an ANOVA test. The fictitious experiment here is a between-subjects experiment with three conditions: Windows menubar, Mac menubar, and menubar at bottom of screen. So our condition factor in this dataset now has three different values in it (win, mac, btm). The aov function ("analysis of variance") does the test, and returns an object with the results. If we just display that object directly, however, it doesn't give us the information we want, like the F statistic and the p value. We have to use the summary() function to get out the critical stuff. See <http://www.statmethods.net/stats/anova.html>

Here's how to read the output. **Sum Sq** shows the sum of the squared *deviations from the mean*, which is how ANOVA measures how broadly a sample varies. The residual SumSq shows the deviation of each sample from its group's mean, so the first Windows sample would contribute  $(625-584.0)^2$  to the within-groups SS. The condition SumSq replaces each sample with its group's mean and then uses the deviation of these group means from the overall mean of all samples; so the same Windows sample would contribute  $(584.0-531.5)^2$  to the between-groups SS. **df** is the degrees of freedom of each SumSq statistic, and **Mean Sq** is the mean sum of squared deviations (SS/df). Finally the **F** statistic is the ratio of the between-groups MS and the within-groups MS. It is this ratio that tells us whether there is more variation *between* the three menubar conditions than *within* the samples for each (due to other random uncontrolled variables, like user differences). If the F statistic is significantly greater than 1, then the **p-value** (Pr>F) will show significance

In this case, the p value is 0.038, so we say that there is a significant difference between the three menubars (one-way ANOVA,  $F_{2,27}=3.69$ ,  $p < 0.05$ ). Note that degrees of freedom for the F statistic are usually shown as subscripts, as shown.

It turns out that ANOVA is equivalent to the t test when the number of conditions is 2; in that case, the F statistic used in ANOVA is related to the t statistic simply as  $F=t^2$ , and you get the same p value from both tests.

## Running Within-Subjects ANOVA (in R)

- `data4 = read.csv(file.choose())`
- `fit = aov(data4$time ~ data4$condition +  
Error(data4$subject/data4$condition))`
- `summary(fit)`

```
Error: data4$subject
      Df Sum Sq Mean Sq F value Pr(>F)
Residuals  9  88203  9800.3

Error: data4$subject:data4$condition
      Df Sum Sq Mean Sq F value Pr(>F)
data4$condition  2  41553 20776.5  5.8634 0.01094 *
Residuals      18  63782  3543.4
```

condition	time	subject
win	625	u1
win	480	u2
win	621	u3
win	633	u4
win	694	u5
win	599	u6
win	505	u7
win	527	u8
win	651	u9
win	505	u10
mac	647	u1
mac	503	u2
mac	559	u3
mac	586	u4
mac	458	u5
mac	380	u6
mac	477	u7
mac	409	u8
mac	589	u9
mac	472	u10
btm	485	u1
btm	436	u2
btm	512	u3
btm	564	u4
btm	560	u5
btm	587	u6
btm	391	u7
btm	444	u8
btm	444	u9
btm	444	u10

Spring 2012

6.813/6.831 User Interface Design and Implementation

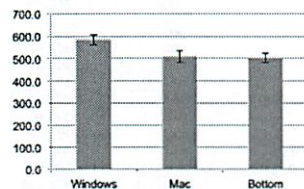
17

Within-subjects ANOVAs are possible in R, but require more information. First, we need to create a factor for the subject – which subject provided each measurement? That factor now becomes one of the independent variables of the experiment. But when we write the model, we use this factor not as part of the process, but in a special Error term, as shown.

<http://www.statmethods.net/stats/anova.html>

## Tukey HSD Test

- Tests pairwise differences for significance after a significant ANOVA test
  - More stringent than multiple pairwise t tests



- Be careful in general about applying multiple statistical tests

Spring 2012

6.813/6.831 User Interface Design and Implementation

18

So the ANOVA test told us that the choice of menubar affects time. But how? By itself the test doesn't say *which* differences are significant. Is Windows significantly worse than the other two, but Mac and Bottom are basically the same? Or are all three different from each other? Graphing with error bars can give a rough answer; it looks like Mac and Bottom are within their error bars of each other, while Windows is beyond. For a statistical test of this question, we can use the Tukey HSD (Honestly Significant Difference) test, which is also sometimes called the Tukey post-hoc test.

Not every stats package can do this test, but there are web sites that do, and the calculation of the Tukey statistic uses data that is already included in an ANOVA report (like the means for each group, the within-group degrees of freedom, and the within-group MS), so it's straightforward. The larger the Tukey statistic is, the better. In this case (for  $n=10$  and  $df=27$ ), the critical value for 5% significance is roughly 3.5. None of the pairwise comparisons reach that level, so even though we can say that the choice of menubar significantly affected time at a 5% level, we don't have enough evidence to say that the Windows menubar was actually worse at the 5% level.

Why don't we just apply a t test between each pair of conditions? That's a risky thing to do. Statistical testing is only sound when you apply just *one* test to any given set of data. Roughly speaking, any given test has a 5% chance of lying to you and indicating a significant difference at the 5% level even when there isn't one. (Statisticians call this a "type I error.") The more tests you do, the more likely you are to fall into this trap. This is why you need to choose your statistical tests *before* you collect your data, rather than just dredge around through the data afterwards to see what tests work. Sometimes experimenters *plan* to run multiple tests on the same data; when they do this, however, they use a stricter level of significance than 5% for each test, so that the overall ("familywise") risk of making the type I error is still bounded by 5%. This stricter significance is called "Bonferroni adjustment"; you'll see it in seriously empirical HCI papers from time to time.

The Tukey HSD test is an example of a **post-hoc test**, one that you didn't originally plan to run (because your decision to run it was triggered by the successful ANOVA), and so you didn't adjust your significance level for it. So the Tukey test is designed to be much more stringent than a t test – you need bigger differences in your data to get 5% significance. You may have noticed that the data in all these fake experiments happens to be identical – but where the t test comparing Win vs. Mac was significant, the Tukey HSD test was not.



## Tukey HSD Test (in R)

- TukeyHSD(fit)

Tukey multiple comparisons of means  
95% family-wise confidence level

Fit: aov(formula = time ~ menubar)

\$menubar	diff	lwr	upr	p adj
mac-btm	5.6	-77.592144	88.79214	0.9847693
win-btm	81.6	-1.592144	164.79214	0.0553394
win-mac	76.0	-7.192144	159.19214	0.0783231



## Two-Way ANOVA

- 2 independent variables with  $j$  and  $k$  levels, respectively
- Tests whether each variable has an effect independently
- Also tests for interaction between the variables

Spring 2012

6.813/6.831 User Interface Design and Implementation

20

ANOVA can be extended to multiple independent variables, by looking at the variation between different levels of one independent variable (while holding the other independent variable constant). This is **two-way** (or two-factor) ANOVA.

Two-way ANOVA can be used to analyze a within-subjects experiment, where one independent variable is the variable we were testing (e.g. menubar position), while the other independent variable is the user's identity.

This can really only be done in a real stats package like R – Excel and online statistical calculators don't support multiway ANOVAs. See <http://www.statmethods.net/stats/anova.html> for more about how to do it in R.

## Two-way Within-Subjects ANOVA (in R)

```
time      = [ 625, 480, ...,      647, 503, ...,      485, 436, ...]  
menubar = [ win, win, ...,      mac, mac, ...,      btm, btm, ...]  
device =  [ mouse, pad, ..., mouse, pad, ..., mouse, pad, ...]  
subject  = [ u1, u1, u2, u2, ..., u1, u1, u2, u2 ..., u1, u1, u2, u2, ...]
```

- `fit = aov(time ~ menubar*device + Error(subject/  
menubar*device))`
- `summary(fit)`

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
menubar	2	41553	20776.5	3.4086	0.08498
Residuals	8	48763	6095.3		

<http://www.statmethods.net/stats/anova.html>

## Other Tests

- Two discrete-valued variables
  - “does past experience affect menubar preference?”
    - independent var { WinUser, MacUser}
    - dependent var {PrefersWinMenu, PrefersMacMenu}
  - contingency table

	PrefersWin	PrefersMac
WinUser	25	9
MacUser	8	19
  - Fisher exact test and chi square test
- Two (or more) scalar variables
  - Regression

Spring 2012

6.813/6.831 User Interface Design and Implementation

22

The t test and ANOVA are designed for cases where your independent variable is **discrete-valued** (e.g. two or three menubar conditions) and your dependent variable is a **scalar** quantity (in fact, a normally distributed scalar). This is very common, but other kinds of variables sometimes appear in UI experiments.

When both variables are discrete-valued, you end up visualizing your data as a table of occurrence frequencies (called a contingency table) rather than a bar graph. There are two statistical tests for this kind of relationship: the Fisher exact test works best when the numbers are small (some frequencies in the table are less than 10) and the table is 2x2 (i.e., each variable has only two possible values). It can be computed for larger table sizes, but not by hand. The chi-square test is more suitable for large table dimensions, and when the frequencies in each cell are big enough (all greater than 5). Excel doesn't have these tests, but the web does.

When both variables are scalar, the right tool is regression. If our menubar experiment varied the starting distance from the menu, for example, and measured the time to access, then we might use regression to “fit a model” to the data that actually *quantifies* the relationship between these two variables – i.e., that time varies with the log of starting distance. Regression can do this fitting, and tell us how well the model fits the data (the analog of statistical significance for hypothesis testing) as well as the constant factors of the relationship. Regression is beyond the scope of this class (get it in a statistical methods course), but you'll see it from time to time in empirical HCI papers, particularly work on low-level input like Fitts's Law.

## Tools for Statistical Testing

- Web calculators
- Excel
- Statistical packages
  - Commercial: SAS, SPSS, Stata
  - Free: R

Spring 2012

6.813/6.831 User Interface Design and Implementation

23

There are many free calculators on the web for running simple t tests, ANOVAs, and the other tests mentioned in this lecture. Search for “T test calculator” and “ANOVA calculator”, for example, to find good ones. The calculators only really need summary statistics about your data (means, standard deviations, counts), which you can easily compute yourself in a spreadsheet, so it isn’t necessary to enter all your data in a web form.

Microsoft Excel also has some support for the common tests. It requires installing the optional Analysis Toolpak (go to Options/Add-ins/Manage Excel Add-ins); once installed, go to the Data tab and look for the Data Analysis button.

Full-fledged stats applications have much richer support for these tests, at the cost of a greater learning curve. MIT has some discounts for SPSS and Stata. There’s also a free GNU stats system called R which is rapidly growing in popularity.



## Summary

- Use t test to compare two means
- Use ANOVA to compare 3 or more means

## 6.813/6.831 • USER INTERFACE DESIGN AND IMPLEMENTATION

Spring 2012 Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

### GR3: PAPER PROTOTYPING

Prototype building	Monday, March 12, 2012, 3-5 pm, in Walker Memorial 3rd floor gym
Prototype testing	Wednesday, March 14, 2012, 3-5 pm, in Walker Memorial 3rd floor gym Friday, March 16, 2012, 3-5 pm, in Walker Memorial 3rd floor gym
Final hand-in	Due at 11:59 pm on Sunday, March 18, 2012, on the wiki.

In this group assignment, you will do your first implementation of your term project, which will be a paper prototype.

There are several class meetings associated with this assignment (days and times shown above):

- The **building session** offers you an opportunity to start building a paper prototype. The course staff will be available to make suggestions, and some materials will be provided.
- The **testing sessions** will allow you to test a paper prototype on your classmates. For part of the time, you will also serve as a user for somebody else's paper prototype.

You must do two rounds of testing, with a revision of your paper prototype in between. Each round must involve at least 3 users. Note that if you don't have time to run 3 users during the in-class testing sessions, you will have find users outside of class as well. After the first round, revise your paper prototype to address the critical usability problems and explore possible design alternatives.

#### Preparing for Testing

Before testing your prototype, you should:

- **Build your prototype.**

Draw the static background, menus, dialog boxes, and other windows. Decide how to implement the dynamic parts of your interface. **Hand-sketching is encouraged.** You don't have to prepare every possible screen in advance; it may be much easier to write responses on the fly.

- **Prepare a briefing for test users.**

This should be at most a page of information about the purpose of your application and any background information about the domain that may be needed by your test users (who may be classmates) to understand it. These are your notes for the briefing, so make them short, simple and clear, not dense wordy paragraphs. This is not a manual or quick-reference card. It should not describe how to use the interface.

- **Write your scenario tasks on separate index cards.**

Your scenario should have involved at least three tasks. You should write these tasks down to give to your users. Just write the concrete goal(s) of the task (e.g. "buy milk, tomatoes, and bread"). Don't write the specific steps to follow, since that's for your users to figure out. The

tasks should be brief, roughly 5 minutes to run.

- **Choose roles for your team members.**

One person must play the computer. The other team members (if any) will be observers. We won't bother with a facilitator for these pilot tests. It may be useful for you to swap roles after every user on during the testing sessions, so that each of you gets a chance to try each role, but decide how you'll do it in advance.

- **Practice running your paper prototype.**

Every team member should practice playing the computer, learning the steps involved in making the prototype functional, such as rearranging pieces and writing responses. It isn't important to be fast, just competent and confident. A few trials are enough. Make sure your prototype can handle the tasks involved in your scenario.

## Running the Tests

When you run your prototype on a user, you should do the following things:

- **Brief the user.**

Use the briefing you wrote up to describe orally the purpose of the application and background information about the domain. Don't waste too much time on this: 1 minute should be enough.

- **Present one task.**

Hand the index card to the user and let them read it. Make sure they understand the task.

- **Watch the user do the task.**

Take notes from your observations.

- **Repeat with the other tasks.**

Run as many tasks on the user as you have time for.

Bring extra materials to your testing sessions. Having extra blank Post-it notes, correction tape, and index cards on hand will help you improvise if a user does something unexpected, or help you make small fixes to your prototype between users.

## Playing a User for Your Classmates

During the testing sessions, when you are serving as a user, you should:

- **Relax and enjoy yourself.**

You're not being tested -- the interface is. Part of the point of this experience is to feel what it's like to be users in a user test, so that you can empathize with them.

- **Be cooperative.**

Don't be intentionally dense, e.g. looking for Exit everywhere but the File menu. Interact with the interface as you would if you were really using it.

- **Think aloud.**



Help the observers understand what you're thinking by verbalizing your thought process. "Let's see, I want to enter this bottle of milk, so where's the scanner... oh, here it is. I'll scan the bottle like this, oops that didn't work, let me find the bar code..." You get the idea.

### What to Hand In

Update your group's wiki page so that it contains a section **GR3 Paper Prototyping**, containing the following subsections:

- **Prototype photos.**

Digital photos of the pieces of your prototype. Show the prototype in interesting states; don't just show a blank window. Although you will iterate your paper prototype during this assignment, the photos only need to show one iteration.

- **Briefing.**

The briefing you gave to users.

- **Scenario Tasks.**

The tasks you gave to users, as you wrote them on the cards.

- **Observations.**

Usability problems you discovered from the testing. Describe what users did, but **don't record users' names**.

- **Prototype iteration.**

You did two rounds of paper prototyping. Describe how your prototype changed between those two rounds.



3/16

## Learning/changes

— Admin links — Underline + letters

~~all~~ leave end date plank

Everyone misses download

What does event do

— fill in RSVP #s

— label graphs

— add slashes

~~all~~ — make #s bigger on download pg

— no login

---

email rider

describe steps

RGB confused

make clearer barcode scanner  
app

3/16

dshayden - Bubble Cursor

d\_chang - menu  
cezeozue

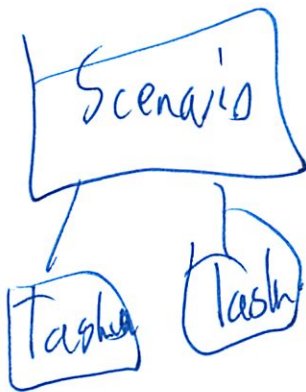
6.813 Grad Students

I helped

(talking about PS2)

Leash<sup>code</sup> almost done: The online file to download:  
- they used a table They posted the solution:

All done - just need to write up



~~6/17/11~~

Filling all of it in

## Untitled

### Briefing:

RScanVP is a system that allows event organizers to add QR codes to their posters. Attendees can scan these QR codes to add the events to their calendars. Event organizers can view the history of scans and RSVPs to help with their event planning.

You are the event chair of HKN and you are planning a lecture called Bob's UI Lecture. You are making a poster for the event. You want people to be able to easily add the event to their calendars, so you add a RScanVP QR code.

### Scenario:

- Create a QR code

### Observations:

- Subject is unsure about how to begin. (because of paper)
- Subject did not notice the download button (because of piece of paper related to email?)

### Scenario:

- You are taking a picture of a QR code on a poster.

### Observations:

- Subject asks about phone interface, not so obvious it was Subject's calendar.
- Subject didn't notice the Yes/No slider (not drawn well); thought it would perform an action

### Scenario:

- User receives an email with an admin link to view results several days later.

### Observations:

- Subject didn't notice the link in the email (was not underlined)
- Subject did not know the purpose of the admin page (describe it in an email)
- User wants a complaint link...



Name:

Age:

Email:

to receive email updates

Password:

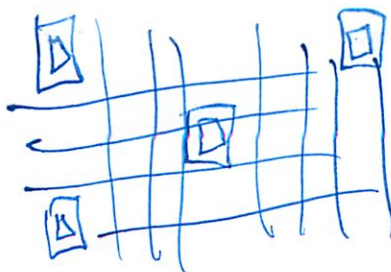
Create  
Account

email:

password:

login

Welcome, User! logout



Email

# RScanVP

create account login

A quick and reliable way to gain RSVP statistics about your upcoming events. Click the button on right to begin, or log in to view your results.

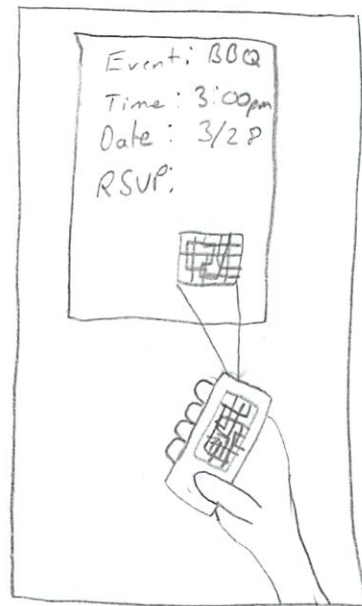
Make A  
QR  
Code

RScanVP > Create a QR Code  
Step 2: Design QR Code

You can always change this

Colorscale	R	<input type="text"/>
	G	<input type="text"/>
	B	<input type="text"/>

Size  px  x  px



Zuckerberg Lecture  
Mon. 3/28 @ 4:00pm  
Upper Kresge

RSVP now:



# RScan VP > Create a QR Code

## Step 1: Enter details

Title

Location

Start  Cal

~~Time~~  AM

End  Cal

AM

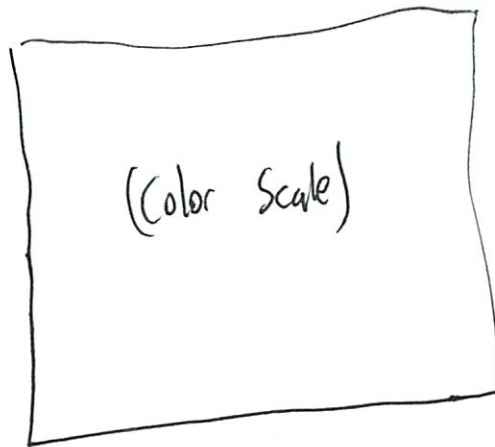
Cancel

Next >

# RScan VP > Create a QR Code

## Step 2: Design QR Code

You can always change this later



R

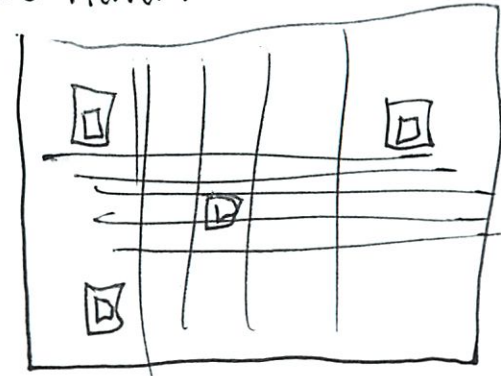
G

B

Size  px  x  px

[Back](#)

Live Preview:



Size:  x

[Download >](#)



# RScanVP > Create a QR Code

## Step 3: Download

1. Download Code GIF PNG

2. Insert into poster ~~via~~  
using your favorite graphic design program

An email <sup>will be</sup> ~~has been~~ sent to you with a link to  
edit the QR code and to view RSVPs

Cancel

Done >

## Task 2:

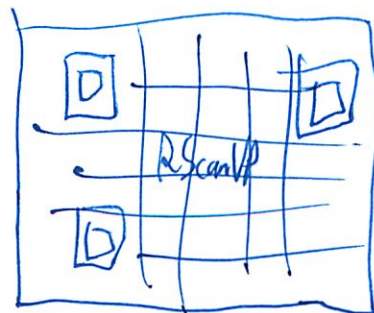
- Scan a QR code with your phone and RSVP to an event.

# Bob's UI Lecture

Mon 4/12 4-5 PM

32 - 123

Scan this QR code to add  
to your calendar

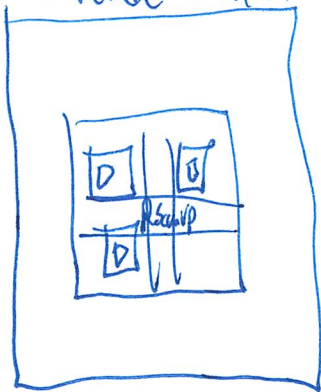


### Task 3:

- View RSVPs to one of your events. So can order food



# Barcode Scanner



RScan VP

Bob's UI Lecture

Mon 4/12 4-5 PM

32-123

Add to Calendar

RSVP

☒ Yes ☐ No

---

Alt methods

Google Cal

Monday April 9 2012

10

11

12

1

2

3

4

5

6

Bob's VI Lecture  
32-123

### Task 3:

- View RSVPs to one of your events. So can order food



# Email

To: You

From: RScanVP

Subject: Bob's UI Lecture Admin link

Here is the admin link for  
Bob's UI lecture

Edit QR code

Edit event details

View RSVP

# Bob's UI Lecture

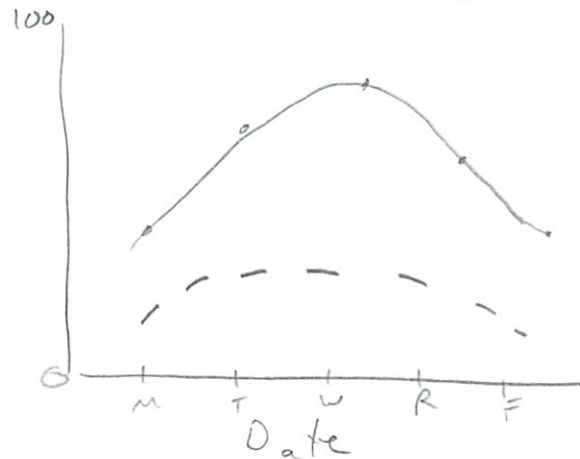
[<back](#)

Number of Scans 75

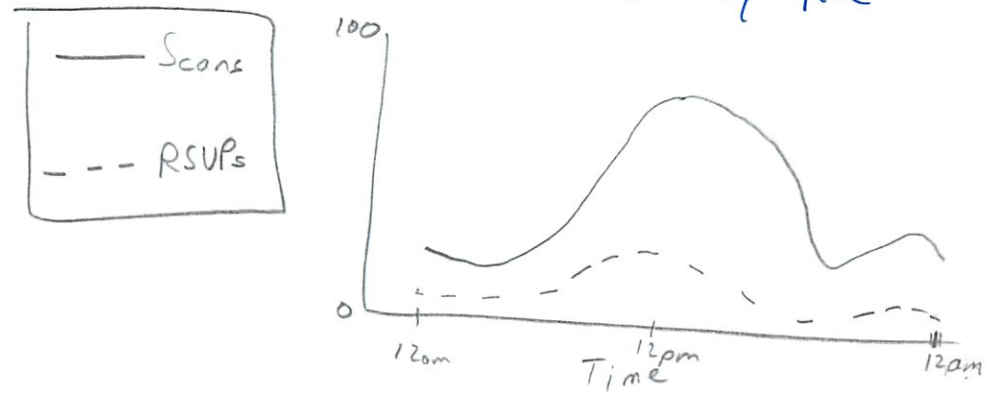
Number of RSUPs 50

[Edit QR code](#)

Scans by Day



Scans by Time



RSUP List: (email group)

- ~~~~~  
- ~~~~~  
- ~~~~~  
- ~~~~~  
- ~~~~~

- ~~~~~  
- ~~~~~  
- ~~~~~  
- ~~~~~

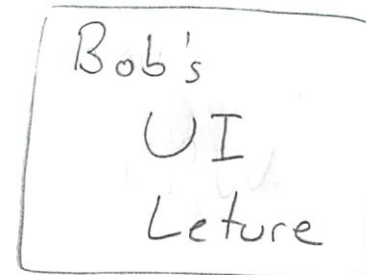
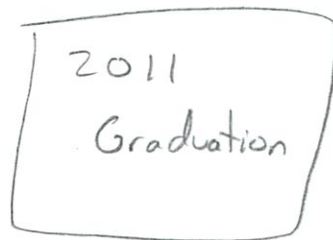
- ~~~~~  
- ~~~~~  
- ~~~~~  
- ~~~~~

# Welcome, User!

Choose an event:



Completed Events



PS2 due at end of Spring Break

Hall of Fame/Shame Google Home Page

- Simplicity
- Learnability
  - it doesn't say what it for
- What is "I'm Feeling Lucky"?
- tooltip?
- or show results then click on 1st one
- No help
  - its in About Google
  - but can search for it

---

## Nano quiz

COMUS - Nazi + Syphilis drove federal

User talks the most informative eval

Formative eval - identify visibility problems  
? "forming a new VI"



(2)

Today

Simplicity

- reduction
- regularity
- double duty

Shifting  
aesthetics

Contrast

- visual variables
- associative + selectivity
- squint test

Simplicity

- less is more
- remove inessential elements
- remove inessential features
- if I remove it do I have a working system?
- wheel chair system



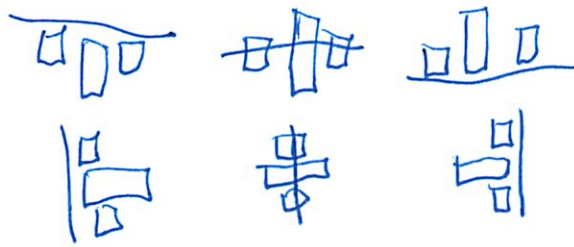
- toss out till can't any more
- TiVo remote

③

## Regularity

make same decision decision

like how do alignment boxes



## Double Duty

A part of the system gives multiple roles

### Door knobs

- turns latch
- lets you open door
- ~~lets~~ lets you see which way door opens

~~Window tilt~~

### Scrollbar ~~and~~ thumb

- lets you move
- but also lets you see how large a doc is

④

Text box w/ gray default text

Bread crumbs

page #s

## Contrast

Allows changes to pop out

Making select things stand out

- Value - darkness

- hue - color

- shape

- position

- Orientation

- size

texture

- used to be trace in lecture

- less relevant in UI

Are perceived earlier than #s or words

5

Nominal - can compare only for equality

Ordered - can compare  $>$   $<$

Quantitative - can compare amt of difference

What is easy to perceive:

Table in notes

Can't use angles to encode differences in class  
Distinction too ~~small~~ small to see difference

Hue is not ordered or quantitative

People try ROYGBIV spectrum

But must do mental mapping to scale

So much more work

Could use brightness / saturation difference

Can select for position - look to certain part (left or right)

Color - see red letters

But not all the hrs - must scan around  $\rightarrow$  Shape is not selective  
- examine each object




⑥

Selectivity - can attention be focused on

Associative - can variable be ignored when looking at the other variables

---

### Techniques

- choose appropriate visual variables
- squint test
  - close 1 eye
  - let other details
- like see bold / size of unread emails in many emails
  - values becoming blacker
- but icon does not strongly pop out in a squint test
  - instead
    - o  circles
      - small vs big
      - white vs green

⑦

It's not needed to label everything

title: —  
Summary: —  
etc

vs Google results

- colors
- position
- hierarchy
- size
- bold (value)

Let your data describe itself!

You can get rid of it and it still works

---

Redesign Movies list

Restaurant entry

3/19

## **L16: Graphic Design**

- PS2/RS2 due at end of spring break
- GR4 (computer prototype) starts right after spring break
- consider doing more paper prototyping in the meantime



Google is an outstanding example of **simplicity**. Its interface is as simple as possible. Unnecessary features and hyperlinks are omitted, lots of whitespace is used. Google is fast to load and trivial to use.

But maybe Google goes a little too far! Take the perspective of a completely novice user coming to Google for the first time.

- What does Google actually do? The front page doesn't say.
- What should be typed into the text box? It has no caption at all.
- The button labels are almost gibberish. "Google Search" isn't meaningful English (although it's gradually becoming more meaningful as *Google* enters the language as a noun, verb, and adjective). And what does "I'm Feeling Lucky" mean?
- Where is Help? Turns out it's reachable from About Google, but the scent isn't too strong for that.

Although these problems would be easy for Google to fix, they are actually minor, because Google's interface is simple enough that it can be learned by only a small amount of exploration. (Except perhaps for the I'm Feeling Lucky button, which probably remains a mystery until a user is curious enough to hunt for the help. After all, maybe it does a random choice from the search results!)

Notice that Google does **not** ask you to choose your search domain first. It picks a good default (web pages), includes a mix of results if they seem relevant (e.g. images & videos & maps too, not purely web pages), and makes it easy to change.



## Today's Topics

- Simplicity
  - reduction
  - regularity
  - double-duty
- Contrast
  - visual variables
  - associativity & selectivity
  - squint test

# **SIMPLICITY**

Spring 2012

6.813/6.831 User Interface Design and Implementation

6

## Simplicity

- "Perfection is achieved not when there is nothing more to add, but when there is nothing left to take away."
  - Antoine de St-Exupery
- "Simplicity does not mean the absence of any decor... It only means that the decor should belong intimately to the design proper, and that anything foreign to it should be taken away."
  - Paul Jacques Grillo
- "Keep it simple, stupid." (KISS)
- "Less is more."
- "When in doubt, leave it out."

Spring 2012

6.813/6.831 User Interface Design and Implementation

7

Okay, we'll shout some slogans at you now. You've probably heard some of these before. What you should take from these slogans is that designing for simplicity is a process of *elimination*, not accretion. Simplicity is in constant tension with task analysis, information preconditions, and other design guidelines, which might otherwise encourage you to pile more and more elements into a design, "just in case." Simplicity forces you to have a good reason for everything you add, and to take away anything that can't survive hard scrutiny.

## Techniques for Simplicity: Reduction

- Remove inessential elements



- Remove inessential features



Spring 2012

6.813/6.831 User Interface Design and Implementation

8

Here are three ways to make a design simpler.

**Reduction** means that you eliminate whatever isn't necessary. This technique has three steps: (1) decide what essentially needs to be conveyed by the design; (2) critically examine every element (label, control, color, font, line weight) to decide whether it serves an essential purpose; (3) remove it if it isn't essential. Even if it seems essential, try removing it anyway, to see if the design falls apart.

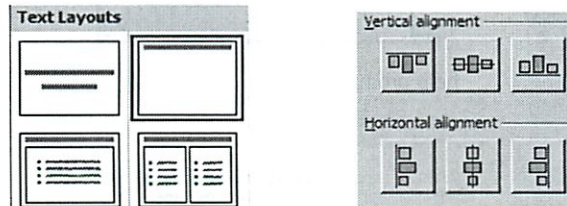
**Icons** demonstrate the principle of reduction well. A photograph of a pair of scissors can't possibly work as a 32x32 pixel icon; instead, it has to be a carefully-drawn picture which includes the bare minimum of details that are essential to scissors: two lines for the blades, two loops for the handles. The standard US Department of Transportation symbol for handicapped access is likewise a marvel of reduction. No element remains that can be removed from it without destroying its meaning.

We've already discussed the minimalism of Google. The Tivo remote is another notable example, about minimalizing **functionality**. It's much simpler than comparable remote controls, which tend to be dense arrays of tiny rectangular buttons, all alike. Tivo's designers aggressively removed functions from the remote, to keep it as simple as possible ("Now Preening on the Coffee Table", *New York Times*, Feb 19, 2004, <http://query.nytimes.com/gst/fullpage.html?res=9c0de2d6123df93aa25751c0a9629c8b63>).



## Techniques for Simplicity: Regularity

- Use a regular pattern
- Limit inessential variation among elements



Spring 2012

6.813/6.831 User Interface Design and Implementation

9

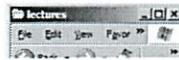
For the essential elements that remain, consider how you can minimize the unnecessary differences between them with **regularity**. Use the same font, color, line width, dimensions, orientation for multiple elements. Irregularities in your design will be magnified in the user's eyes and assigned meaning and significance. Conversely, if your design is mostly regular, the elements that you do want to highlight will stand out better.

PowerPoint's Text Layouts menu shows both reduction (minimalist icons representing each layout) and regularity. Titles and bullet lists are shown the same way.

## Techniques for Simplicity: Double-Duty

- Combine elements for leverage
  - Find a way for one element to play multiple roles

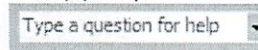
title bar



scrollbar thumb



help prompt



breadcrumbs

[Travel](#) > [Guides](#) > North America

pagination

Results Page:  
1 2 3 4 5 6 7 8 9 10 > Next

Spring 2012

6.813/6.831 User Interface Design and Implementation

10

Another technique for simplicity is to **combine elements**, making them serve multiple roles in the design. Desktop and web interfaces have a number of patterns in which elements have multiple duties. For example, the “thumb” in a scroll bar actually serves three roles. It affords dragging, indicates the position of the scroll window relative to the entire document, and indicates the fraction of the document displayed in the scroll window. Similarly, a window’s title bar plays several roles: label, dragging handle, window activation indicator, and location for window control buttons. In the classic Mac interface, in fact, even the activation indicator played two roles. When the window was activated, closely spaced horizontal lines filled the title bar, giving it a perceived affordance for dragging.

The breadcrumbs pattern and the pagination pattern also do double duty, not only showing you where you are but also providing an affordance for navigating somewhere else. Pagination links, like a scrollbar, may also show you how many pages there are.

# **CONTRAST**

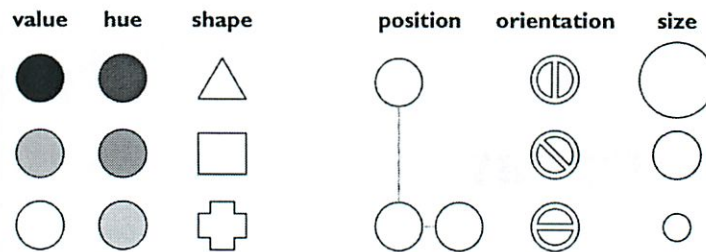
Spring 2012

6.813/6.831 User Interface Design and Implementation

11

## Contrast & Visual Variables

- Contrast encodes information along visual dimensions



Spring 2012

6.813/6.831 User Interface Design and Implementation

12

**Contrast** refers to perceivable differences along a visual dimension, such as size or color. Contrast is the irregularity in a design that communicates information or makes elements stand out. Simplicity says we should eliminate **unimportant** differences. Once we've decided that a difference is important, however, we should choose the dimension and degree of contrast in such a way that the difference is salient, easily perceptible, and appropriate to the task.

Crucial to this decision is an understanding of the different visual dimensions. Jacques Bertin developed a theory of *visual variables* that is particularly useful here (Bertin, *Graphics and Graphics Information Processing*, 1989). Visual variables identified by Bertin are shown above. Bertin called these dimensions *retinal variables*, in fact, because they can be compared effortlessly without additional cognitive processing, as if the retina were doing all the work.

Each column in this display varies along only one of the six variables. Most of the variables need no explanation, except perhaps for hue and value. **Hue** is pure color; **value** is the brightness or luminance of color. (Figure after Mullet & Sano, p. 54).



## Characteristics of Visual Variables

- Scale = kinds of comparisons possible
  - Nominal (can compare only for equality)
  - Ordered (can compare <, >)
  - Quantitative (can compare amount of difference)
- Length = number of distinguishable levels

	Value	Hue	Shape	Position	Orient	Size
Nominal	✓	✓	✓	✓	✓	✓
Ordered	✓			✓		✓
Quantitative				✓		✓
Scale	~10	~10	very long	very long	~4	~10
Spring 2012	6.813/6.831 User Interface Design and Implementation					13

The visual variables are used for communication, by encoding data and drawing distinctions between visual elements. But the visual variables have different characteristics. Before you choose a visual variable to express some distinction, you should make sure that the visual variable's properties match your communication. For example, you could display a temperature using any of the dimensions: position on a scale, length of a bar, color of an indicator, or shape of an icon (a happy sun or a chilly icicle). Your choice of visual variable will strongly affect how your users will be able to perceive and use the displayed data.

Two characteristics of visual variables are the kind of **scale** and the **length** of the scale.

A **nominal** scale is just a list of categories. Only comparison for equality is supported by a nominal scale. Different values have no ordering relationship. The shape variable is purely nominal. Hue is also purely nominal, at least as a *perceptual* variable. Although the wavelength of light assigns an ordering to colors, the human perceptual system takes no notice of it. Likewise, there may be some cultural ordering imposed on hue (red is "hotter" than blue), but it's weak, doesn't relate all the hues, and is processed at a higher cognitive level.

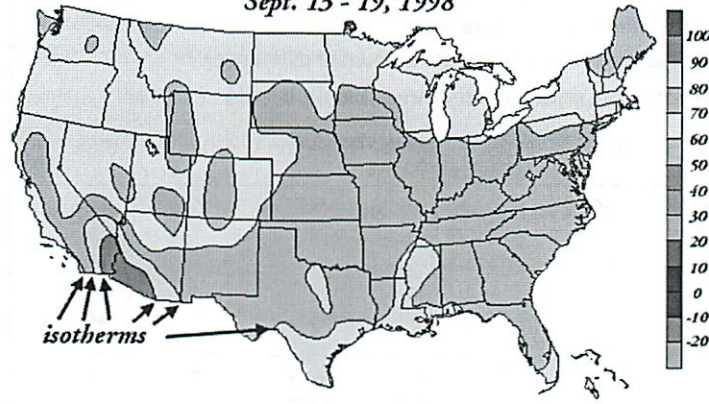
An **ordered** scale adds an ordering to the values of the variable. Position, size, and value are all ordered.

With a **quantitative** variable, you can perceive the *amount* of difference in the ordering. Position is quantitative. You can look at two points on a graph and tell that one is twice as high as the other. Size is also quantitative, but note that we are far better at perceiving quantitative differences in one dimension (i.e., length) than in two dimensions (area). Value is not quantitative; we can't easily perceive that one shade is twice as dark as another shade.

The **length** of a variable is the number of distinguishable values that can be perceived. We can recognize a nearly infinite variety of shapes, so the shape variable is very long, but purely nominal. Position is also long, and particularly fine-grained. Orientation, by contrast, is very short; only a handful of different orientations can be perceived in a display before confusion starts to set in. The other variables lie somewhere in between, with roughly 10 useful levels of distinction, although size and hue are somewhat longer than value.

## Hue Is Not Ordered or Quantitative

*Average Temperature (°F)  
Sept. 13 - 19, 1998*



Spring 2012

6.813/6.831 User Interface Design and Implementation

14

## Selectivity & Associativity

- Selective perception
  - Can attention be focused on one value of the variable, excluding other variables and values?
    - Shape doesn't "pop out"
- Associative perception
  - Can variable be ignored while looking at other variables?
    - Small size and low value interfere with ability to perceive hue, value, and shape

	Value	Hue	Shape	Position	Orient	Size
Selective	✓	✓		✓	✓	✓
Associative		✓	✓	✓	✓	

Spring 2012

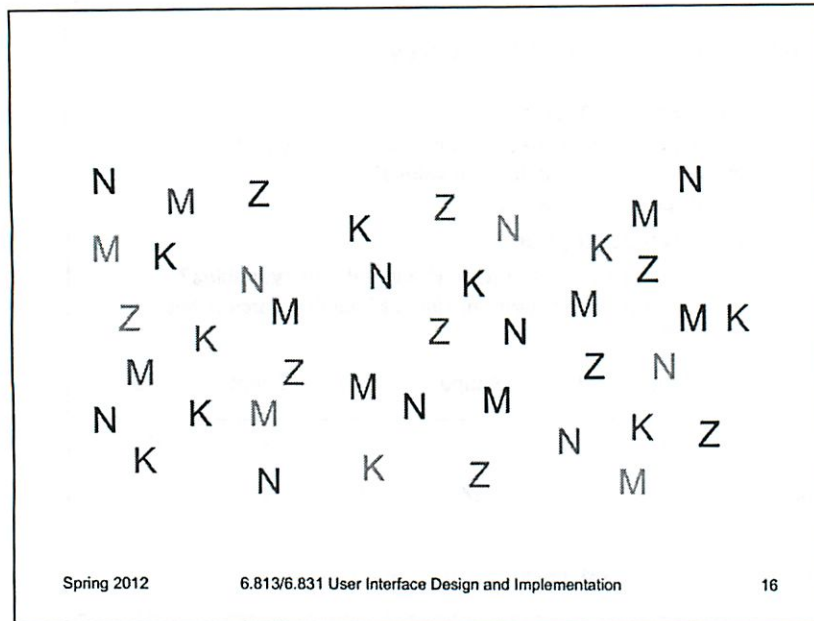
6.813/6.831 User Interface Design and Implementation

15

There are two ways that your choice of visual variables can affect the user's ability to attend to them.

**Selectivity** is the degree to which a single value of the variable can be selected from the entire visual field. Most variables are selective: e.g., you can locate green objects at a glance, or tiny objects. Shape, however, is not selective in general. It's hard to pick out triangles amidst a sea of rectangles.

**Associativity** refers to how easy it is to ignore the variable, letting all of the distinctions along that dimension disappear. Variables with poor associativity interfere with the perception of other visual dimensions. In particular, size and value are dissociative, since tiny or faint objects are hard to make out.



Ask yourself these questions:

- find all the letters on the left edge of the page (**position**)
- find all the red letters (**hue**)
- find all the K's (**shape**)

Which of these questions felt easy to answer, and which felt hard? The easy ones were **selective** visual variables.



## Techniques for Contrast

- Choose appropriate visual variables
- Use as much length as possible
- Sharpen distinctions for easier perception
  - Multiplicative scaling, not additive
  - Redundant coding where needed
  - Cartoonish exaggeration where needed
- Use the “squint test”

Spring 2012

6.813/6.831 User Interface Design and Implementation

17





















Once you’ve decided that a contrast is essential in your interface, **choose the right visual variable** to represent it, keeping in mind the data you’re trying to communicate and the task users need to do with the data. For example, consider a text content hierarchy: title, chapter, section, body text, footnote. The data requires an ordered visual variable; a purely nominal variable like shape (e.g., font family) would not by itself be able to communicate the hierarchy ordering. If each element must communicate multiple independent dimensions of data at once (e.g., a graph that uses size, position, and color of points to encode different data variables), then you need to think about the effects of associativity and selectivity.

Once you’ve chosen a variable, use as much of the **length** of the variable as you can. Determine the minimum and maximum value you can use, and exploit the whole range. In the interests of simplicity, you should minimize the number of distinct values you use. But once you’ve settled on N levels, distribute those N levels as widely across the variable as is reasonable. For position, this means using the full width of the window; for size, it means using the smallest and the largest feasible sizes.

Choose variable values in such a way as to make **sharp, easily perceptible distinctions** between them. Multiplicative scaling (e.g., size growing by a factor of 1.5 or 2 at each successive level) makes sharper distinctions than additive scaling (e.g., adding 5 pixels at each successive level). You can also use redundant coding, in several visual variables, to enhance important distinctions further. The title of a document is not only larger (size), but it’s also centered (position), bold (value), and maybe a distinct color as well. Exaggerated differences can be useful, particularly when you’re drawing icons: like a cartoonist, you have to give objects exaggerated proportions to make them easily recognizable.

The **squint test** is a technique that simulates early visual processing, so you can see whether the contrasts you’ve tried to establish are readily apparent. Close one eye and squint the other, to disrupt your focus. Whatever distinctions you can still make out will be visible “at a glance.”

## Choosing Visual Variables for a Display

	Subject	Sender		Date
	Содействие в трудоустройстве.	chao		10/15/2004 4:26...
	Автовладелец	АвтоГранд		10/15/2004 4:45...
	Обучение теннису	eliot		10/15/2004 7:15 AM
	PITTSBURGH PA Silverton Home Services for...	Erica Gallenbeck		10/15/2004 7:21...
	156 - 00 - 00 &#1085;&#1072;&#1096; &#1...	XJXFLLXmXpX@tdb.com		10/15/2004 10:4...
	156-00-00	hucksterEOPN		10/15/2004 11:12 ...
	A Library A Dream...	Arthur GuoBin Yin		10/15/2004 6:38...
	SAVE 20% on holiday cards by shopping early	Snapfish		5:18 AM
	How are you	Анисимов К.И.		11:24 AM

Spring 2012

6.813/6.831 User Interface Design and Implementation

18

Let's look at an email inbox to see how data associated with email messages are encoded into visual variables in the display. Here are the data fields shown above, in columns from left to right:

**Spam flag:** nominal, 2 levels (spam or not)

**Subject:** nominal (but can be ordered alphabetically), infinite (but maybe only ~100 are active)

**Sender:** nominal (but can be ordered alphabetically), infinite (but maybe ~100 people you know + everybody else are useful simplifications)

**Unread flag:** nominal, 2 levels (read or unread)

**Date:** quantitative (but maybe ordered is all that matters), infinite (but maybe only ~10 levels matter: today, this week, this month, this year, older)

This information is **redundantly** coded into visual variables in the display shown above, for better contrast. First, all the fields use position as a variable, since each is assigned to a different column. In addition:

Spam: shape, hue, value, size (big colorful icon vs. little dot)

Subject: shape

Sender: shape

Unread: shape, hue, value, size (big green dot vs. little gray dot) and value of entire line (boldface vs. non)

Date: shape, size (today is shorter than earlier dates), position (list is sorted by date)

Exercise: try designing a visualization with these encodings instead:

Spam: size (this takes advantage of dissociativity)

Subject: shape

Sender: position

Unread: value

Date: position

## Designing Information Displays

Title: HCI Bibliography : Human-Computer Interaction / User Interface ...  
Summary: The HCI Bibliography (HCIBIB) is a free-access bibliography on Human-Computer Interaction, with over 20000 records in a searchable database. ... Learn about HCI. ...  
Keywords: HCI  
URL: [www.hcibib.org/](http://www.hcibib.org/)  
Size: 14k

### HCI Bibliography : Human-Computer Interaction / User Interface ...

The HCI Bibliography (HCIBIB) is a free-access bibliography on Human-Computer Interaction, with over 20000 records in a searchable database. ... Learn about HCI. ...  
[www.hcibib.org/](http://www.hcibib.org/) - 14k - [Cached](#) - [Similar pages](#)

### Human-Computer Interaction Resources on the Net

... This is a collection of information related to Human-Computer Interaction (HCI). ...  
Collections of resources for HCI researchers and practitioners ...  
[www.ida.liu.se/labs/aslab/groups/um/hci/](http://www.ida.liu.se/labs/aslab/groups/um/hci/) - 9k - [Cached](#) - [Similar pages](#)

Spring 2012

6.813/6.831 User Interface Design and Implementation

19

Here's another example showing how redundant encoding can make an information display easier to scan and easier to use. Search engine results are basically just database records, but they aren't rendered in a simplistic caption/field display like the one shown on top. Instead, they use rich visual variables – and no field labels! – to enhance the contrast among the items. Page titles convey the most information, so they use size, hue, and value (brightness), plus a little shape (the underline). The summary is in black for good readability, and the URL and size are in green to bracket the summary.

Take a lesson from this: your program's *output displays* do not have to be arranged like *input forms*. When data is self-describing, like names and dates, let it describe itself. (This is yet another example of the **double duty** technique for achieving greater simplicity – data is acting as its own label.) And choose good visual variables to enhance the contrast of information that the user needs to see at a glance.

## Let's Redesign This

My Movies		ADD
Title	Year	
The Lord of the Rings: The Return of the King	2003	
Pirates of the Caribbean: Dead Man's Chest	2006	
The Dark Knight	2008	
Harry Potter and the Sorcerer's Stone	2001	
Pirates of the Caribbean: At World's End	2007	
Harry Potter and the Order of the Phoenix	2007	
Harry Potter and the Half-Blood Prince	2009	
The Lord of the Rings: The Two Towers	2002	
Star Wars Episode I: The Phantom Menace	1999	
Alice in Wonderland	2010	
Shrek 2	2004	
Jurassic Park	1993	

Spring 2012

6.813/6.831 User Interface Design and Implementation

20



## Let's Redesign This

### Hong Kong Cafe



[view restaurant](#)

Address:  
1033 Commonwealth Avenue  
Boston, MA 02215  
617.787.4242

Hours:  
Weekends:  
11am-11pm  
Weekdays:  
11am-10pm

## Contrast in Publication Styles

# Title

## Heading

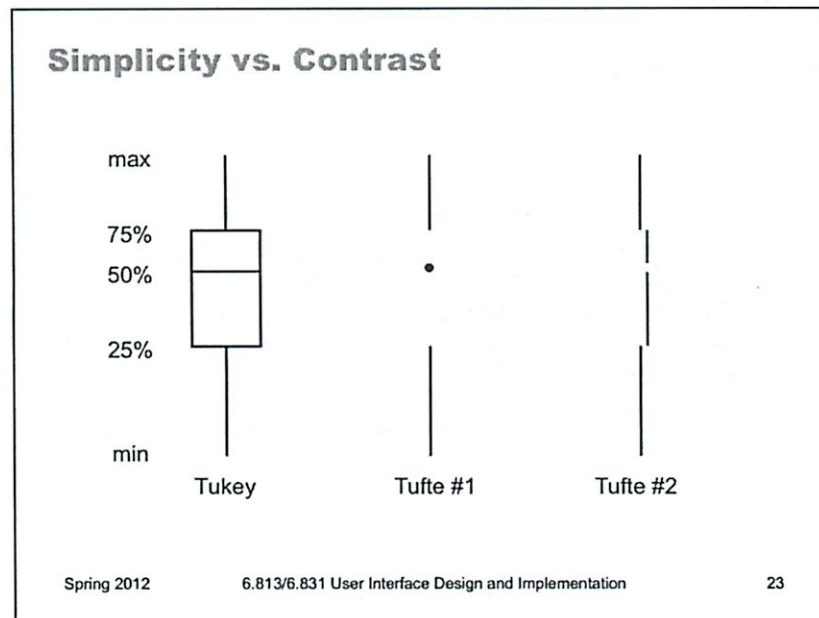
This is body text. It's smaller than the heading, lighter in weight, and longer in line length. We've also changed its shape to a serif font, because serifs make small text easier to read. Redundant encoding produces an effective contrast that makes it easy to scan the headings and distinguish headings from body text.<sup>1</sup>



Figure 1. This is a caption, which is smaller than body text, and set off by position, centering, and line length.

<sup>1</sup>This is a footnote. It's even smaller, and positioned at the bottom of the page.

Titles, headings, body text, figure captions, and footnotes show how contrast is used to make articles easier to read. You can do this yourself when you're writing papers and documentation. Does this mean contrast should be maximized by using lots of different fonts like Gothic and Bookman? No, for two reasons – contrast must be balanced against simplicity, and text shape variations aren't the best way to establish contrast.



Conversely, here's a case where simplicity is taken too far, and contrast suffers. Simplicity and contrast seem to fight with each other. The standard Tukey box plot shows 5 different statistics in a single figure. But it has unnecessary lines in it! Following the principle of simplicity to its logical extreme, Edward Tufte proposed two simplifications of the box plot which convey exactly the same information – but at a great cost in contrast. Try the squint test on the Tukey plot, and on Tufte's second design. What do you see?

## Contrast Problems

Form Title - [appears above URL in most browsers and is used by WWW search]		Background Color:
Q&D Software Development Order Desk		FFFFFF00
Form Heading - [appears at top of Web page in bold type]		Text Color:
Q&D Software Development Order Desk		000080
E-Mail responses to (will not appear on dversch@q-d.com)	Alternate (for mailto forms only)	Background Graphic:
Text to appear in Submit button	Text to appear in Reset button	<input type="radio"/> Mailto
Send Order	Clear Form	<input checked="" type="radio"/> (URL)
Scrolling Status Bar Message (max length = 200 characters)		
**WebMania 1.5b with Image Map Wizard is here!!**		
<input type="button" value="Previous Tab"/>		<input type="button" value="Next Tab &gt;&gt;"/>

Source: Interface Hall of Shame

Spring 2012

6.813/6.831 User Interface Design and Implementation

24

Here's an example of too little contrast. It's important to distinguish captions from text fields, but in this design, most of the visual variables are the same for both:

- the **position** is very similar: the box around each caption and text field begins at the same horizontal position. The text itself begins at different positions (left-justified vs. aligned), but it isn't a strong distinction, and some of the captions fill their column.
- the **size** is the same: captions and text fields fill the same column width
- the background **hue** is slightly different (yellow vs. white), but not easily differentiable by the squint test
- the background **value** is the same (very bright)
- the foreground **hue** and **value** are the same (black, plain font)
- the **orientation** is the horizontal, because of course you have to read it.

The result is that it's hard to scan this form. The form is also terribly crowded, which leads us into our next topic...



## Summary

- Strive for simplicity
  - **Reduce** features and data displayed
  - **Regularize** visual properties that aren't important
  - Make elements perform **double-duty**
- Enhance contrast
  - Choose visual variables carefully
  - Use the squint test

6.813  
L17 Layout

3/21

No announcements  
(Hes' side)

Hall Fane/Shane Google Advanced Search ~~Old~~ Old vs New

Old: complex - lots of boxes  
not aligned  
- violation of simplicity  
white space important

New: (actually has since been replaced)

Hierarchy of 3 things

Can look at features people used

- But could they not find them before

Current: even simpler  
labels shorter  
help text

brought back other stuff

removed the self disclosure

- look at screen size ~~more~~  
- only displays some time

②

Middle double duty - shows you your current term  
Contrasts w/ rest of screen

---

Nansquiz Not effective for ~~shape~~ quant comparison

~~position~~

size

hue

~~value~~

value - ok for ordering, but not comparison

Least selective

Shape - hard to pick out distinct shapes

Orientation - someone selective

Simplicity

reduction

regularity

double duty

③

Today

Layout

- grouping + hierarchy
- white space
- balance + symmetry
- alignment + grid

) builds on simplicity

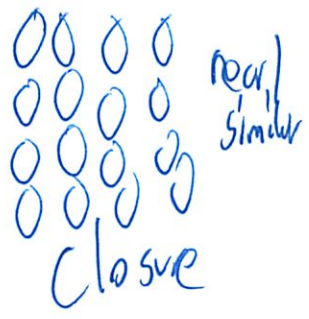
Layout implementation

Gestalt Grouping

- low level parts → see things as a whole

most important to UI design

Proximity



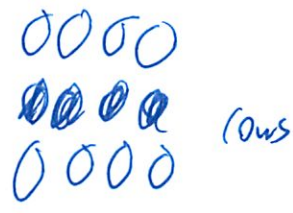
near / similar

Closure



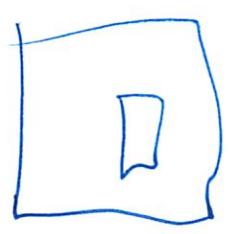
Complete contours w/ gaps

Similarity



rows

area



Small things in front of big things - look separate

Continuity



← crossing, connected not separate

Symmetry



want to see symmetry



## Grouping + Hierarchy

- Group together related items
- Make a hierarchy of importance of items

## White Space

Use white space to show proximity

- not lines

Use margins to draw eye

Integrate figure and ground

Don't crowd controls together

- creates spatial tension
- inhibits scanning

~~can~~

Ex: Old fashioned Crowded Dialog

- using contours - not white space

Minimize use of boundaries

⑤ Use white space to offset labels

a) bad example

b) good example

Make it easy to scan

a) makes it hard to scan labels

b) can focus on diff dimensions of design

---

Tufte's Designs

↓ vs needed lines / pixels

Use white space as line bars

---

Balance + Symmetry

Unless when want contrast

Choose axis

↳ usually vertical

Equalize mass on both sides

6

Hard to center > 2 items

Not centered stuff pops out

- Sharpens contrast

---

## Alignment

Align labels on left or right

Align controls on left and right <sup>↑ if short</sup>

(I basically do this on my papers)

NYT is grid based design

- align content to the cols

- news paper used it for decades

- Things can span multiple cols

Dialog box example

p 16 slides

violates grid at bottom

⑦

## Implementation

- decide ~~size~~ <sup>size</sup> + pos of everything on screen
- declarative code better than procedural
- called automatic layout
  - higher level
  - just specify constraints
- Print designers can work in inches
- But users can redesign screens
  - resize
  - diff monitors
  - widgets will differ
  - internationalization
    - Chinese short
    - German long!
- add new info/data - dynamic model



8

CSS's default is inline

- good for documents
- not for UI

Div's default is block

Can have inline-blocks

Float puts block on left/right edge

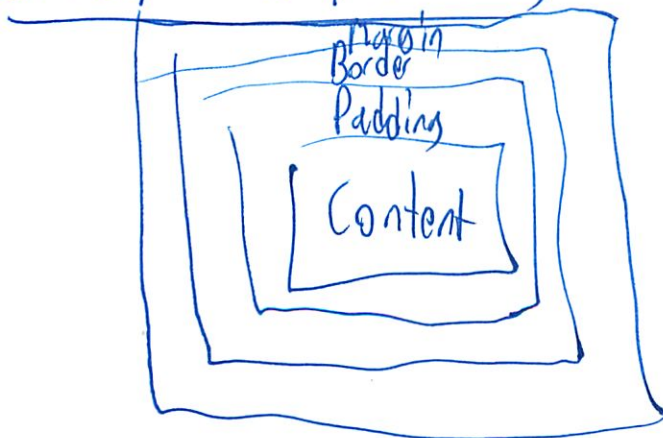
He recommends a table

- more control

Are proposals in the works for CSS Grid Layouts

Can also absolutely position stuff

Borders, Margins, Padding



9

## Space-Filling + Alignment

%

vertical-align makes a node up + down

- thinks it should all ~~the~~<sup>rest</sup> on baseline




## L17: Layout

Spring 2012

6.813/6.831 User Interface Design and Implementation

1

## UI Hall of Fame or Shame?


**Advanced Search**
[Advanced Search Tips](#) | [About Google](#)

---

**Find results**

☐ with all of the words

☐ with the exact phrase

☐ with at least one of the words

☐ without the words

10 results

---

**Language**
Return pages written in

any language

**File Format**
☒ Only ☐ return results of the file format

any format

**Date**
Return web pages updated in the

anytime

**Numeric Range**
Return web pages containing numbers between

and

**Occurrences**
Return results where my terms occur

anywhere in the page

**Domain**
☒ Only ☐ return results from the site or domain

e.g. google.com, .org [More info](#)

**Usage Rights**
Return results that are

not filtered by license [More info](#)

**Safe Search**
☒ No filtering
 ☐ Filter using SafeSearch

---

**Page-Specific Search**

**Similar**
Find pages similar to the page

**Links**
Find pages that link to the page

Spring 2010
 6.813/6.831 User Interface Design and Implementation
 2

Today's Hall of Fame and Shame is a comparison of two generations of Google Advanced Search. This is the old interface.



## UI Hall of Fame or Shame?

Google **Advanced Search** [Advanced Search Tips](#) [About Goo](#)

Use the form below, and your advanced search will appear here.

Find web pages that have...

all these words:  to

this exact wording or phrase:  to

one or more of these words:  OR  OR  to

But don't show pages that have...

any of these unwanted words:  to

Need more tools?

Results per page:  to

Language:  to

File type:  to

Search within a site or domain:

or a youtube.com site

[Date, usage, rights, numeric range, and more](#)

Spring 2010

6.813/6.831 User Interface Design and Implementation

3

And this is the new interface.

Let's compare and contrast these two interfaces in terms of:

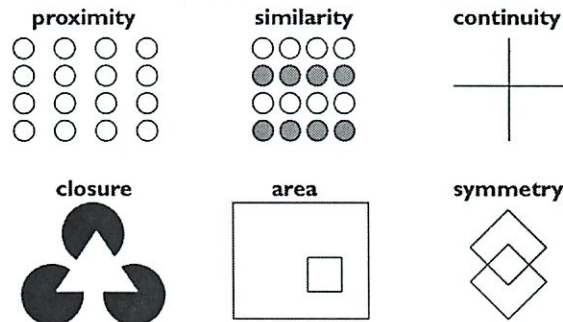
- visibility (specifically self-disclosure)
- graphic design
- task analysis
- efficiency

## Today's Topics

- Layout principles
  - Grouping & hierarchy
  - Whitespace
  - Balance & symmetry
  - Alignment & grids
- Layout implementation
  - Box, flow, grid layouts
  - Margins & padding
  - Space-filling & alignment rules

## The Gestalt Principles of Grouping

- Gestalt principles explain how the eye creates a whole (*gestalt*) from parts



Spring 2012

6.813/6.831 User Interface Design and Implementation

7

The power of white space for grouping derives from the Gestalt principle of proximity. These principles, discovered in the 1920's by the Gestalt school of psychologists, describe how early visual processing groups elements in the visual field into larger wholes. Here are the six principles identified by the Gestalt psychologists:

**Proximity.** Elements that are closer to each other are more likely to be grouped together. You see four vertical columns of circles, because the circles are closer vertically than they are horizontally.

**Similarity.** Elements with similar attributes are more likely to be grouped. You see four rows of circles in the Similarity example, because the circles are more alike horizontally than they are vertically.

**Continuity.** The eye expects to see a contour as a continuous object. You primarily perceive the Continuity example above as two crossing lines, rather than as four lines meeting at a point, or two right angles sharing a vertex.

**Closure.** The eye tends to perceive complete, closed figures, even when lines are missing. We see a triangle in the center of the Closure example, even though its edges aren't complete.

**Area.** When two elements overlap, the smaller one will be interpreted as a figure in front of the larger ground. So we tend to perceive the Area example as a small square in front of a large square, rather than a large square with a hole cut in it.

**Symmetry.** The eye prefers explanations with greater symmetry. So the Symmetry example is perceived as two overlapping squares, rather than three separate polygons.

A good paper about perceptual grouping in HCI and information visualization is Rosenholtz et al, "An Intuitive Model of Perceptual Grouping for HCI Design", CHI 2009.

## **Grouping & Hierarchy**

- Group together related items
- Make a hierarchy of importance among items



## White Space

- Use white space for grouping, instead of lines
- Use margins to draw eye around design
- Integrate figure and ground
  - Object should be scaled proportionally to its background
- Don't crowd controls together
  - Crowding creates spatial tension and inhibits scanning

Spring 2012

6.813/6.831 User Interface Design and Implementation

9

White space plays an essential role in composition. Screen real estate is at a premium in many graphical user interfaces, so it's a constant struggle to balance the need for white space against a desire to pack information and controls into a display. But insufficient white space can have serious side-effects, making a display more painful to look at and much slower to scan.

Put **margins** around all your content. Labels and controls that pack tightly against the edge of a window are much slower to scan. When an object is surrounded by white space, keep a sense of proportion between the object (the **figure**) and its surroundings (**ground**). Don't crowd controls together, even if you're grouping the controls. Crowding inhibits scanning, and produces distracting effects when two lines (such as the edges of text fields) are too close. Many UI toolkits unfortunately encourage this crowding by packing controls tightly together by default, but Java Swing (at least) lets you add empty margins to your controls that give them a chance to breathe.

## Crowded Dialog

The dialog box titled "Section" is a classic example of a crowded interface. It features several groups of controls packed closely together without margins. The "Page Number" group includes a "Start" dropdown set to "New Page", a "Page Number" section with a checked "Auto" checkbox, an unchecked "Restart at 1" checkbox, a small list box containing "1 2 3", and two text fields for "From Top" and "From Right", both set to "0.5in". The "Line Numbers" group has a "Line Numbers" section with a "By Page" dropdown, a "Count by:" text field set to "1", and a "From Text:" text field set to "Auto". The "Header/Footer" group contains two text fields for "From Top" and "From Bottom", both set to "0.5in", and an unchecked "First Page Special" checkbox. On the right side, there are four buttons: "OK", "Cancel", "Apply", and "Set Default". The overall layout is cramped, with many lines and controls overlapping or placed too close to each other.

Source: Mullet & Sano, p. 110

Spring 2012

6.813/6.831 User Interface Design and Implementation

10

Here's an example of an overcrowded dialog. The dialog has no **margins** around the edges; the controls are **tightly packed** together; and **lines are used for grouping** where white space would be more appropriate. Screen real estate isn't terribly precious in a transient dialog box.

The crowding leads to some bad perceptual effects. Lines appearing too close together – such as the bottom of the Spacing text field and the group line that surround it – blend together into a thicker, darker line, making a wart in the design. A few pixels of white space between the lines would completely eliminate this problem.

## Using White Space to Set Off Labels

Composite

Source 1

Document: concept.1

Channel: RLB

Mask

Document: concept.1

Channel: RLB

Source 2

Document: concept.1

Channel: RLB

Destination

Document: New

Channel: New

OK Cancel

(a)

Composite

Document: Channel

Source 1: concept.1 RLB

Mask: concept.1 RLB

Source 2: concept.1 RLB

Destination: New RLB

OK Cancel

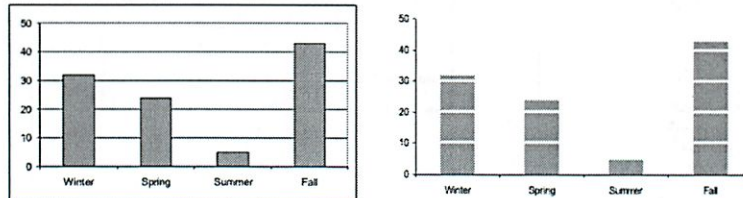
(b)

Source: Mullet & Sano, p. 96

A particularly effective use of white space is to put labels in the left margin, where the white space sets off and highlights them. In dialog box (a), you can't scan the labels and group names separately; they interfere with each other, as do the grouping lines. In the redesigned dialog (b), the labels are now alone on the left, making them much easier to scan.

For the same reason, you should put labels to the left of controls, rather than above.

## White Space Avoids Visual Noise



Spring 2012

6.813/6.831 User Interface Design and Implementation

12

Here's an interesting idea from Tufte: get rid of the grid rules on a standard bar chart, and use whitespace to show where the grid lines would cross the bars. It's much less noisy. (But alas, impossible to do automatically in Excel.)



## Balance & Symmetry

- Choose an axis (usually vertical)
- Distribute elements equally around the axis
  - Equalize both mass and extent



Spring 2012

6.813/6.831 User Interface Design and Implementation

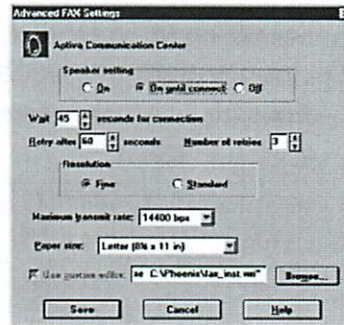
13

Balance and symmetry are valuable tools in a designer's toolkit. In graphic design, symmetry rarely means exact, mirror-image equivalence. Instead, what we mean by symmetry is more like balance: is there the same amount of stuff on each side of the axis of symmetry. We measure "stuff" by both mass (quantity of nonwhite pixels) and extent (area covered by those pixels); both mass and extent should be balanced.

An easy way to achieve balance is to simply center the elements of your display. That automatically achieves balance around a vertical axis. If you look at Google's home page, you'll see this kind of approach in action. In fact, only one element of the Google home page breaks this symmetry: the stack of links for Advanced Search, Preferences, and Language Tools on the right. This slight irregularity (a kind of **contrast**) actually helps emphasize these links slightly.

## Alignment

- Align labels on left or right
- Align controls on left *and* right
  - Expand as needed
- Align text baselines



Spring 2012

6.813/6.831 User Interface Design and Implementation

14

Finally, simplify your designs by aligning elements horizontally and vertically. Alignment contributes to the simplicity of a design. Fewer alignment positions means a simpler design. The dialog box shown has totally haphazard alignment, which makes it seem more complicated than it really is.

**Labels** (e.g. “Wait” and “Retry after”). There are two schools of thought about label alignment: one school says that the left edges of labels should be aligned, and the other school says that their right edges (i.e., the colon following each label) should be aligned. Both approaches work, and experimental studies haven’t found any significant differences between them. Both approaches also fail when long labels and short labels are used in the same display. You’ll get best results if you can make all your labels about the same size, or else break long labels into multiple lines.

**Controls** (e.g., text fields, combo boxes, checkboxes). A column of controls should be aligned on both the left and the right. Sometimes this seems unreasonable -- should a short date field be expanded to the same length as a filename? It doesn’t hurt the date to be larger than necessary, except perhaps for reducing its perceived affordance for receiving a date. You can also solve these kinds of problems by rearranging the display, moving the date elsewhere, although be careful of disrupting your design’s functional grouping or the expectations of your user.

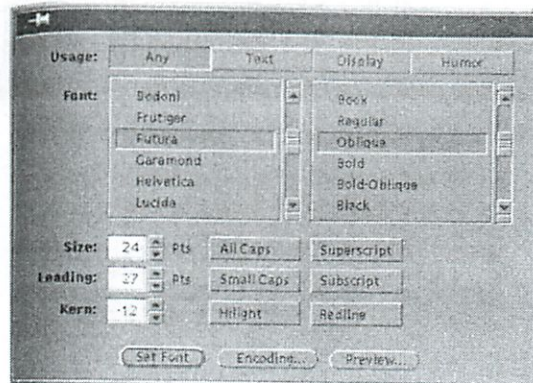
So far we’ve only discussed left-to-right alignment. Vertically, you should ensure that labels and controls on the same row share the same **text baseline**. Java Swing components are designed so that text baselines are aligned if the components are centered vertically with respect to each other, but not if the components’ tops or bottoms are aligned. Java AWT components are virtually impossible to align on their baselines. The dialog shown here has baseline alignment problems, particularly among the controls in the last row: the checkbox “Use custom editor”, the text field, and the Browse button.



A **grid** is one effective way to achieve both alignment and balance, nearly automatically. A grid means that you divide the user interface into equal-width columns (separated by gaps, and with margins on both sides of the window), and align content and controls on the column boundaries. Some elements may span multiple columns, but they align (start or end at) column boundaries.

Newspapers are famous for designing with grids, but if you look carefully at magazines, posters, and many other print designs, you'll often see a grid guiding the design.

## Grids Are Effective



Source: Mullet & Sano, p. 165

Spring 2012

6.813/6.831 User Interface Design and Implementation

16

Here's a grid in a user interface. Notice the four-column grid used in this dialog box (excluding the labels on the left). The only deviation from the grid is the row of three command buttons at the bottom which are nevertheless still balanced. In fact, their deviation from the grid helps to set them off, despite the minimal white space separating them from the rest of the display.

One criticism of this dialog is false grouping. The controls for Size, All Caps, and Superscript tend to adhere because of their proximity, and likewise for the next two rows of the display. This might be fixed by pushing the toggle buttons further to the right, to occupy columns 3 and 4 instead of 2 and 3, but at the cost of some balance.

# LAYOUT IMPLEMENTATION

Spring 2012

6.813/6.831 User Interface Design and Implementation

17



## Automatic Layout

- **Layout** determines the sizes and positions of components on the screen
  - Also called geometry in some toolkits
- Declarative layout
  - CSS styles
- Procedural layout
  - Write Javascript code to compute positions and sizes

Spring 2011

6.813/6.831 User Interface Design and Implementation

18

In HTML/CSS, automatic layout is a declarative process. First you specify the graphical objects that should appear in the window, which you do by creating instances of various objects and assembling them into a view tree. We've seen how HTML does this. Then you specify how they should be laid out by attaching styles.

You can contrast this to a procedural approach to layout, in which you write Javascript code that computes positions and sizes of objects in the view tree.

## Reasons to Do Automatic Layout

- Higher level programming
  - Shorter, simpler code
- Adapts to change
  - Window size
  - Font size
  - Widget set (or theme or skin)
  - Labels (internationalization)
  - Adding or removing nodes

Spring 2011

6.813/6.831 User Interface Design and Implementation

19

Here are the two key reasons why we like automatic layout – and these two reasons generalize to other forms of declarative UI as well.

First, it makes programming **easier**. The code that sets up layout managers is usually much simpler than procedural code that does the same thing.

Second, the resulting layout can **respond to change** more readily. Because it is generated automatically, it can be *regenerated* any time changes occur that might affect it. One obvious example of this kind of change is resizing the window, which increases or decreases the space available to the layout. You could handle window resizing with procedural code as well, of course, but the difficulty of writing this code means that programmers generally *don't*. (That's why many Windows dialog boxes, which are often laid out using absolute coordinates in a GUI builder, refuse to be resized! A serious restriction of user control and freedom, particularly if the dialog box contains a list or file chooser that would be easier to use if it were larger.)

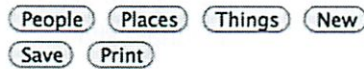
Automatic layout can also automatically adapt to font size changes, different widget sets (e.g., buttons of different size, shape, or decoration), and different labels (which often occur when you translate an interface to another language, e.g. English to German). These kinds of changes tend to happen as the application is moved from one platform to another, rather than dynamically while the program is running; but it's helpful if the programmer doesn't have to worry about them.

Another dynamic change that automatic layout can deal with is the appearance or disappearance of nodes from the view tree-- if the user is allowed to add or remove buttons from a toolbar, for example, or if new textboxes can be added or removed from a search query.

## Flow Layout

- Left-to-right, automatically-wrapping
- CSS calls this “inline” layout  
display: inline
- Many elements use inline layout by default

```
<button>People</button>  
<button>Places</button>  
<button>Things</button>  
<button>New</button>  
<button>Save</button>  
<button>Print</button>  
...
```



Spring 2012

6.813/6.831 User Interface Design and Implementation

20

```
<button>People</button>
```

```
<button>Places</button>
```

```
<button>Things</button>
```

```
<button>New</button>
```

```
<button>Save</button>
```

```
<button>Print</button>
```

Author:

```
<input type="text" />
```

Comment:

```
<textarea></textarea>
```

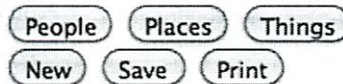
```
<button>OK</button>
```

```
<button>Cancel</button>
```

## Box Layout

- Blocks are laid out vertically
  - display: block
  - divs default to block layout
- Inline blocks are laid out in flow
  - display: inline-block

```
<div>  
<button>People</button>  
<button>Places</button>  
<button>Things</button>  
</div>  
<div>  
<button>New</button>  
<button>Save</button>  
<button>Print</button>  
</div>
```

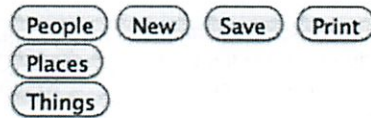


## Float Layout

- Float pushes a block to left or right edge

```
<style>
.navbar { float: left; }
.navbar button { display: block; }
</style>
```

```
<div class="navbar">
<button>People</button>
<button>Places</button>
<button>Things</button>
</div>
```



```
<div>
<button>New</button>
<button>Save</button>
<button>Print</button>
</div>
```



## Grid Layout

- Blocks & floats are typically not enough to enforce all the alignments you want in a UI
- CSS grid layout is coming but not quite here
- For now, use tables for 2D alignment instead

```
<table>
<tr><td>Name:</td>
    <td><input type="text" /></td>
</tr>
<tr><td>Groups:</td>
    <td><textarea></textarea></td>
</tr>
</table>
```

Name:

Groups:

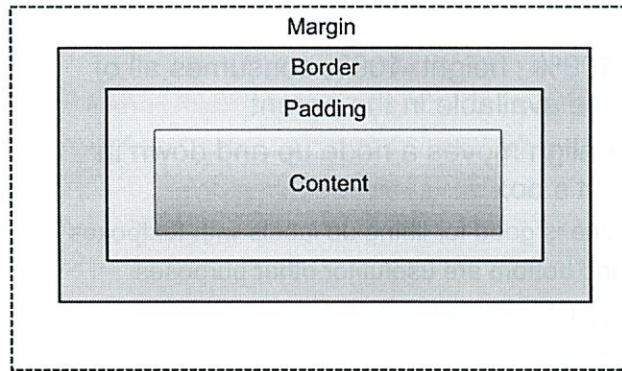
## Absolute Positioning

- Setting position & size explicitly
  - in coordinate system of entire window, or of node's parent
  - CSS has several units: px, em, ex, pt
  - mostly useful for popups

```
<style>  
button { position: absolute;  
          left: 5px;  
          top: 5px; }  
</style>
```

[Print](#) [js](#) the user interface goes here

## Margins, Borders, & Padding



## Space-Filling & Alignment

- width: 100% , height: 100% consumes all of the space available in the parent
- vertical-align moves a node up and down in its parent's box
  - baseline is good for lining up labels with textboxes
  - top and bottom are useful for other purposes
- Centering
  - margin: auto for boxes
  - text-align: center for inlines

## Summary

- Layout should establish grouping of items
- Use whitespace & alignment to preserve simplicity
- Automatic layout adapts to changes in UI



G12/G13 review

Presentation is part of grade

- Hard to read
- Sep links per thing
- Whole assignment on 1 pg

Document not readable

Groups had grid-like structure



	<u>Learning</u>	<u>Efficiency</u>	<u>Security</u>
P.	—	—	—
C.	—	—	—

G13 - see reasons for iteration

Q: Asked about log in

- not pt of product

②

Or critically improve visibility

---

I Suggested more structured testing

---

How much info you pre do  
How you run  
Consistency

6.813 L18

3/23

Color

Hall Fame/Ghane: CD creator

- the check / cross same color are the same
  - L "squint test"
- 

Wano quiz

Automatic layout

- Uses declarative ~~program~~ programming
- tolerant
- CSS

Gestalt this slide

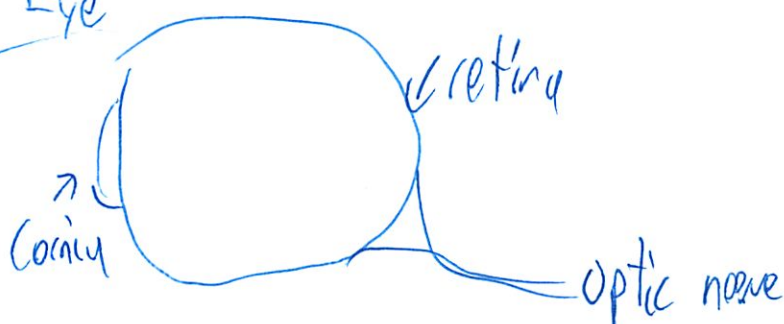
similarity  
proximity  
not hierarchy

(2)

Today: Color

- vision
  - color models
  - limitations
- 

The Eye



Photoreceptors

Rods

low light

only 1 kind

freq response centered around green

Cones

normal light

color

3 kinds

Shorter - weak freq response - blue  
medium - green  
long - yellow (red)

③ Things are not distributed well

Blind spot where optic nerve hits

↳ we fill it in psychology

### Signals

Brightness

Red-green difference

Blue-yellow difference

### Color Blind

8-10% male

0.5% female

Three diff kinds

- Protanopia - missing L cones

- Deutanopia - red green differences  
M cones

~~Ambloupe~~

- Tritanopia - smalling short cones

So don't rely on it solely



④

So is traffic light - people know the position  
Online tools to check [vischeck.com](http://vischeck.com)

Red green tests in slides

Cube of colors

- losing red-green

---

### Chromatic Abberation

don't use red on blue text

Blue details in general hard to resolve

- blue starts to yellow as you get older

- esp on dark bg

---

### Color Constancy (image)

A, B regions same!

But we perceive diff colors differently,

Things look diff w/ diff light

⑤

Check w/ eye dropper tool

Color Models

How we represent colors

RGB - used on screens (RT, LCD)

CMYk - k = key = black  $\Rightarrow$  printing

HSV - How we perceive color

Color match game

Accurate Color

CIE 1936 gamut

What colors CRT can show

(lots of WP entries)

A display produces color on plane

3 corners  $\rightarrow$  primary colors for display

6

Diff devices have diff gamut

~~Color~~ Some colors are outside the gamut

Are other color spaces

- diff amt of space for each color

- more to perceptual

---

### Guidelines

- Avoid saturated colors

- Use few colors

- Be consistent w/ expectations

- eyes perceive stuff differently  
like w/ sunglasses

- Use <sup>saturation</sup> sparingly

- instead pastels

- Epcot: sidewalk pink so grass looks greener

① Green looks greener after looking at red

---

Use few colors in icons

Adding a little bit of red makes things stick out more

---

BG colors

- pale can be easier to look at
- want contrast w/ text
- dark bgs tricky → need contrast!

---

Red = stop, error, hot  
often culturally dep

---

So can pick 1 color and several shades of grey  
Rob Miller likes fire bug  
Or pick colors from photos

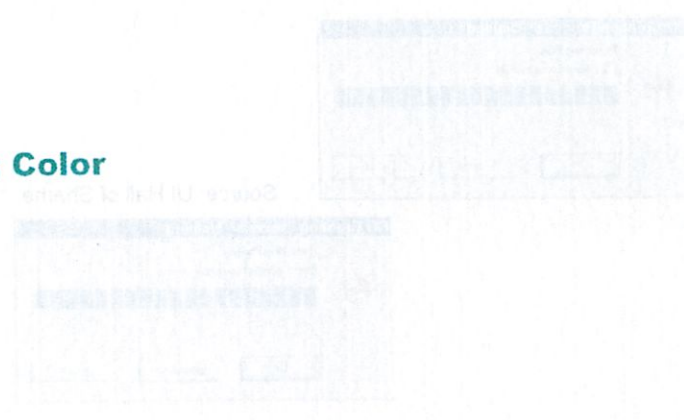


4/30

# UI Hall of Fame or Shame?

## L18: Color

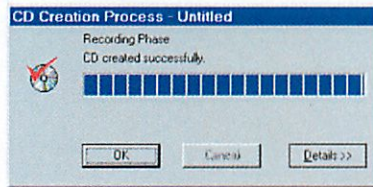
Source: UI Hall of Shame



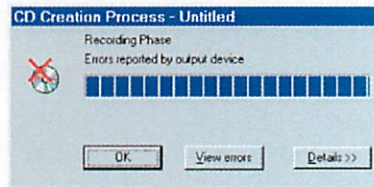
Spring 2012      6.813/6.831 User Interface Design and Implementation      1



## UI Hall of Fame or Shame?



Source: UI Hall of Shame



Spring 2012

6.813/6.831 User Interface Design and Implementation

2

Our Hall of Shame candidate for the day is this dialog box from Adaptec Easy CD Creator, which appears at the end of burning a CD. The top image shows the dialog when the CD was burned successfully; the bottom image shows what it looks like when there was an error.

What does the squint test tell you about these dialogs?

The key problem is the **lack of contrast** between these two states. Success or failure of CD burning is important enough to the user that it should be obvious at a glance. But these two dialogs look identical at a glance. How can we tell? Use the **squint test**, which we talked about in the graphic design lecture. When you're squinting, you see some labels, a big filled progress bar, a roundish icon with a blob of red, and three buttons. All the details, particularly the text of the messages and the exact shapes of the red blob, are fuzzed out. This simulates what a user would see at a quick glance, and it shows that the graphic design doesn't convey the contrast.

One improvement would change the check mark to another color, say green or black. Using red for OK seems **inconsistent** with the real world, anyway. But designs that differ only in red and green wouldn't pass the squint test for color-blind users.

Another improvement might remove the completed progress bar from the error dialog, perhaps replacing it with a big white text box containing a more detailed description of the problem. That would clearly pass the squint test, and make errors much more noticeable.

## Today's Topics

- Color
  - Human vision
  - Color models
  - Design guidelines

Spring 2012

6.813/6.831 User Interface Design and Implementation

5

Today's lecture is about choosing colors for a user interface. We'll discuss some of the properties of **human vision** that affect this decision, particularly the limitations of color vision. We'll go over some **models** for representing colors, not just the familiar RGB model. And we'll discuss some **guidelines** for choosing colors. The most important guidelines will be applications of rules we already discussed in **graphic design**: **simplicity** as much as possible, **contrast** where important.

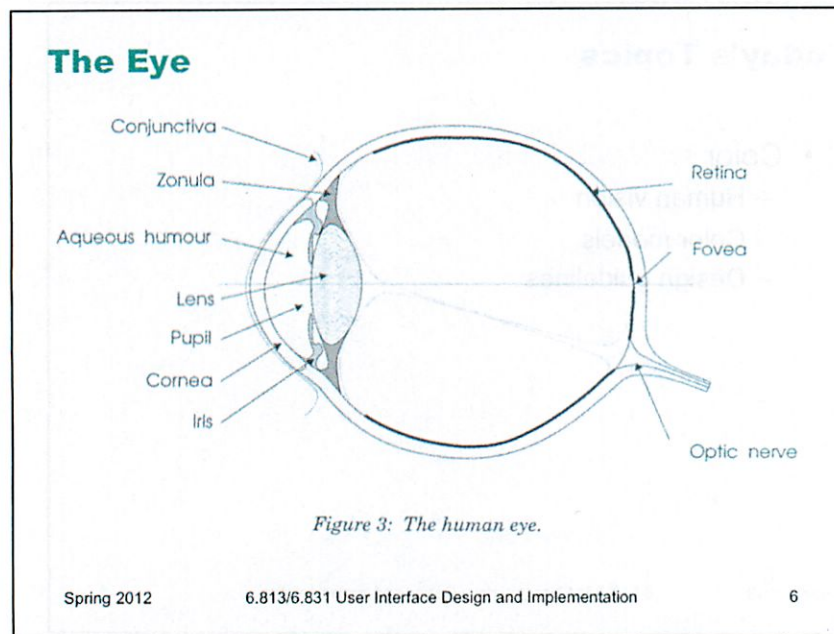
A good reference about color is Colin Ware, *Information Visualization: Perception for Design*, Morgan Kaufmann, 2000 (which we also mentioned in the graphic design lecture).

An aside about why we're learning this. This lecture is effectively part 2 of the graphic design lecture, and many of the guidelines in both lectures are more about aesthetics than pure usability. Serious *mistakes* in graphic design certainly affect usability, however, so we're trying to help you avoid those pitfalls. But there's a larger question here: in practice, should software engineers have to learn this stuff at all? Shouldn't you just hire a graphic designer and let them do it? Some people who teach UI to CS students think that the most important lesson a software engineer can learn from a course like this is "UI design is hard; leave it to the experts." They argue that a little knowledge can be a dangerous thing, and that a programmer with a little experience in UI design but too much self-confidence can be just as dangerous as an artist who's learned a little bit of HTML and thinks they now know how to program. But I prefer to believe that a little knowledge is a step on the road to greater knowledge. Some of you may decide to *become* UI designers, and this course is a step along that road.

In a commercial environment, you *should* hire experienced graphic designers, just as you should hire an architect for building your corporation's headquarters and you should contract with a licensed building firm. Big jobs for big bucks require experts who have focused their education and job experience on those problem. One reason this course is useful is that you can appreciate what UI experts do and evaluate their work, which will help you work on a team with them (or supervise them).

But it's also worth learning these principles because you can apply them yourself on smaller-scale problems. Are you going to hire a graphic designer for every PowerPoint presentation you make, every chart you draw, every web page you create? Those are all user interfaces. Many problems in life have a user interface, and many of them are up to you to do-it-yourself. So you should know when to leave it to the experts, but you should be able to do a creditable job yourself too, when the job is yours to do.





Here are key parts of the anatomy of the eye:

- The **cornea** is the transparent, curved membrane on the front of the eye.
- The **aqueous humor** fills the cavity between the cornea and the lens, and provides most of the optical power of the eye because of the large difference between its refractive index and the refractive index of the air outside the cornea.
- The **iris** is the colored part of the eye, which covers the lens. It is an opaque muscle, with a hole in the center called the **pupil** that lets light through to fall on the lens. The iris opens and closes the pupil depending on the intensity of light; it opens in dim light, and closes in bright light.
- The **lens** focuses light. Under muscle control, it can move forward and backward, and also get thinner or fatter to change its focal length.
- The **retina** is the surface of the inside of the eye, which is covered with light-sensitive receptor cells.
- The **fovea** is the spot where the optical axis (center of the lens) impinges on the retina. The highest density of photoreceptors can be found in the fovea; the fovea is the center of your visual field.

Figure from Lilley, Lin, Hewitt, & Howard, "Colour in Computer Graphics", University of Manchester.

## Photoreceptors

- Rods
  - Only one kind (peak response in green wavelengths)
  - Sensitive to low light ("scotopic vision")
    - Multiple nearby rods aggregated into a single nerve signal
  - Saturated at moderate light intensity ("photopic vision")
    - Cones do most of the vision under photopic conditions
- Cones
  - Operate in brighter light
  - Three kinds: S(hort), M(edium), L(ong)
  - S cones are very weak, centered in blue wavelengths
  - M and L cones are more powerful, overlapping
  - M centered in green, L in yellow (but called "red")

Spring 2012

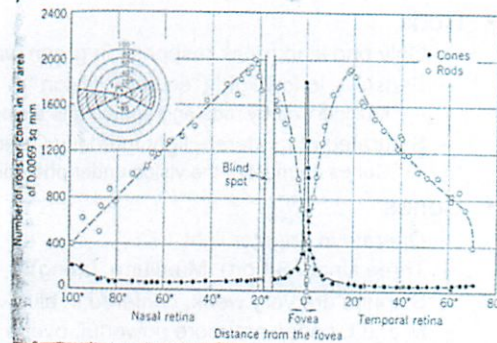
6.813/6.831 User Interface Design and Implementation

7

There are two kinds of photoreceptor cells in the retina. **Rods** operate under low-light conditions – night vision. There is only one kind of rod, with one frequency response curve centered in green wavelengths, so rods don't provide color vision. Rods saturate at moderate intensities of light, so they contribute little to daytime vision. **Cones** respond only in brighter light. There are three kinds of cones, called S, M, and L after the centers of their wavelength peaks. S cones have very weak frequency response centered in blue. M and L cones are two orders of magnitude stronger, and their frequency response curves nearly overlap.

## Rods & Cones

- Cones are mostly in the center of the eye.
- Rods are mostly on the edges.



Spring 2012

6.813/6.831 User Interface Design and Implementation

8

Rods and cones aren't distributed evenly in the eye.

Most of the cones are in the center, most of the rods are on the edge.

Rods, since they're on the edge and pick up very coarse-grained information, are what we use for most of our peripheral vision. In fact, if you're trying to spot shooting stars at night, you often pick out more things in your peripheral vision.

You can also see the blind spot in this poorly-scanned photo (from <http://www.unc.edu/~ejw/rod-cone-dist.html>). The blind spot is where the optic nerve is located on the back of the eye.



## Signals from Photoreceptors

- Brightness  
 $M + L + \text{rods}$
- Red-green difference  
 $L - M$
- Blue-yellow difference  
weighted sum of S, M, L

Spring 2012

6.813/6.831 User Interface Design and Implementation

9

The rods and cones do not send their signals directly to the visual cortex; instead, the signals are recombined into three channels. One channel is **brightness**, produced by the M and L cones and the rods. This is the only channel really active at night. The other two channels convey color **differences**, red-green and blue-yellow. For the red-green channel, for example, high responses mean red, and low responses indicate green.

These difference channels drive the theory of **opponent colors**: red and green are good contrasting colors because they drive the red-green channel to opposite extremes. Similarly, black/white and blue/yellow are good contrasting pairs.

## Color Blindness

- Red-green color blindness (protanopia & deuteranopia)
  - 8% of males
  - 0.4% of females
- Blue-yellow color blindness (tritanopia)
  - Far more rare (~50 people in a million)
- Guideline: don't depend solely on color distinctions
  - use redundant signals: brightness, location, shape

Spring 2012

6.813/6.831 User Interface Design and Implementation

10

Color deficiency (“color blindness”) affects a significant fraction of human beings. An overwhelming number of them are male.

There are three kinds of color deficiency, which we can understand better now that we understand a little about the eye's anatomy:

• **Protanopia** is missing or bad L cones. The consequence is reduced sensitivity to red-green differences (the L-M channel is weaker), and reds are perceived as darker than normal.

• **Deuteranopia** is caused by missing or malfunctioning M cones. Red-green difference sensitivity is reduced, but reds do not appear darker.

• **Tritanopia** is caused by missing or malfunctioning S cones, and results in blue-yellow insensitivity.

Red/green color blindness affects about 8% of males and 0.4% of females; blue/yellow color blindness is much much rarer.

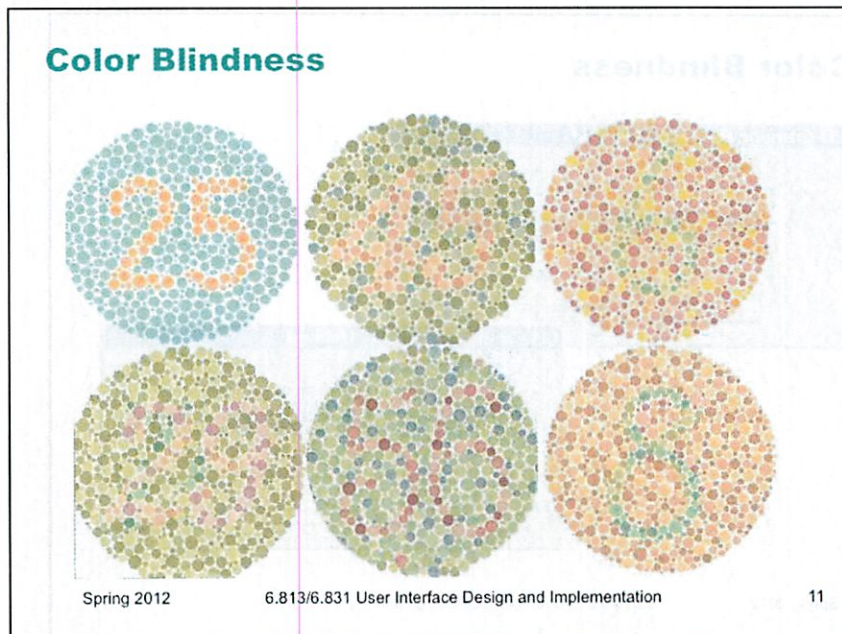
But since color blindness affects so many people, it is essential to take it into account when you are deciding how to use color in a user interface. Don't depend solely on color distinctions, particularly red-green distinctions, for conveying information. Microsoft Office applications fail in this respect: red wavy underlines indicate spelling errors, while identical green wavy underlines indicate grammar errors.

Traffic lights are another source of problems. How do red-green color-blind people know whether the light is green or red? Fortunately, there's a spatial cue: red is always above (or to the right of) green. Protanopia sufferers (as opposed to deuteranopians) have an additional advantage: the red light looks darker than the green light.

There are online tools for checking your interface against various kinds of color blindness; one good one is Vischeck (<http://www.vischeck.com/vischeck/>).

Henry Sturman is a red-green colorblind software developer who has written a good article about what it's like (<http://henrysturman.com/english/articles/colorvision.html>).



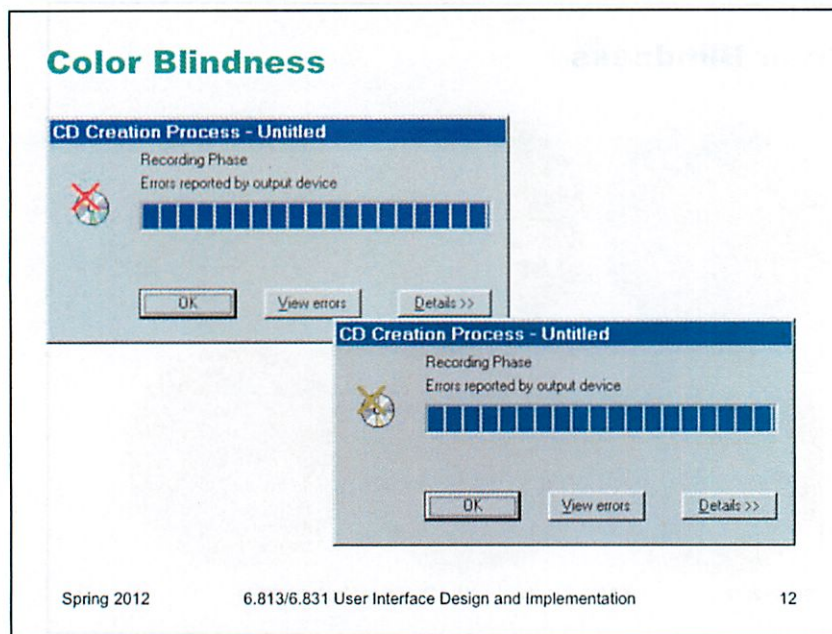


One of the most common ways to detect colorblindness is using the Ishihara plates. Let's look at a few – can you see the numbers hidden in each of the circles? These test red-green colorblindness.

If you're colorblind, you can't see the bottom of the left pair, the top of the middle pair, and neither of the right pair.

Don't depend solely on color distinctions, particularly red-green distinctions, for conveying information. Microsoft Office applications fail in this respect: red wavy underlines indicate spelling errors, while identical green wavy underlines indicate grammar errors.

Traffic lights are another source of problems. How do red-green color-blind people know whether the light is green or red? Fortunately, there's a spatial cue: red is always above (or to the right of) green. Protanopia sufferers (as opposed to deuteranopians) have an additional advantage: the red light looks darker than the green light.

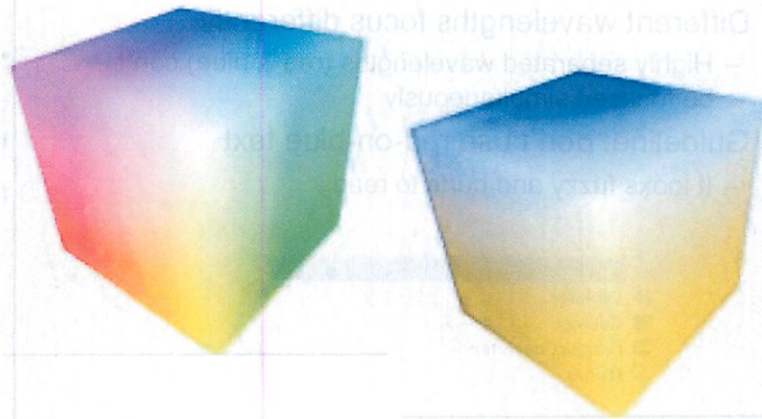


There are online tools for checking your interface against various kinds of color blindness; one good one is Vischeck (<http://www.vischeck.com/vischeck/>).

Henry Sturman is a red-green colorblind software developer who has written a good article about what it's like (<http://henrysturman.com/english/articles/colorvision.html>).

Here we can see the hall of fame or shame example.

## Color Blindness



Spring 2012

6.813/6.831 User Interface Design and Implementation

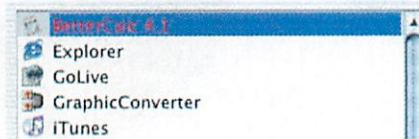
13

And this cube of colors basically loses the red and green!



## Chromatic Aberration

- Different wavelengths focus differently
  - Highly separated wavelengths (red & blue) can't be focused simultaneously
- Guideline: don't use red-on-blue text
  - It looks fuzzy and hurts to read



Spring 2012

6.813/6.831 User Interface Design and Implementation

14

The refractive index of the lens varies with the wavelength of the light passing through it; just like a prism, different wavelengths are bent at different angles. So your eye needs to focus differently on red features than it does on blue features.

As a result, an edge between widely-separated wavelengths – like blue and red – simply can't be focused. It always looks a little fuzzy. So blue-on-red or red-on-blue text is painful to read, and should be avoided at all costs.

Apple's ForceQuit tool in Mac OS X, which allows users to shut down misbehaving applications, unfortunately fell into this trap. In the dialog, unresponsive applications are helpfully displayed in red. But the selection is a blue highlight. The result is incredibly hard to read.

Here's an experiment you can try to demonstrate chromatic aberration. Put a small purple dot on a piece of paper. Hold it close to your eye; it should look blue in the center, surrounded by a red halo. Then move it farther away; the colors should switch places, so that red is in the center and blue is the halo.

## Blue Details Are Hard to Resolve

- Fovea has few S cones
  - Can't resolve small blue features (unless they have high contrast with background)
- Lens and aqueous humor turn yellow with age
  - Blue wavelengths are filtered out
- Lens weakens with age
  - Blue is harder to focus
- Guideline: don't use blue against dark backgrounds where small details matter (text!)

Spring 2012

6.813/6.831 User Interface Design and Implementation

15

A number of anatomical details conspire to make blue a bad color choice when small details matter.

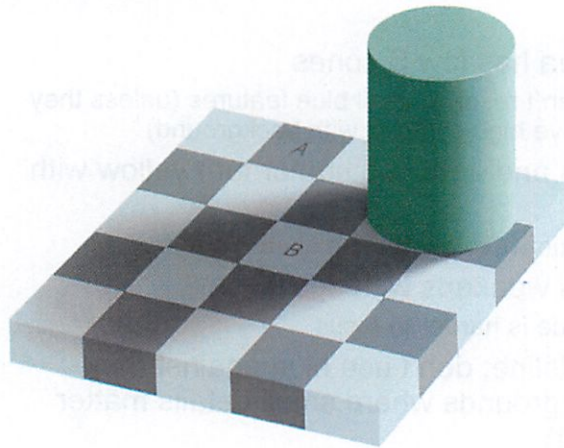
First, the fovea has very few S cones, so you can't easily see blue features in the center of your vision (unless they have high contrast with the background, activating the M and L cones).

Second, older eyes are far less sensitive to blue, because the lens and aqueous humor slowly grow yellower, filtering out the blue wavelengths.

Finally, the lens gets weaker with age. Blue is at one extreme of its focusing range, so older eyes can't focus blue features as well.

As a result, avoid blue text, particularly small blue text.

## Color Constancy



Spring 2012

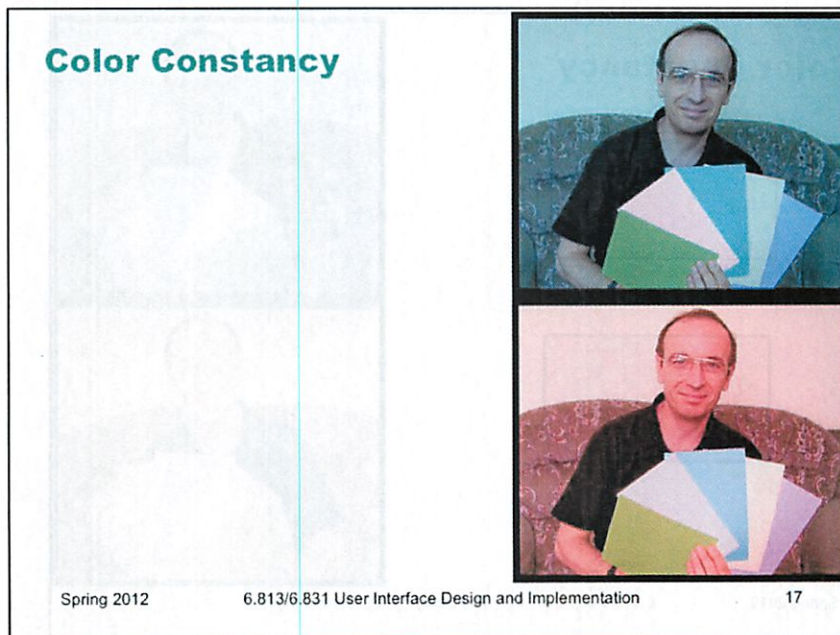
6.813/6.831 User Interface Design and Implementation

16

Color constancy is another thing that can modify our color perceptions.

You've probably seen this image before. The color of A and the color of B are identical, but our brain sees them as different because of the shadow. We perceive colors as constant, if we have reason to, even if they are under different light.



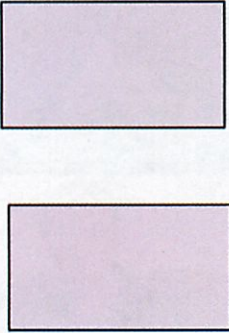


You can see an example of this in these two images. Do any of those sheets of paper look like the same color?

We see the red lighting of the second picture and adapt our assumptions about the colors accordingly. In reality, the second sheet from the left is the same color in both pictures.

We can explore this using an eyedropper tool or the Digital Color Meter in OS X or your OS.

### Color Constancy



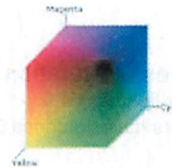
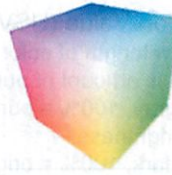
Spring 2012      6.813/6.831 User Interface Design and Implementation      18

These two rectangles are rectangles made (using the eyedropper in color selection) from the second sheet from each photo.



## Color Models

- Red-Green-Blue (RGB)
  - Red: 0% - 100%
  - Green: 0% - 100%
  - Blue: 0% - 100%
- Cyan-Magenta-Yellow
  - Used for printing



Spring 2012

6.813/6.831 User Interface Design and Implementation

19

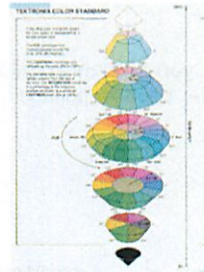
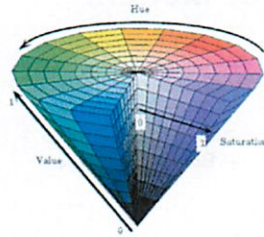
Now let's look at how colors are represented in GUI software. At the lowest level, the RGB model rules. The RGB model is a unit cube, with (0,0,0) corresponding to black, (1, 1, 1) corresponding to white, and the three dimensions measuring levels of red, green, and blue. The RGB model is used directly by CRT and LCD monitors for display, since each pixel in a monitor has separate red, green, and blue components.

The CMYK (cyan, magenta, yellow, and sometimes black) is similar to the RGB model, but used for print colors, where pigments absorb wavelengths instead of generating them.

K stands for key.

## More Color Models

- Hue-Saturation-Value (HSV)
  - Hue is wavelength of color
  - Saturation is amount of pure color
    - 0% = gray, 100% = pure
  - Value is brightness
    - 0% = dark, 100% = bright
  
- Hue-Lightness-Saturation (HLS)
  - White has lightness 1.0
  - Pure colors have lightness 0.5



Spring 2012

6.813/6.831 User Interface Design and Implementation

20

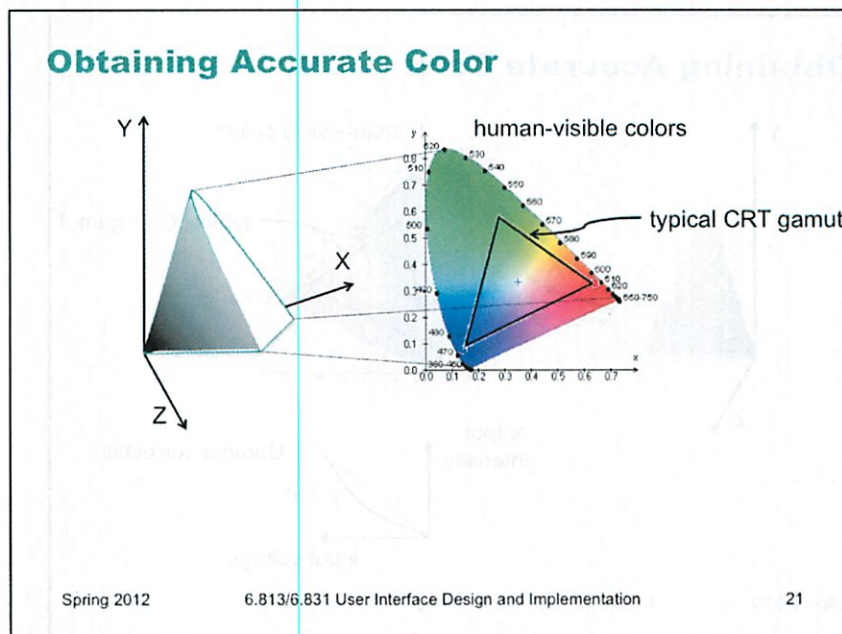
HSV (hue, saturation value) is a better model for how humans perceive color, and more useful for choosing colors in user interface design. HSV is a cone. We've already encountered hue and value in our discussion of visual variables. Saturation is the degree of color, as opposed to grayness. Colors with zero saturation are shades of gray; colors with 100% saturation are pure colors.

HLS (hue, lightness, saturation) is a symmetrical relative of the HSV model, which is elegant. It basically pulls up the center of the HSV cone to make a double-ended cone.

Many applications have a color picker that lets you pick HSV values as an alternative to RGB.

Let's look at a website that helps you test your color matching skills across hue, saturation, and value.

<http://color.method.ac>



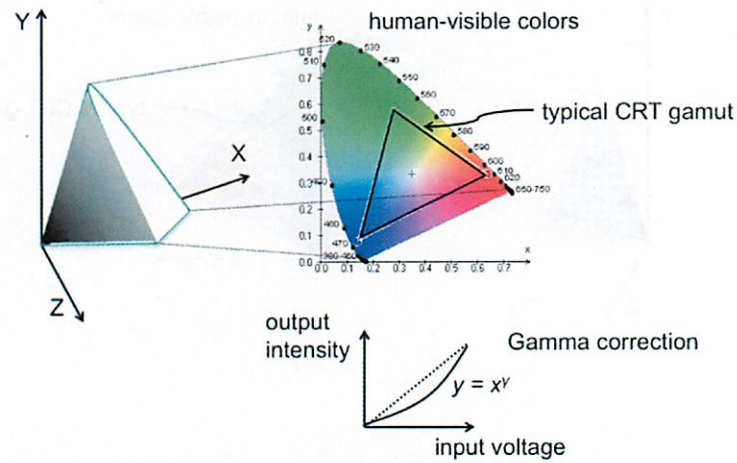
Although RGB and HSV are commonly used for implementing graphical user interfaces, they are not in fact *standardized*. The way that a color like pure red (RGB=1,0,0) actually looks depends strongly on the display's characteristics, so your application can't be sure it will get exactly the right color.

The science of colorimetry is concerned with accurate measurement and reproduction of color. Most of it is outside the scope of this course, but here are a few things you should know. Colorimetry starts with a 3D space based on three primary colors, called XYZ, chosen so that all human-visible colors are bounded within the positive octant -- so that any visible color can be made as a mix of positive amounts of X, Y, and Z. The solid area on the left shows the visible colors perceivable to the human eye; black is of course at the origin. Note that X, Y, and Z are *imaginary* colors in the sense that they cannot be produced by a physical light source or perceived by the human eye; but they're useful bases for the space, much like imaginary numbers are useful. To consider hue and saturation in isolation, we look at a plane of constant intensity, shown on the right.

The wedge-shaped figure shows the whole space of human-perceivable colors (with fully-saturated, pure-wavelength colors around its perimeter).



## Obtaining Accurate Color



Spring 2012

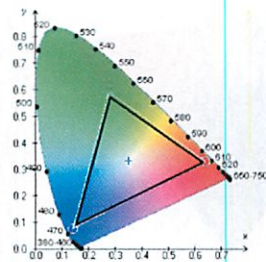
6.813/6.831 User Interface Design and Implementation

22

Any given display device can produce some triangle of colors on this plane (called the device's **gamut**), where the corners are the three colors used by the device as its primary colors – e.g., the exact colors of red, green, and blue in a CRT's phosphors. The triangle here shows a typical cathode-ray television's gamut. Devices with different gamuts will produce different colors from the same RGB value. This problem can be addressed by calibrating the display device, producing an ICC profile that specifies how the device's RGB space maps into a standardized space like XYZ.

Different devices have different gamuts, so if you make something that uses a specific range of colors, some devices may not be able to see this.

## Color Gamuts and YOU



Spring 2012

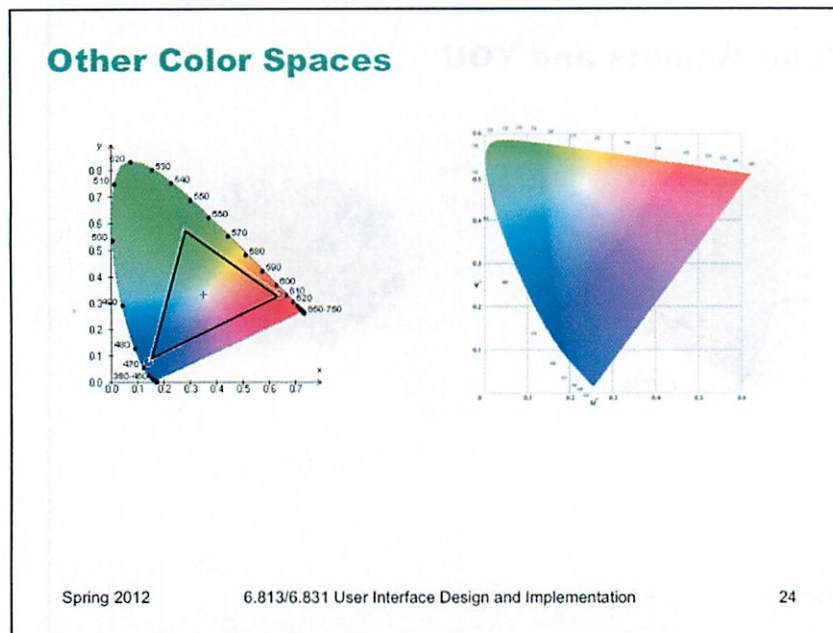
6.813/6.831 User Interface Design and Implementation

23

Different devices have different gamuts, so if you make something that uses a specific range of colors, some devices may not be able to see this.

This is a picture of some ultramarine pigment, which is my favorite color, International Klein Blue. While the picture looks pretty intense, odds are good it's not actually reproducing what it looks like in person, because it's outside of the gamut of my computer. The projector doesn't help either – and if you look at it on your computer it probably looks slightly different too.





This is a comparison of the CIE 1931 model on the left, and the CIE LUV model on the right.

CIE is the International Committee on Illumination. The new model, which is one of two current standards, is easier to compute, and is used more frequently in computer graphics. One of the goals of this new model was to have the sizes of each area better conform to our perceptual mappings.

Other standardized color spaces exist. One drawback of XYZ is that it's not perceptually uniform – green occupies a huge chunk at the top of the wedge, while yellow is a narrow little line; it would be preferable if the distance in color space produced the same difference in perception in both areas. LUV is an alternative model that addresses that by distorting the projection. But neither XYZ or LUV is particularly useful for programming because they're imaginary colors; sRGB aims to fix that by standardizing the RGB color space instead (<http://www.w3.org/Graphics/Color/sRGB.html>).

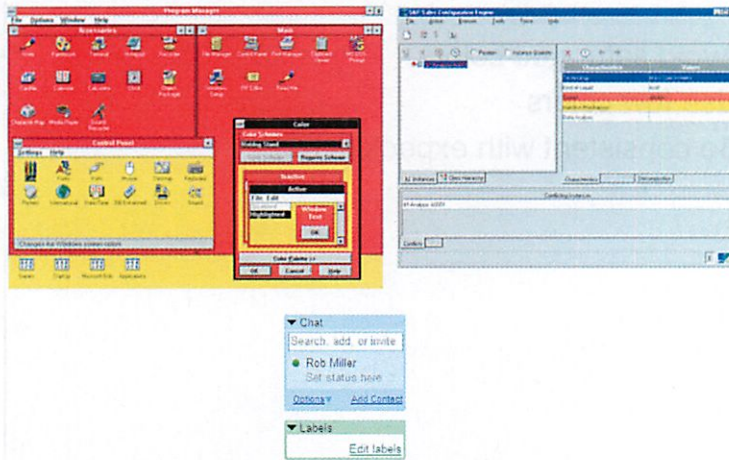
Another issue in accurate color reproduction is the intensity (value) of the color. Different display devices have different response curves. When the red component of an RGB value (0-1) is mapped directly to a voltage applied to an electron gun, the intensity of light produced does not vary linearly from 0 to 1, but typically follows a power curve ( $y=x^{\gamma}$ , for some  $\gamma > 1$ ). **Gamma correction** is the process of standardizing the intensity so that a linear response is obtained.

All these issues also apply to cameras as well as displays, of course. Cameras have gamuts (for the hues and saturations they can record) and response curves (to intensity) that require calibration and correction. To learn more about human perception and color, take 6.098/6.882 Computational Photography.

## Color Guidelines

- Avoid saturated colors
- Use few colors
- Be consistent with expectations

## Avoid Saturated Colors



Spring 2012

6.813/6.831 User Interface Design and Implementation

26

In general, avoid strongly saturated colors – i.e., the colors around the outside edge of the HSV cone. Saturated colors can cause visual fatigue because the eye must keep refocusing on different wavelengths. They also tend to saturate the viewer's receptors (hence the name). One study found that air traffic controllers who viewed strongly saturated green text on their ATC interfaces for many hours had trouble seeing pink or red (the other end of the red/green color channel) for up to 15 minutes after their shift was over.

Use less saturated, “pastel” colors instead, which mix gray or white into the pure color.

The examples on top use colors with high saturation; on the bottom, low saturation. Shades of gray have minimum saturation.



### When saturation is cool

- When your cones get saturated with one color, you can use that to make other colors seem brighter.
- Epcot in Disney World uses this – they make their sidewalks pink-tinted, which slowly fatigues your eyes, so the grass looks greener.

Spring 2012

6.813/6.831 User Interface Design and Implementation

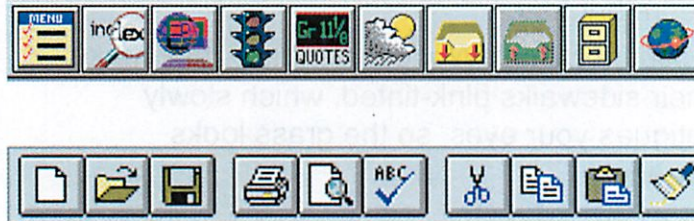
27

Saturating one color in your vision can have a temporary effect on your perception of other colors. Think of when you take off colored sunglasses, for instance. You can use this to change perception in cool ways – for instance, making grass look greener – but it can also be detrimental.

To see what this feels like, look at a full screen of bright red for a little bit. Then look at a full screen of green. The green that you see will be more green than you can normally perceive, because your red-perceiving cones won't be firing at all. This green is an imaginary color. It's visible light, but it's greener than the range of normal human vision.



## Use Few Colors



Spring 2012

6.813/6.831 User Interface Design and Implementation

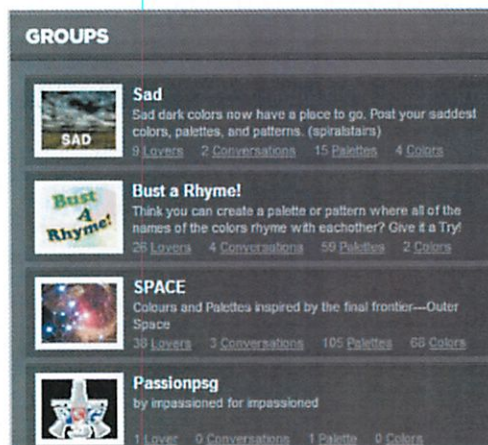
28

In general, colors should be used sparingly. An interface with many colors appears more complex, more cluttered, and more distracting. Use only a small number of different hues.

The toolbar on top uses too many colors (many of them highly saturated), so none of the buttons stand out, and the toolbar feels hard to scan. In contrast, the toolbar at the bottom uses only a handful of colors. It's more restful to look at, and the buttons that actually use color (like the Open File button) really pop out.

A simple and very effective color scheme uses just one hue (like blue or green, weakly saturated and in various values), combined with black, white, and shades of gray. On top of a scheme like that, a bit of red in an icon will pop out wonderfully.

## Background Colors



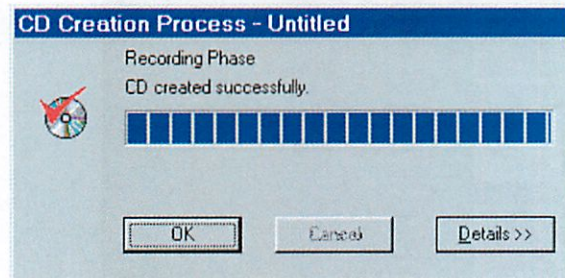
Spring 2012

6.813/6.831 User Interface Design and Implementation

29

Background colors should establish a good contrast with the foreground. White is a good choice, since it provides the most contrast; but it also produces bright displays, since our computer displays emit light rather than reflecting it. Pale (desaturated) yellow and very light gray are also good background colors. Dark backgrounds are tricky; it's too easy to mess up the contrast and make text less legible, as shown in this example.

## Be Consistent With Expectations



Spring 2012

6.813/6.831 User Interface Design and Implementation

30

Finally, match expectations. One of the problems with the Adaptec dialogs at the beginning of this lecture was the use of red for OK. Red generally means stop, warning, error, or hot. Green conventionally means go, or OK. Yellow means caution, or slow.

(But note that these conventional meanings for colors are culturally dependent, and what works in Western cultures may not work for all users.)



## Choosing Good Colors

- Copy colors from other interfaces
  - FireBug, EclipsePalette, Digital Color Meter
- Pick colors out of a photograph with natural colors
- Pick one color and several shades of gray (safe choice)
  - Or pick two colors that seem coordinated (ask for other opinions on this)

Spring 2012

6.813/6.831 User Interface Design and Implementation

31

Given all these rules about what colors *not* to choose, what colors *should* you choose? There are no hard-and-fast rules here, but there are a few heuristics. The first heuristic is an old standby – use color schemes that seem to work well for other interfaces on the desktop or the web. There are several tools you can use to probe your web browser (Firebug for Firefox) or desktop screen (EclipsePalette for Windows, Digital Color Meter for Mac) to determine what color is being used by a particular display element.

Another effective heuristic is to find a photograph of a natural scene that looks appealing to you, and extract colors from it (using the same tools, or using the eyedropper tool in a paint program). The intuitive basis for this heuristic is that our visual systems evolved to easily perceive and appreciate the natural world.

Keep your choices simple. You can't go far wrong by choosing one weakly saturated color and a few shades of gray. As soon as you choose two colors, however, you run the risks of an aesthetic clash between them; it's good to get some other opinions on your choice, particularly if you might be somewhat colorblind yourself.



## Tools

- Use browser developer tools to look at CSS style
- DigitalColorMeter (Mac), ColorPic (Win) identifies colors from screen
- ColourLovers (crowdsourced palettes)
- NASA Color Tool

Spring 2012

6.813/6.831 User Interface Design and Implementation

32

There are also some sites out there that help you choose colors. Colour Lovers (<http://www.colourlovers.com/>) is a large collection of user-contributed color schemes, with ratings and votes. The NASA Color Tool (<http://colorusage.arc.nasa.gov/ColorTool.php>) helps select a palette of colors using HLS and view them side-by-side on sample data.

## Color in CSS

- You can declare CSS colors in a few different ways:
  - With the actual color name (ie red, blue, green)
  - With the hex code (#ffffff)
  - With the RGB value `rgb(67,250,90);`

## Summary

- Don't rely solely on color distinctions
  - Color blindness is common
- Keep your color design simple
  - Use few colors, weakly saturated

## 6.813/6.831 • USER INTERFACE DESIGN AND IMPLEMENTATION

Spring 2012 Massachusetts Institute of Technology  
Department of Electrical Engineering and Computer Science

### PS2: OUTPUT

Due at 11:59pm, Sunday, April 1st, 2012, by uploading submission to Stellar.

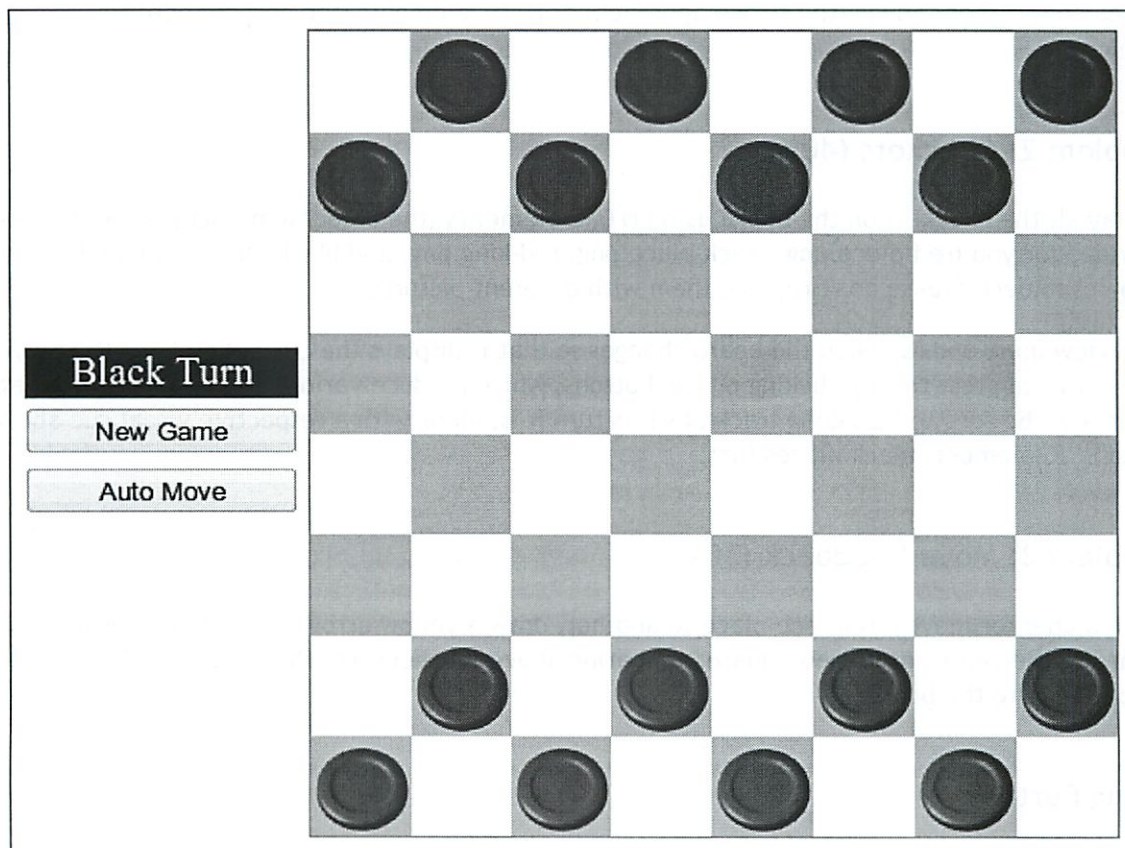
This assignment explores the following topics related to GUI output:

- the object approach;
- the stroke approach (also called vector graphics);
- the pixel approach;
- handling events sent from a model to a view.

In this problem set, you will implement a view that displays a checkers game. You'll implement using a mix of component, stroke, and pixel techniques. In this problem set, you'll only be concerned with output. In the next problem set, you'll add input handling to your views, so that the user can click and drag checkers using the mouse.

You will need to use HTML5 Canvas on this assignment, and this reference may be helpful:

- Mozilla Canvas tutorial





## Provided Resources

We provide you with a lot of existing code for this assignment. You can get it all at once here:

- [ps2.zip](#): a zip file containing a main page, stylesheet and javascript libraries for this problem set.

You can import this zip file directly into Eclipse using File/Import/Existing Projects into Workspace, or use whatever text editor you would like. In the root folder of the project are the following files:

- **index.html**: a skeleton file for your user interface
- **mainLayout.css**: a stylesheet file for index.html
- **checker.js**: a javascript file containing the Checker class
- **board.js**: a javascript file containing the Board class
- **boardEvent.js**: a javascript file containing the BoardEvent class
- **graphics**: a folder containing all the graphics files

The board model actually has pieces on it, but you won't see them until you've implemented the pieces display.

### Problem 1: Board (30%)

Fill in the skeleton of index.html so that it displays a 400x400-pixel checkerboard. The upper left square should be white. The number of squares across the board should be dynamically determined by the `BOARD_SIZE` variable in the code. This size defaults to 8x8, but you can change it to any value  $n$  by adding `?size= $n$`  to the end of the URL, e.g. `index.html?size=16`. No matter how many squares are in the checkerboard, it should always be 400x400 pixels.

You can use either canvas (the stroke approach) or HTML elements (the object approach) to solve this problem.

### Problem 2: Checkers (40%)

Display all the checkers on the board using **HTML elements** (the object approach). Four pictures are provided for you (red-piece.png, black-piece.png, red-king.png, and black-king.png, found in the graphics folder). Please don't replace them with different pictures.

Your view must update when the board changes so that it displays the current state of the board at all times. You can test this by clicking on the buttons, which produce various legal board configurations in the model. Be sure to also keep track of who's turn it is, along with a respective visual cue above the buttons. Remember, black moves first.

### Problem 3: Move Feedback (30%)

When a checker moves from one place to another, draw a yellow arrow from the center of its old square to the center of its new square, appearing above all checkers. This arrow should persist until the next change to the board.

## Going Further

If you found this assignment easy and you're inclined to go further, try the following:

- Animate checkers when they move
- Draw the checkerboard in perspective, so that it's more obvious to the user which side they're playing.

## What to Hand In

Package your completed assignment as a zip file that contains all of your files.

List your collaborators in the comment at the top of index.html. Collaborators are any people you discussed this assignment with. This is an individual assignment, so be aware of the course's [collaboration policy](#).

Here's a checklist of things you should confirm before you hand in:

1. Make a fresh folder and unpack your zip file into it
2. Make sure your collaborators are named in index.html
3. Make sure all assets used by your code are found in the fresh folder and load successfully
4. Make sure that the page renders correctly in at least two modern, standards-compliant web browsers (Firefox, Chrome, Safari)

Submit your zip file on Stellar.

# PS2 Output

3/29

Object, stroke, pixel approach on the web

I should read about Canvas

HTML element where JS can draw stuff

First in Mac OS X Dash board

Width & height

full back content

---

Get w/ JS

Access drawing context w/ `getContext`

---

Then draw

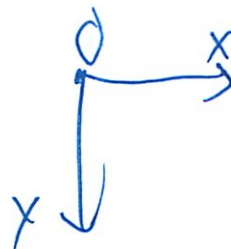
`ctx. fillStyle = "rgb(200,0,0)"`

`ctx. fillRect(10, 10, 55, 50)`

`body onload = draw()`

---

Grid coord space



⑦

fill  $\leftarrow$  middle  
stroke  $\leftarrow$  border  
(x, y, width, height)

MoveTo (x, y) places pencil somewhere else  
for Paths

or lineTo

Arc ( x, y , radius, start Angle, end Angle, anticlockwise)

Quadratic CurveTo ( )

bezier Curve

load images

or put data in as base64!

or draw on top of

transparency, line style, gradients, etc...

Oh enough of that



③

## Board

HTML or Canvas

? more comfortable w/  
So where put this code?

He uses tables...

Woot my rows + cells worked on first try!

Trouble w/ displ, inline

float left better

Weird checkerboard ~~color~~ display  
w/ diagonal of white!!

Since 1 to many!

① Done

But w/ 4? an odd #?

Still wrong straight lines

① Fixed

but not 11 for some reason??

9

Looks to be some dividing error

Done part 1

---

## Part 2 checkers

Display the checkers w/ HTML elements

Just display or also support moves?

So some sort of model provided

Where do we add code

- a 3rd party script using `getAlCheckers`  
- or modify Add script  
    ? do

place it in that board

.html()

Object oriented JS  
Color

---

Running into trouble here adding the item  
Why won't it select!!!  
it's w/ canvas?

5

Yeah that seemed to be it - 15 min on that!

1 dynamically size pieces?

Now move

being lazy and not doing proper functions

its not adding...

- ✓ So black is moving now
- but red never moves
- ? promote disappearing

Need to track whose turn it is

oh I think I need to track that  
I stupid!

Remember ~~extra~~ functions can be var

✓ Fixed whose turn  
Now display

⑥

① Done

Everything appears to be working

Pieces capture

(when promote

Something would here...

it promotes old cards

perhaps don't do anything w/ those lines  
~~it~~

Now it promotes when it leaves

- is that correct?

② Fixed - Done part 2

---

Part 3

Move feed back

Draw an arrow above all archers

(Need canvas)

---

Do we need to be able to drag them?



⑦

Don't think so, but we were supposed to use listeners

Oh well

Violated separation principles

Do we have to fix?

Prob should

Well wait for Piazza answer

---

More ~~Add~~ Feedback

Guess need canvas

Or put that stuff on listener!

Also position to pull out of doc

Got canvas to show up

Got arrow

but not showing up...

Why not?!

WTF!

Oh need stroke()

⑧

Now need to clear

① Done

Assignment done ①

Check Chrome ②

## 6.813/6.831 • USER INTERFACE DESIGN AND IMPLEMENTATION

Spring 2012 Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

### PS3: INPUT

Due at 11:59pm, Sunday, April 8, 2012, by uploading submission to Stellar.

This assignment explores the following topics related to GUI input:

- event handling
- hit detection
- dragging
- undo and redo
- enabling/disabling commands based on view state

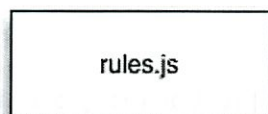
In this problem set, you will implement input handling for the checkerboard you created in PS2, so that the user can pick up checkers with the mouse and drag them around. We provide a class which implements the rules of checkers that gives you feedback about the legality of the user's moves. You will use this object to implement turn order, undo, and redo. After this problem set, you should have a basic but functional checkers game.

### Provided Resources

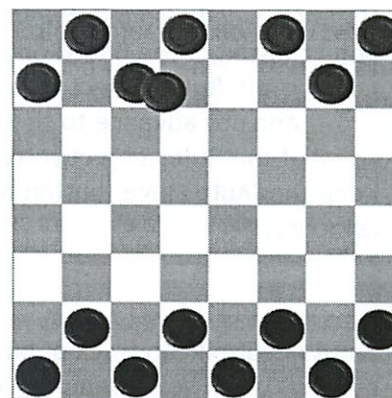
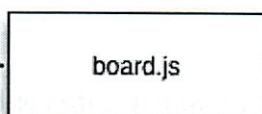
This assignment builds on top of the code you wrote for PS2.

In addition to the **Board** object provided in `board.js`, you will be using the **Rules** object found in `rules.js`. The **Rules** object wraps the **Board** object and provides an interface to the board that constrains your manipulations to those which fall into the rules of checkers.

Manipulate the board state  
according to the rules of Checkers



Manipulate the  
board state



The **Rules** class has two methods you'll need: one which attempts a pre-selected move for a player and one which attempts to randomly move for a player. Both of these methods return null if the move was invalid (or no moves were possible for that player, in the latter case). On success, they return an object containing the results of the move if successful (any jumped pieces, etc). See the `rules.js` file for detailed documentation.

### Problem 1: Dragging Checkers (40%)

Add input handling to your Checkerboard, so that the user can drag checkers around.



Pressing and dragging in a square containing a checker should pick up the checker from the board. The checker should not make any abrupt jumps — neither when the user presses the mouse button, nor when the user starts moving the mouse. The checker should move smoothly with the mouse pointer, hovering over the other checkers on the board.

- If the user releases the mouse button when the mouse pointer is over an empty square, the checker should be moved to that square using `Board.move()`. The board model should not be changed until then. If the mouse pointer is over a filled square, then the dropped checker should be put back where it was taken from. (note: we'll change this to use the `Rules` object on the next step)
- If the mouse pointer leaves the checkerboard during the drag operation, the dragged checker may either follow it or stop wherever it is on checkerboard — the behavior is up to you. But if the mouse pointer moves back into the checkerboard, the checker should resume following the mouse. If the mouse button is released when the mouse pointer is off the checkerboard, the checker should be put back where it was taken from.

Hints:

- Be aware of z-order, since you want the dragged component to hover on top. (note: HTML5-compatible browsers may not require adjusting z-order)
- You will need to do mouse capture correctly to handle cases where the mouse pointer leaves the whole browser window. You can either attach mouse listeners to the root of the view tree (document or window object), or (on HTML5-compatible browsers), simply listening to drag and drop events on the game objects may be sufficient.

## Problem 2: Turn-based Play and Following the Rules (20%)

To make the checkers game actually playable, you'll need to constrain the user's inputs those that respect turn order and the rules of the game. You've already implemented turn order tracking in PS1, so this should be an easy step.

- Modify the code from *Problem 1* so that checker movements are performed through the `Rules` object instead of the `Board` object.
  - If the `Rules` object returns a null value, you should return the piece to its original position and not advance to the next turn.
  - If the `Rules` object returns a non-null value, advance to the next turn.
- Keep the "Auto Move" button from PS2 -- that way you can play against the (randomly moving) computer.

Hints:

- You'll need to keep track of both whose turn it is (red or black) and what direction that player is moving. When you attempt to make a move, you'll need to pass the `Rules` object the turn direction and the player direction. Both of these are either +1 or -1. Turn direction represents whose turn it is. Player direction represents which color piece is trying to move. For example, if red is +1 and black is -1, and on red's turn, the user moves a black piece, then turn direction would be +1 and player direction would be -1. See the documentation in `rules.js` for details.

## Problem 3: Undo (20%)

What's a game without the ability to take a move back?

- Add an "Undo" button that, each time it is pressed, goes one step backwards in history.



- The "Undo" buttons should only be enabled when that action is possible. For example, "Undo" should be disabled upon game start.

#### Hints

- When you make a move, either randomly or specific, the **Rules** object returns a data structure with sufficient information to implement the undo and redo functionality.
- Undo is outside the rules of checkers. That means you'll have to implement it using direct operations on the **Board** object. Don't worry about **Rules** -- it takes whatever the current state of the board is, even if you've manipulated the board behind its back.

### Problem 3: Redo (20%)

Add a redo button that plays the undo stack forward.

#### Hints:

- Think about what should happen to the redo stack when the user makes a move (either manually or by pressing automove)
- The Redo button should only be enabled when that action is possible

### Going Further

If you found this assignment easy and you're inclined to go further, here are some ideas for optional improvements:

- Do hit testing for the true area of a checker, so that clicking in the corner of a square doesn't pick up the checker.

### What to Hand In

Package your completed assignment as a zip file that contains all of your files.

List your collaborators in the comment at the top of index.html. Collaborators are any people you discussed this assignment with. This is an individual assignment, so be aware of the course's [collaboration policy](#).

Here's a checklist of things you should confirm before you hand in:

1. Make a fresh folder and unpack your zip file into it
2. Make sure your collaborators are named in index.html
3. Make sure all assets used by your code are found in the fresh folder and load successfully
4. Make sure that the page renders correctly in at least two modern, standards-compliant web browsers (Firefox, Chrome, Safari)

Submit your zip file on Stellar.

## Doing PS3

3/29

Going to add input to Checkers

Drag + Drop

So can play it as a normal checkers game

Rules.js says if ~~move~~ move valid

↳ Why sep classes for pre select vs random?

↳ generates a random move I guess

Do drag + drop

↳ watch leaving browser window

Whole square clickable (isn't that harder?)

So this seems challenging

How to drag + drop

↳ Use a library?

Or do on our own? w/ event listeners

Seems like we want too much custom stuff to  
do library

(2)

Research jQuery drag + drop

Piazza 179: You really should not use a library

Use mouse down  
up  
over to dynamically do it

---

So mouse down on the square  
starts the move

Then mouse up on the checker reads  $x/y$   
Attempts to drop there (check rules, is)

---

But how to direct clicks w/ canvas??

I saw a Piazza on this

#181: Read events from canvas

Lame

It doesn't even give you canvas OAL

Can calc offset

3

Then make checker show up

(This is very manual - unclean!)

(I never realized height + weight + only 1 letter different!!!!)

Mouse move

(I hope this is fast enough!)

↳ But I think JQuery does this

---

Opps flipped row + col

Move fires properly!

---

But top + left not updating?

Why!!

---

Woot the checkers kinda follow my mouse around!

↳ x, y screwed up

⓪ Done Dragging - it was ⓪ offset, not ⊕



④

Now dropping

① Drops

But must remove other one!

② Drops!

(So manual - I'm surprised it works!)

③ CheckBoard does leave board correctly

④ Think it leaves window correctly

---

Part 2 Turn Based Play + Following the rules

---

rules.js has 2 methods

- test move

- make a random move

- ↳ Random move already done

They are all null

- ↳ oh it's player's turn

⑤

An app's use page X not client X to scroll

Rules.js still does not allow!

What are fun direction + player direction

Oh both are +

① Fixed

Now change whose move

② Done

Red still says wrong dir

③ Fixed - was wrong obj

④ Pat 2 Done

---

### Part 3 Undo

- maintain history
- from rules object
- button graced at if nothing
- all through board.js

6

I'm doing undo + redo ~~on~~ both at once kinda

↳ why won't undo button be disable!

↳ Oh true, false ADT in quotes

(This is not very clean code - but not the point of this class!)

Now need to do the action

✓ Worked

Remove arrow

✓ Done Step 3

---

## Part 4 : Redo

But also Now game must reset by

✓ Done

Now test undo

↳ esp remaining pieces

⑦

Opps new game never sets blacks move

Do we need to redraw arrow?

I'll go w/ no

Don't want to ask

(Though if properly build should be simple!)

Oh didn't update log on auto move

— That's why!

Not building checker correctly

Undo should prob toggle turn

— Also make king was incorrectly capitalized

○ Reading seems to work

○ Everything seems to work!

— Not all moves changed the log → buggy  
Note: every game!



8

Sometimes it works perfectly

- I don't know what the bug is

! Declare it done?

Bug not reproducible!

Try it 1 more time

Oh if you undo then auto move from there, then undo

! Ah I inserted at end

✓ Fixed

Gray out redo when auto move.

- kill the rest of the log

✓ Fixed

---

Test Chrome ✓ Works

Test 16 size ✓ Works

---

Asked the arrows question just in case

PS 3 Out today, due Sun  
Last P-set

GR3 due in 12 weeks

---

Hall of Fame/Shame: Google Docs updated

- lots of text in same color
  - all ~~the~~ links say Click Here
  - See more link before ~~all~~ changes
- 

Nanoquiz

Which color pairs are valid reasons for not using  
(people confused)

Red, blue not <sup>b/c</sup> colorblind

Green red - yes

blue yellow - yes, less often

(2) blue, black - since yellowing in older  
blue / red - ~~color~~ chromatic ab  
green / yellow - not " "  
~~black~~ black, dark gray - contrast

highly sat

FF0000 - red

FF00FF - purple

Not gray stuff, white, or black

---

Today Readability  
Fonts  
Whitespace

Last lecture in graphic design

Latin - alphabet centric

↳ he doesn't have a reference

3

## Readability

fixations and saccades

reading process not continuous

- illusion brain is creating
- actually jump + focus

Shape of word is very important

legibility - rec. ind characters

readability - rec. whole blocks of text

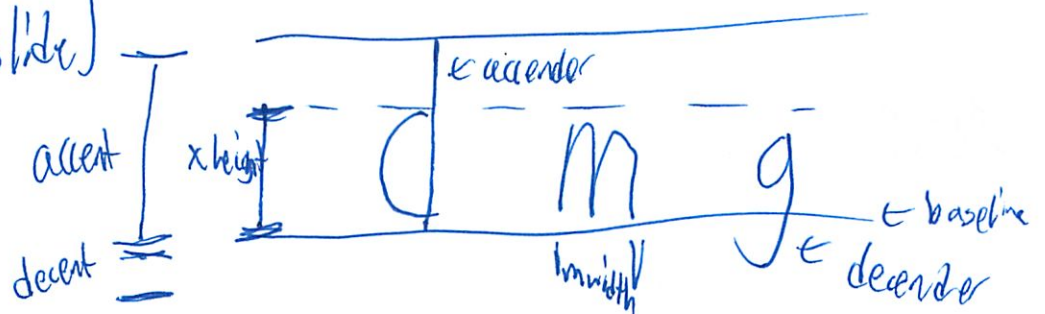
metrics: speed, comprehension

Subjective preference

---

Dim of a font

(see slide)





④

Ascenders far more imp. than descenders

font size  $\rightarrow$  diff. ascent line and descent line

point =  $\frac{1}{72}$  of an inch

12 pt =  $\frac{1}{6}$  in

Spacing

Lettering = white space b/w the letters

Space b/w lines - more space = easier to jump

Don't line height = font space

Have it little tighter

but not too far!

120% line height

~~not~~  $\rightarrow$  leave margin around page

leading = extra space b/w lines

5

## Typeface

text w/ sans easier to read

More ~~of~~ cve about edge

Sans-serif was better on low-res screen  
-falling apart

Variable vs fixed width  
better for code

↑ more contrast  
read faster

## Style

roman, bold, italic  
↑ some pre designed  
↑ pops out more than italic

## Header text

Laka "display text"

best when less text

all caps painful to read

↳ fine for headings, not body text

(6)

Some fonts even have lower case digits!

Was popular in 1800s, 1900s

Cool when embedded in text

Not for col of #s

Fonts determine it

---

## Encoding

Char sets

- ASCII
- Latin 1
- Unicode

Not all fonts implement all glyphs

Instead the error glyph 

Must know file's encoding to be able to load it

⑦

## Tricky issues

\* X astrix vs multiply  
" " quotes <sup>L only in unicode</sup>

~~ask~~

Word replaces straight quotes w/ curly quotes (smart quotes)

- are only in Latin-1

- not in ASCII

- So browser renders wrong

### Hyphens + Dashes

- is a variation / distinction

- hyphen - hyphen word

- em-dash - ranges

- em-dash

Unicode

### Spaces

- diff type of ~~space~~ spaces

- like non-breaking spaces



8

## Font selection

- don't use more than 2 or 3 typefaces
  - header + body
- Only 1 from same cat
- then use weights
  - but not more than 4-5

## CSS

font-family, etc

Extensions in CSS 3

can provide font files

↳ Google font site

## Tools

Browser CSS inspector

~~Font~~ Indentifont - 20 questions

What The Font - image lookup

## **L19: Typography**

Spring 2012

6.813/6.831 User Interface Design and Implementation

1

## UI Hall of Fame or Shame?

**Google Docs** notify@google.com  
to me ▾  
See the changes in your Google Document "nanoquiz makeups 6.813/6.831 spring 2012": [Click here](#)  
A user made changes from 3/18/12 6:59 PM to 2:30 AM (Eastern Daylight Time)  
• Form submit (6)  
  
Open the current version of your Google Document "nanoquiz makeups 6.813/6.831 spring 2012": [Click here](#)  
Powered by Google Docs  
—  
Want to stop receiving this email? [Click here](#)

Spring 2012

6.813/6.831 User Interface Design and Implementation

2

Google Spreadsheets has a nice feature that allows you to track changes made to the spreadsheet – e.g., when a form submission adds a new row to the worksheet, or when somebody edits cells in the worksheet. Here's an example of an email that Google Spreadsheets sends you to report changes made recently. (Does an email message have a user interface? Yes!)

Let's talk about what's good or bad in the usability of this email. Some points to ponder:

- simplicity
- information scent
- what does a squint test show?

## Today's Topics

- Readability
- Fonts
- Whitespace

Spring 2012

6.813/6.831 User Interface Design and Implementation

5

**Typography** is the art and science of displaying text (or “setting type” as print designers call it). The key decisions of typography concern font (the shapes of letters and other characters) and spacing (the white space around letters, words, lines, and paragraphs). Both are important to successful text display; without adequate spacing, the shape of the text is much harder for the eye to discriminate.

For typography, an outstanding book is Robert Bringhurst, *The Elements of Typographic Style*, Hartley & Marks, 2002. Also useful is “Principles of Typography for User Interface Design” by Paul Kahn and Krzysztof Lenk, *interactions*, which you can find online.



## Readability

- Reading process consists of fixations and saccades
- Readability vs. legibility
- Metrics of readability
  - Speed
  - Comprehension
  - Subjective preference

Spring 2012

6.813/6.831 User Interface Design and Implementation

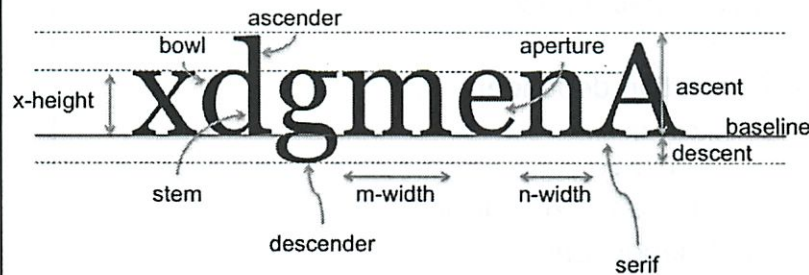
6

In keeping with our brief tours of cognitive science for each design topic, let's say a bit about what's known about reading. Note that these high-level comments are applicable to most written languages, not just English. Most of the rest of the lecture will be specific to languages that use the Latin alphabet and its corresponding fonts, however.

First, reading is not a smoothly linear process, even though it may feel that way to introspection. The eye does not move steadily along a line of text; it proceeds in fits and starts, called **fixations** (stopping and focusing on one place to recognize a word or several words at a time) and **saccades** (an abrupt jump to the next fixation point). At the end of a line, the eye must saccade back to the beginning of the next line.

Researchers studying reading and typography often make a distinction between *legibility*, which is low-level and concerns how easy it is to recognize and distinguish individual letter shapes, and *readability*, which concerns the effectiveness of the whole reading process. A single fixation can consume whole words or multiple words, so fluent readers recognize the shape of an entire word, not necessarily its individual letters. Readability can be measured by several metrics, including speed, comprehension, error rate, and subjective preference. Readability is essentially the usability of a display of text.

## Dimensions of a Font



Spring 2012

6.813/6.831 User Interface Design and Implementation

7

Now a few definitions. Characters in the Latin alphabet all sit on a common baseline. Some characters have descenders dipping below the baseline; others have ascenders rising above the typical height of a lowercase character (the **x-height**). Capital letters also ascend above the x-height. The typical height of ascenders above the baseline is called the **ascent**, and the typical height of descenders below it is called the **descent** of the font.

The **font size** is typically ascent + descent (but not always, alas, so two fonts with the same numerical size but using different typefaces may not line up in height!). Font size is denoted in points; a point is 1/72 inch, so a 12-point font occupies 1/6 of an inch vertically.

X-height, m-width, and n-width are useful font-dependent length metrics. You can find them used in CSS, for example. They allow specifying lengths in a way that will automatically adapt when the font is changed.

More font terminology can be found at <http://www.davidairey.com/images/design/letterform.gif>.

## Measurements in CSS

- Device-dependent
  - px
- Resolution-dependent
  - in, cm, mm
  - pt = 1/72 in
  - pc ("pica") = 12pt = 1/6 in
- Font-dependent
  - em = font size
  - ex = x-height

# Spacing

Vott

20/20

Four score and seven years ago,  
our forefathers brought forth upon  
this continent a new nation, conceived in  
liberty and dedicated to the proposition  
that all men are created equal.

Vott

20/24

Four score and seven years ago,  
our forefathers brought forth upon  
this continent a new nation, conceived in  
liberty and dedicated to the proposition  
that all men are created equal.

rnm

20/28

Four score and seven years ago,  
our forefathers brought forth upon  
this continent a new nation, conceived in  
liberty and dedicated to the proposition  
that all men are created equal.

kerning

Spring 2012

6.813/6.831 User Interface Design and Implementation

9

Several kinds of spacing matter in typography. At the lowest level is character spacing. The gaps between characters must be managed to prevent uneven gaps or characters appearing to run into each other. **Kerning** is the process of adjusting character spacing for particular pairs of characters; sometimes it needs to be narrowed (as V and o shown here), and sometimes widened (e.g. to keep “rn” from looking too much like “m”). A good font has kerning rules built into it, and a good GUI toolkit uses them automatically, so this is rarely something GUI programmers need to worry about, except when choosing a font. Note that the top Vott is displayed in Georgia, which at least on my system appears *not* to have any kerning for V and o. The second Vott is displayed in Times New Roman.

Spacing between words and lines matters too. Words must have adequate space around them to allow the word shape to be easily recognized, but too much space interferes with the regular rhythm of reading. Similarly, adequate line spacing is necessary to make word shapes recognizable in a vertical dimension, but too much line spacing makes it harder for the eye to track back to the start of the next line. Line spacing is also called **leading**; technically speaking, the leading is the distance between baselines of adjacent lines. Both font size and leading are important. Print designers say, for example, “12 point type on 14 points of leading” (or “12/14”) to indicate that the font size is 12 points (typically ascent + descent) with 2 points of space between the descent of one line and the ascent of the next. Using the same line height as font size (like 20/20) is almost always a mistake; characters from adjacent lines touch each other, and the paragraph is much too crowded. Note that leading also strongly affects the overall **value** of the body text (which type designers somewhat confusingly call the “color” of the text; historically print is mainly black-and-white, of course, but it’s confusing when talking about modern printing and modern computer displays). Tight spacing looks much darker than loose spacing.

In most toolkits, choosing a font size implicitly chooses a leading, but the default leading may not always be the best choice. Look at it and adjust if necessary. CSS makes this possible using the line-height property.



## Spacing Guidelines

- Use whitespace
  - Always leave margins around body text; never pack it tightly against an edge
- Use generous leading
  - Make sure body text is not overcrowded
  - e.g. CSS: line-height: 120%;
- Keep text paragraphs narrow
  - About 60-75 characters / 12 - 15 words / 30-45 em

Spring 2012

6.813/6.831 User Interface Design and Implementation

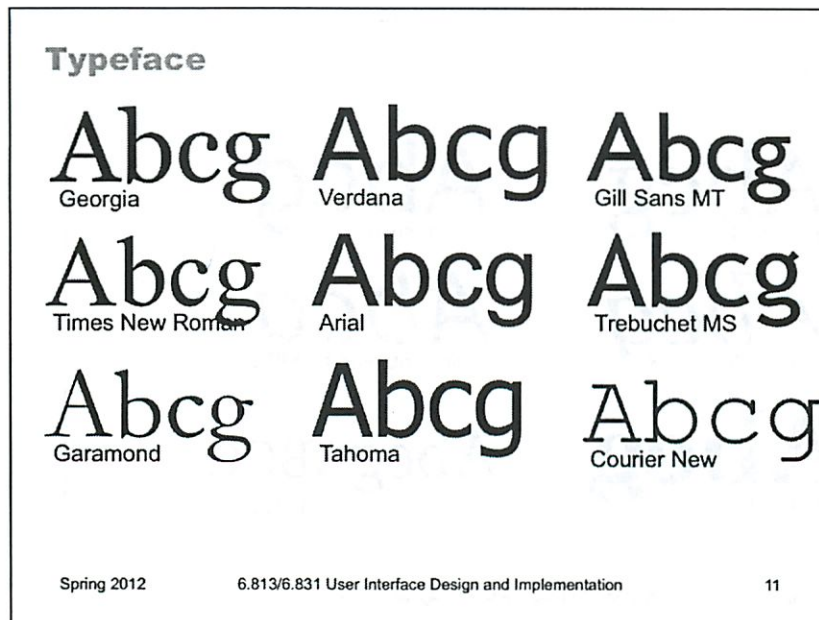
10

Here is some advice for choosing spacing for body text (text set in paragraphs). Always leave margins around body text; never pack it tightly against a rule, an edge, or a window boundary. The margin helps frame the text and also helps the reader find the ends of the lines, which is essential for the saccade back to the beginning of the next line.

Use generous leading, but not too generous. 120% of the font size is a good rule of thumb; this would correspond to the 20/24 leading shown on the previous slide.

Line length (or equivalently, paragraph width) is another important consideration. Hundreds of years of experience from print typography suggest that fairly short lines (3-4 inches) are both faster to read and preferred by users. Unfortunately the studies of onscreen reading yield mixed answers; apparently, on screen, longer lines (about twice the ideal length for print) help users read faster, but users still prefer the short lines (perhaps because their consistent feel with print). These same studies show that onscreen reading is slower than print reading, however, and recent studies have shown less and less effect of line length on speed, possibly because display and font technology is improving rapidly. So it's possible that making the lines longer merely offsets the poorer resolution and legibility of computer displays relative to print, and as the displays approach print in quality, this distinction will go away. (Bailey, "Optimal Line Length: Research Supporting How Line Length Affects Usability", December 2002, [http://webusability.com/article\\_line\\_length\\_12\\_2002.htm](http://webusability.com/article_line_length_12_2002.htm))

Translating the 3-4 inch rule into characters or m-widths for typical 10-point to 12-point type gives 60-75 characters or 30-45 em widths.



After spacing, a key decision is what **typeface** to use. Typeface refers to a family of fonts sharing the same name, like “Arial” or “Georgia.” A **font** is a choice of typeface *and* size *and* style, like roman, italic, oblique, boldface, etc.

Typefaces can be classified in many ways, and can convey strong associations that influence how the user perceives the text. One important classification you should know is between **serif** fonts, like Georgia and Times, and **sans serif** fonts, like Verdana and Arial. Historically, in print typography, serif fonts have been used for **body text** (text set in paragraphs), because they offer stronger cues to word shape that allow measurably faster reading. Sans serif fonts were generally used for **display text** (text that stands alone, like headings and labels), for which reading speed is less important and contrast from body text is useful.

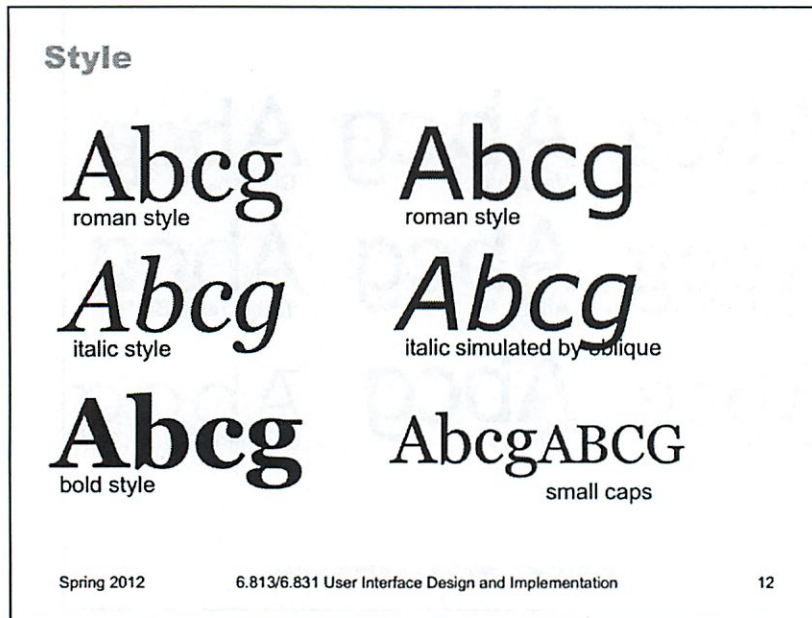
In the early days of computer typography, sans serif fonts were often preferred for all uses, because their simpler letter shapes were far more legible on low-resolution displays. As displays became higher resolution, however, serif text may once again assert itself; even now, there is evidence that serif fonts are faster to read on screen (Bernard et al, “A Comparison of Popular Online Fonts: Which is Best and When?”, Usability News, 2001, <http://www.surl.org/usabilitynews/32/font.asp>).

Another key distinction is between proportional fonts (in which each character has a different width) and monospace fonts (in which all characters have the same width, like Courier New shown here). Monospace fonts waste screen space and generally look worse than well-designed proportional fonts, so avoid them unless you have a good reason.

Your choice of font family conveys a tone. Some designers think Times and Arial look cheap because they’re so widely used; using Georgia or Garamond will give your UI a more “designed” look (i.e., you actually made an informed choice, rather than choosing a default). Another consideration is whether the font was designed for screen use. On this slide, Verdana, Georgia, Tahoma, and Trebuchet were commissioned by Microsoft primarily for onscreen use. Most of the other fonts shown here are digital updates of old fonts originally designed for print. Note some distinct features of Georgia/Verdana/etc. relative to the others – larger x-height (as a fraction of total ascent) and generous bowls and apertures, intended to make the fonts more legible at small sizes on lower-resolution displays.

All the fonts shown here are appropriate and useful for body text (though they aren’t the only possibilities, of course). Fonts for body text are designed to have evenly-distributed “color” (the value of the text in a squint test) so that they look good in bulk. You will find many other fonts installed on your computer, some of them very wacky. Some of these may be useful for occasional display text, but certainly not body text. Unfortunately today’s word processors give users many fonts but very little help selecting the right one for the right use. Instead we get a long undifferentiated list of installed fonts that hides gems like Garamond and Georgia in a sea of Comic Sans, Goudy Stout, and Old English Text MT -- some of which probably should not be used in any imaginable circumstance.





We said that a font consists of typeface, size, and style. Here are a few common styles you can use to establish contrast.

Italic and boldface create contrast in orientation and value, respectively, without substantially changing the shape of the typeface. Some typefaces lack a true italic, and instead substitute an *oblique* font which is just a slanted version of the normal roman style (sometimes even automatically-transformed from the roman font, not hand-designed). Georgia, shown on the left, has a true italic – notice that the b loses its lower serif, and the g actually changes shape. Sans serif fonts have an oblique rather than italic; look at Arial for an example.

Small caps is another useful style. Small caps are uppercase letters that are as tall as the x-height, rather than the full ascent of the font. Like italic, small caps are sometimes a hand-designed font included with the typeface family (often slightly wider and lighter than capital letters), and sometimes simply automatically generated by shrinking the font.

## All Caps vs. Mixed Uppercase/Lowercase

LEDGER

all caps

0123456789

uppercase digits

Ledger

mixed case

0123456789

lowercase digits

Spring 2012

6.813/6.831 User Interface Design and Implementation

13

While we're talking about capital letters, it's worth discussing when it's appropriate to set text in all capitals. All-caps has very little variation in word shape, because all the letters have the same top (the full ascent of the font) and the same bottom (the baseline, with almost no descenders). For this reason, it's both slow and unsatisfying to read body text set in all-caps. All-caps should be reserved only for display text (headings, labels, etc), and even then used very sparingly.

Older print typography actually had lowercase *digits*, not just letters. Notice that the lowercase digits predominantly follow the x-height, with ascenders and descenders for certain digits, just like lowercase letters. You may have seen typesetting like this in older books, published in the first half of the 20<sup>th</sup> century or earlier. Lowercase digits fell out of fashion in print in favor of more uniform uppercase digits, which may be monospaced horizontally as well, so that columns of digits line up easily; all the digits in Times New Roman have equal width, for example, even though the rest of the typeface is proportional. But lowercase digits are worth some consideration. They are more readable in body text than uppercase digits, for the same reasons as lowercase letters, and they convey a feel that is simultaneously retro and "designed." Unfortunately the character sets we use (ASCII and Unicode) make no distinction between lowercase 5 and uppercase 5 (unlike a and A), so when you choose a typeface, you either get *only* lowercase digits (like Georgia on the bottom) or *only* uppercase digits (like Times on top).



## Character Sets and Encodings

- Character sets
  - ASCII: A-Z, a-z, 0-9, punctuation, control characters
  - Latin-1: ASCII + accented Latin alphabet
  - Unicode: Latin-1 + Greek, Cyrillic, CJK, math symbols, ...
- Fonts map characters to visual appearance
- Encodings map characters to numbers
  - ASCII: A-Z map to 65-90
  - Latin-1: Å maps to 192
  - UCS-2: each character maps to 2 bytes
  - UTF-8: each character maps to 1-3 bytes

Spring 2012

6.813/6.831 User Interface Design and Implementation

14

Note the difference between **character sets** and **fonts**. The Unicode character 'A' doesn't actually say how to *draw* A on the screen; a font does that. So even though you can represent many different alphabets in a single Unicode string, the *font* you're drawing the string with doesn't necessarily know how to draw all those characters. The appearance of a particular character in a font is called a *glyph*. Many fonts only have glyphs for a small subset of Unicode. For characters that aren't supported by the font, you'll see an error glyph, which might look like a little empty square or a question mark.

Note also the difference between character sets and **encodings**. A character set is an abstract set of possible characters. ASCII had 128 characters; Latin-1 had 256 characters, and Unicode has thousands of characters. An encoding maps each character in a character set to a number (or a small sequence of numbers). Internally, Java uses a 16-bit encoding for Unicode characters, representing each character by two bytes in memory. But the most common encoding for Unicode text in files and web pages is UTF-8, which does *not* use two bytes per character. Instead, UTF-8 uses 1, 2, or 3 bytes to represent each character. Single bytes are used to represent all the 7-bit ASCII characters, so UTF-8 can be thought of as an extension to ASCII.

There are other encodings as well. ASCII maps its characters to the numbers 0-127, which are stored in bytes. Latin-1 (also called ISO 8859-1 after its ISO standard) maps its characters to 0-255 (compatibly).

In general, **you cannot correctly interpret a text file or web page without knowing its encoding**. If your code ignores encodings and assumes everything is ASCII, you will find that it mostly works as long as you only use English, because encodings generally strive for backwards compatibility with ASCII. In other words, an English text would probably look identical in ASCII, UTF-8, and Latin-1. But it may break horribly on text in other languages. Even English text has problems when the author uses punctuation that isn't available in the basic ASCII character set. For example, ASCII only had one kind of double-quote mark (a vertical one), but many word processors now use left and right double quotes that are available in Unicode and other character sets, which often turn into garbage characters when you load the text into encoding-ignorant programs.

For more about encodings, Joel Spolsky has a good article (<http://www.joelonsoftware.com/articles/Unicode.html>).

## Tricky Characters in Online Typesetting

- Asterisk vs. multiply      \*    ×
- Quotes      '...' '...' "..." "..."
  - ASCII only has the straight quotes, not the curly ones
- Hyphens & dashes      -    –    —
  - hyphens, en-dashes, em-dashes
- Spaces
  - nonbreaking spaces are different from ordinary spaces

## Font Selection

- Simplicity & contrast
  - Don't use more than 2 or 3 typefaces
    - E.g., one for body text, one for display text
  - Don't use two faces from the same font category
    - e.g. only one sans serif
  - Use size, weight, style (e.g. italic/small caps), hue to establish essential contrasts
    - But 4-5 font varieties should be enough

Spring 2012

6.813/6.831 User Interface Design and Implementation

16

In general, decisions about typography are like other decisions in graphic design: use font selection to make important contrasts, and otherwise keep your font choices simple. Don't use more than 2 or 3 typefaces (if that many). You might use a serif face for body text, and a sans serif face for display text. Many interfaces have no real need for body text at all, in which case you can easily get away with a single typeface.

Within the typefaces you chose, use variation of size and style (and color) to establish the necessary contrasts. Size, in particular, makes it easy to establish a hierarchy, such as headings and subheadings. Even so, 4-5 fonts in all should be all you need.

## Font Properties in CSS

- font-family: Georgia, "Times New Roman", serif
  - listed in order of priority
  - generic families include serif, sans-serif, monospace
- font-weight: bold
- font-style: italic
- line-height: 120%
- @font-face {  
    font-family: 'Droid Sans';  
    src: local('Droid Sans'), url('http://...')  
}
- Google web fonts



## Tools

- Use browser developer tools to examine CSS style
- Identifont (20 questions about fonts)
- WhatTheFont (image lookup)

## Summary

- Whitespace matters for text
  - Use generous margins, line spacing, short lines
- Keep font choices simple
  - Few typefaces, few sizes and styles

6.813

4/4

## L20 Accessibility

PS 3 due Sun

~~GR4~~

GR4 due in 2 weeks

### UI Hall of Fame/Shame Flipmunk

Flight Search tool

Can type or click

Feedback - color highlighting

Will find airports close to your hometown

Settley errors - can't use the past

Unclear cal below is selectable

Can type  $\pm 2$

↳ not discoverable

Can say next Mon

### Flight display

Can see length

①

Can see both time zones

See how long layover - and airport

Color matches airline branding

Why sometimes colored

See if flight has wifi

Order by price

Can expand to see worse flight  
- Simplified by default

No ads

Some custom sort i' Agony  
tool tip

Comprehensive filtering

Includes trains

(This very, very nice. I will use in future!)



3

## Nano quiz

(I remember stuff from before!)

leading - space b/w lines

~~For~~ - changes value  
~ adds more white

whitespace - helps recognize eyes

12 pt

- not vert dist b/w baseline

Also includes baseline

X-height is not less  $\frac{1}{2}$  font  
ex

em = font size

~~from~~ ascent to descent + main (x-height)

↳ so A only has ascent

Encoding problem

is in Unicode - 65k characters

could be wrong encoding

Or font no glyph for character

④

Today Moving on from graphic design

Now looking at diff kinds of design populations

~~All~~ Now looking at ~~one~~ special user populations

Should have thought at beginning

—  
kinds of impairments

Assistive tech

Accessibility guidelines

---

Common impairments

ability at all

Visual

- color perception
- acuity (blindness)
- total blindness

Hearing

- often varies w/ freq

⑤

## Motor impairments

- Spasms
- muscle weakness
- paralysis

Some w/ trouble w/ long movements  
Some w/ short "

## Cognitive impairments

### Everyone

#### Aging

- ↓ visual acuity
- hearing loss
- arthritis

#### Overexposure

- Noise induced hearing loss
- RSI

#### Situational

Driving a car

↳ have much more impaired  
since focusing on driving

6

## Universal Design

At the start try to design for everyone

Not all ways - just 1 equitable way

Sidewalk curb cut

- good for luggage
- shopping moving

↓ cost of deployment since everyone benefits  
avoids stigmatizing

big gap

---

## Tools

Output

- screen magnifier
- screen reader
- braille display
- screen flashing on sound



①

## Pointing

- eye or head tracker
- puff and ~~sip~~ sip
- mouse keys

## Typing

- on screen keyboards
  - sticky key
  - speech recognition
- 

Try using a screen reader

Have a basket of tidbits

Like search for phone #

They know how to ~~edit~~ ship

Play stuff very fast

But need help

The type ahead is very hard for a screen reader

↳ big efficiency slow down - no sol yet!

⑧

## Accessibility Guidelines

Section 508

W3C guideline

Support keyboard access

— much better in Windows than OSX

Provide equiv text for images

images → alt

Only for important images

label widgets

It's very easy to edit

↳ must keep model in heap

---

Tool: a Designer

Visualizes page for blind person

Shows how deep info is in the page

⑨

Can mark more things as headers  
Screen readers jump to it

Allow users to pick color

Toolkits have hooks