CS 160: User Interface Design

Prototyping, Widgets, Events 03/03/14
Roll Call (CS 160 Spring 2010)

http://www.youtube.com/watch?v=jQnfC8jZaU
Video Prototype

Due one week from today, before class on 3/10
Mini Assignment

Assigned Today, finish by class on Wednesday:
Mid-semester feedback
0 pts
CS160/260A Spring 2014 Midsemester Feedback

Your answers to the questions below will help us plan for the second half of the semester (and future offerings of this course).

How worthwhile is this class so far?

☐ 1 Not worthwhile at all
☐ 2
☐ 3
☐ 4
☐ 5
☐ 6
☐ 7 Very worthwhile

Feedback to instructor: What aspects of the class do you like? What would you like to see more of between now and the end of the semester?

What do you think we could cut down on? What is not working well?
Individual Programming Assignment 3

Due before Spring Break  
(March 21, 11:59pm)

Implement the clapper for taking photos 
Learn to use sound API 
Learn to interface with camera
Loaner Devices

- **Vizio VTab 1008 (2011)**
  - Android up to 3.2
  - 8" 1024x768
  - WiFi, Bluetooth, Front Cam
  - Accelerometer

- **Nexus 7 1st gen (2012)**
  - Android up to 4.4
  - 7" 1280x800
  - WiFi, BT, NFC,
  - Front Cam, GPS, IMU

- **Nexus One (2010)**
  - Android up to 2.3.6
  - 3.7" 800x480
  - 5MP back cam
  - GPS, Wifi, 3G
Loaner Devices

Fill out loaner device survey
Link on Wiki (announcements)
First come, first served

Bring $200 check made out to “UC Regents” to class. We’ll return the check when you return the device

We don’t have many devices and they are old. A new one costs $230 – see link on wiki.
Prototyping: producing early working versions of the future application system and experimenting with them. (Lichter)
Benefits of Prototyping

1. We know more than we can tell
2. Actions in the world outperform mental operations
3. The value of surprise
The Purpose of Prototyping

What questions do prototypes answer?
When and how should they be constructed?
PURPOSE

Understand Existing Experience

“Inquiring Actions”

Explore

Communicate

Experiment

Anchor Discussion

Validate

Persuade
PURPOSE

Understand Existing Experience

“Inquiring Actions”

Explore
Experiment
Validate

Communicate

Anchor Discussion
Persuade
Figure 2: Experiencing a train journey.
PURPOSE

Understand Existing Experience

“Inquiring Actions”

Explore

Experiment

Validate

Communicate

Anchor Discussion

Persuade
Prototypes for the Microsoft mouse
From Moggridge, *Designing Interactions*, Ch2
PURPOSE

Understand Existing Experience

“Inquiring Actions”

Explore  Experiment  Validate

Communicate

Anchor Discussion  Persuade
Build for Today. Create for Tomorrow.

What can you make with Drupal? Beautiful, personal blogs or mighty, multi-featured, multi-user corporate sites. Our open source publishing software is the platform you need to create your place on the web.

Why Choose Drupal?
Drupal is like Lego. Connect the pieces and build a site limited only by your imagination. Drupal’s passionate, vibrant community are always creating new pieces, or improving existing ones. Choosing Drupal means as your needs evolve, so does your site.

Get Started with Drupal

Who Else Uses Drupal?
Newspapers: The Onion, Morris Digital, Seattle Times and many, many more...

310,721 people in 24 countries speaking 14 different languages

Develop with Drupal
Drupal is extensible, powerful, scalable, and flexible.

Current activity
- Drupal API
4212 CVS a/c holders
632 commits this month

Modules and Themes
Explore Drupal modules and themes

Advertising helps build a successful ecosystem around Drupal
Paper Prototyping
Materials

Large, heavy, white paper (11 x 17)
5x8 in. index cards
Post-it notes
Tape, stick glue, correction tape
Pens & markers (colors & sizes)
Transparencies (including colored)
Scissors, X-acto knives, etc.
Interface Elements

2. Select the Actions for your rule:
- Copy it to the specific folder
- Delete it
- Forward it to people
- Highlight it with color

3. Rule Description (click underlined value to edit):

Apply this rule after the message arrives
where this from line contains Craig Duncan
highlight it with color
Constructing the Prototype

Set a deadline
Don’t think too long - build it!

Draw a window frame on large paper
Draw at a large size, but use correct aspect ratio

Put different screen regions on cards
Anything that moves, changes, appears/disappears
Use greeking to indicate text if necessary

Ready response for any user action
e.g., Have those pull-down menus already made

Use photocopier to make many versions
Wizard of Oz Testing

SIMS 213 Project: Telebears redesign
A bit slow for a computer - but it works!
Conducting a Test

Three or Four testers (preferable)

**Greeter** - Puts users at ease & gets data

**Facilitator** - only team member who speaks
  - Gives instructions & encourages thoughts, opinions

**Computer** - knows application logic & controls it
  - Always simulates the response, w/o explanation

**Observer(s)** - Take notes & recommendations

Typical session should be approximately 1 hour

Preparation, the test, debriefing
Conducting a Test (cont.)

**Greet**
Get forms filled, assure confidentiality, etc.

**Test**
Facilitator explains how test will work
Performs a simple task
Facilitator hands written tasks to the user
Must be clear & detailed
**Facilitator keeps getting “output” from participant**
“What are you thinking right now?”, “Think aloud”
**Observers record what happens**
Avoid strong reactions: frowning, laughing, impatience – biases the test

**Designers should not lead participants**
Let users figure things out themselves as much as possible
Only answer questions if user remains stuck for a long time
Conducting a Test (cont.)

**Debrief**
- Fill out post-evaluation questionnaire
- Ask questions about parts you saw problems on
- Gather impressions
- Give thanks
Preparing for a Test

Select your participants
Understand background of intended users
Use a questionnaire to get the people you need
Don’t use friends or family

Prepare scenarios that are
Typical of the product during actual use
Make prototype support these (small, yet broad)

Practice running the computer to avoid “bugs”
You need every menu and dialog for the tasks
All widgets the user might press
Remember “help” and “cancel” buttons

WOZ is different from pre-built/canned functionality
Wizard of Oz Tips

**Rehearse your actions**
Make a flowchart which is hidden from the user
Make list of legal words for a speech interface

**Stay “in role”**
You are a computer, and have no common sense, or ability to understand spoken English.

Facilitator can remind user of the rules/think-aloud approach if the user gets stuck
Record Critical Incidents

Critical incidents are any unusual/interesting events.

Most of them are usability problems.

They may also be moments when the user

  Got stuck

  Suddenly understood something

  Said “that’s cool” etc.
Using the Results

**Update task analysis and rethink design**
Rate severity & ease of fixing problems
Fix both severe problems & make the easy fixes

**Will thinking aloud give the right answers?**
Not always
If you ask a question, people will always give an answer, even if it is has nothing to do with the facts
Try to avoid leading questions
Prototyping in Software
Fidelity in Prototyping

**Fidelity refers to the level of detail**

**High fidelity**
Prototypes look like the final product

**Low fidelity**
Artists renditions with many details missing
Paper Prototypes are low-fidelity.
What about software?
Low-fidelity prototyping in software

“Informal” design tools:
Goal is to be as rapid and flexible as physical tools
Add benefits of digital media: Undo, copy+paste, resizing, etc.
May be a good idea to design in software, then print out elements and screens for paper testing.

Examples:
DENIM (UC Berkeley)
Balsamiq Mockups
High-fidelity visual mockups

**Interface looks like the final application**
May or may not be interactive
Often, interactivity is limited to static changes

**Example Tools:**
Keynote + Powerpoint
Adobe Flash Catalyst
Hi-Fi Disadvantages

Distort perceptions of the tester
Formal representation indicates “finished” nature
People comment on color, fonts, and alignment

Discourages major changes
Testers don’t want to change a “finished” design
Sunk-cost reasoning: Designers don’t want to lose effort put into creating hi-fi design
“Slideware” Prototypes

Uses slideware to create screen designs

Libraries of widgets for many platforms (e.g., iPhone, Android, Web, Facebook apps) exist

Each slide shows one key screen
To tell others about your design (in a presentation), visually indicate the action a user is taking.

For touch screens: include hands!

Linear: you (designer) are in control of the narrative.
Telling vs. Testing for Touch Screens

To **test** your design, link different screens together and let users explore.

Non-linear: user is in control of the narrative.
High-fidelity, fully-interactive prototypes

Look and behave like the final application.
Take a lot of effort to build – too little payoff?

Example tools:
- HTML+CSS+Javascript
- Apple DashCode
- jqTouch library
Video Prototyping
Video Prototypes

Narrative: You control the story!
Use existing software & images of real settings
Narration optional (but required for your assignment!)
   Explain events while others move images/illustrate interaction

With good storyboards, should be able to create video prototype in a few hours
Creating a Video Prototype

1) Review field data about users & work practices
2) Review ideas from video brainstorm
3) Create use scenario in words
4) Develop storyboard of each action/event with annotations explaining the scene. Put each element on a card.
5) Shoot a video clip for each storyboard card

   Hold last frame of a section/shot for 1s
Roll Call (CS 160 Spring 2010)

http://www.youtube.com/watch?v=jQnfCj8jZaU
Tips & Tricks

Add structure to better explain context

Begin with a title
Follow with an “establishing shot”
Switch between showing UI and showing live shots
At the end, connect back to the original motivation

Editing: Keep it simple!

Live video is most convincing to show context, but still photos + narration can work in a pinch
Don’t obsess about transitions and composition – just tell your story.
Tips & Tricks: Showing UI Interaction

Three options for working with Balsamiq (mix and match for your project):

Link screens in Balsamiq and record your screen
Advantage: No other editing required
Disadvantage: Only discrete “clicks” are supported

Export images from Balsamiq and add interaction in PPT/Keynote/iMovie/…
Advantage: Can show continuous interaction
Disadvantage: More work

Print out Balsamiq designs and shoot live video
Advantage: Can show continuous interaction; faster than digital editing
Advantages of Low-Fi Prototyping

**Takes only a few hours**
No expensive equipment needed

**Can test multiple alternatives**
Fast iterations
Number of iterations is tied to final quality

**Can change the design as you test**
If users are trying to use the interface in a way you didn’t design it – go with what they think! Adapt!

**Especially useful for hard to implement features**
Speech and handwriting recognition
<table>
<thead>
<tr>
<th>Drawbacks of Lo-Fi Prototyping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Evolving the prototype requires redrawing</strong></td>
</tr>
<tr>
<td>Can be slow (but reprogramming usually slower)</td>
</tr>
<tr>
<td>Force manual translation to electronic format</td>
</tr>
<tr>
<td>May not allow real-time end-user interaction</td>
</tr>
</tbody>
</table>
Interactive Application Programming
In the beginning...
The Xerox Alto (1973)
Event-Driven UIs

Old model (e.g., UNIX shell, DOS)
Interaction controlled by system, user queried for input when needed by system

Event-Driven Interfaces (e.g., GUIs)
Interaction controlled by user
System waits for user actions and then reacts
More complicated programming and architecture
Do some work…
Prompt user for input
Wait for user input
Process user input…
Do some more work…
Exit
// Java Example:
Console console = System.console();
String name = console.readLine("Your name:");
System.out.println("You have entered: " + name);
String pass= console.readLine("Password:");
System.out.println("...");
Minimal “interactive” program

Do until a quit command: {
    wait for user input
    process it...
    (optionally) update display
}

Minimal “interactive” program

Do until a quit command: {
    wait for user input
    switch (input-cmd) {
        case insert: do-insert(…)
        case delete: do-delete(…)
        case backspace: …
        (optionally) update display
    }
}
Minimal “interactive” program

Can’t use this (global) approach for window systems, because the result of a user command depends on the active window (and the active component within that window).

Too many possible combinations of input x target window, and window structure is dynamic.
GUI Toolkits

Most user interfaces today are written using toolkits (e.g., QT, Cocoa, Java Swing, GTK, Android SDK,...)

Toolkits come with libraries of interactive elements (widgets) and layouts

**Toolkits also define an architecture:**

A standard way to handle input and output

Usually wrap main() – application programmer writes pieces of code that plug into the architecture

Specifies how to write new widgets for the library
Widgets
Android Widgets

List item number one

Second list item

This is the third item

2 LINE LIST

2-Line List

Austin mixtape cosy sweater butcher. Fixie ad

Second list item

Assumenda commodo laborum accusam

3 LINE LIST

Three line list title

Put a bird on it qui fanny pack, portland irony nu

Donec hendrerit elit nec ligula dapibus

Second row in list

Vinyl laboris lo-fi ethical, adipiscing assumenda

Curabitur gravida quam id orci sodales
iOS 7 Widgets

- Title Label
- Label One
- Label Two
- Label Three
- sketchgems.com
- Message
- Copy
- Select All
- Define
- 3
- 24
Java Swing Widgets
Windows 8 Widgets

- **Button**
  - Submit Query
  - OK

- **Checkbox**
  - Checked

- **Combo Box**
  - California

- **Date Picker**
  - September
  - 13
  - 2011

- **Hyperlink**
  - http://www.buildwindows.com

- **ListBox**
  - Apple
  - Banana
  - Grape
  - Orange
  - Watermelon

- **Progress Bar**

- **Progress Ring**

- **RadioButton**
  - Option 1
  - Option 2

- **Ratings**
  - ★★★★☆

- **Slider**
  - 71

- **Time Picker**
  - 3
  - 15
  - PM

- **Toggle Switch**
  - On
  - Off
Widgets

**Encapsulation and organization of interactive controls**
Class hierarchy encapsulating widgets
Top-level “Component” class
Implements basic bounds management, and event processing

**Drawn using underlying 2D graphics library**

**Input event processing and handling**
Typically mouse, keyboard, touch events

**Bounds management (damage/redraw)**
Only redraw areas in need of updating
User Interface Components

Each component is an object with:

1) Bounding box
2) Paint method for drawing itself
   Drawn in the component’s coordinate system
3) Callbacks to process input events
   Mouse clicks, typed keys

Android:
protected void
onDraw(Canvas canvas) {
    canvas.setStyle(…);
    canvas.drawRect(…);
    canvas.drawText(…);
}

Java Swing:
public void
paint(Graphics g) {…}

Cocoa:
(void)drawRect:
(NSRect)rect
2D Graphics Model

Widget canvas and coordinate system
Origin often at top-left, increasing down and to the right
Units depend on output medium (e.g., pixels for screen)
Rendering methods
Draw, fill shapes
Draw text strings
Draw images
Sizing

Components need to work at multiple sizes – bounding box is often controlled by a Layout Manager, not by the component itself.

Android:
View.onSizeChanged(int w, int h, int oldw, int oldh)
Working with Widgets

Make the common case fast and the uncommon case possible.

Common case: assemble standard widgets into a layout
Uncommon case: write your own widget.

Custom Components in AndroidSDK:
• Extend View class
• Paint method: Override onDraw()
• Bounding box: Override onSizeChanged()
• Callbacks: Override onTouchEvent(), onKeyDown, ...

Composing a User Interface

How might we instruct the computer to generate this layout?
Absolute layout is inflexible and doesn’t scale or resize well. (But: great for prototyping because it’s fast!)
Containment Hierarchy

- Window
  - Panel
    - Label
    - TextArea
    - Panel
      - Button
      - Button
Containment Hierarchy

Principle: Each container is responsible for allocating space and positioning its contents.
Common Hierarchical Layouts

1D Horizontal or Vertical List
2D Grid
Constraint-based Layout (Struts+Springs)
In Android, Layouts are subclasses of ViewGroup.
Android Layouts

```
<LinearLayout orientation="horizontal">
    <TextView text="red" background="..."/>
    <TextView text="green" background="..."/>
    <TextView text="blue" background="..."/>
    <TextView text="yellow" background="..."/>
</LinearLayout>
```
Android Layouts

```
<LinearLayout orientation="vertical">
    <TextView text="row one" />
    <TextView text="row two" />
    <TextView text="row three" />
    <TextView text="row four" />
</LinearLayout>
```
In Android

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout orientation="vertical">
    <TextView text="Enter Text:"></TextView>
    <EditText text="lorem ipsum..."></EditText>
    <LinearLayout orientation="horizontal">
        <Button text="Ok"></Button>
        <Button text="Cancel"></Button>
    </LinearLayout>
</LinearLayout>
```
Specifying Layout

Declarative
e.g., HTML, XAML, MXML,…

Procedural
e.g., Java Swing

GUI Builders exist for either approach (but generating procedural code is brittle)

Is your UI layout determined statically or dynamically at runtime? If at runtime, may need procedural approach.

```xml
<StackPanel>
  <Label>Enter Text:</Label>
  <TextBox TextWrapping="Wrap">…</TextBox>
  <StackPanel Orientation="Horizontal"
               HorizontalAlignment="Right">
    <Button>Ok</Button>
    <Button>Cancel</Button>
  </StackPanel>
</StackPanel>
```
Specifying Layout

Declarative
e.g., HTML, XAML, MXML,…

Procedural
e.g., Java Swing

GUI Builders exist for either approach (but generating procedural code is brittle)

Is your UI layout determined statically or dynamically at runtime? If at runtime, may need procedural approach.

```java
public void init() {
    Container c = getContentPane();
    c.setLayout(new BorderLayout());
    c.add(new JButton("One"),
         BorderLayout.NORTH);
    c.add(new JButton("Two"),
         BorderLayout.WEST);
    c.add(new JButton("Three"),
         BorderLayout.CENTER);
}
```
User input is modeled as “events” that must be handled by the system and applications.

Examples?

- Mouse input (and touch, pen, etc.)
  - Mouse entered, exited, moved, clicked, dragged
  - Inferred events: double-clicks, gestures
- Keyboard (key down, key up)
- Sensor inputs
- Window movement, resizing
Anatomy of an Event

Encapsulates info needed for handlers to react to input

Event Type (mouse moved, key down, etc)
Event Source (the input component)
Timestamp (when did event occur)
Modifiers (Ctrl, Shift, Alt, etc)
Event Content
Mouse: x,y coordinates, button pressed, # clicks
Keyboard: which key was pressed
Callbacks

```
onMouseOver(Event e){...}
onMouseDown(Event e){...}
onMouseClick(Event e){...}
onMouseMoved(Event e){...}
```
Event Dispatch

Apple, Cocoa Event-Handling Guide