11:00-2:00 sleep

2:00-6:00 review slides (1 every 30 minutes)

6:00-8:00 review readings, labs, javascript

8:00-8:15 all review

8:15-8:30 get ready

8:30 take bus

8:30 get coffee

9:00-10:30 pui

10:30-12:00 macro homework

12:00-1:30 cyert

1:30-3:00 macro

3:00-4:30 macro homework

Lecture 1: Introduction

1. Course objective
   * How to implement interactive designs that are useful, usable, and desirable
   * Rapid prototyping
   * Idea expression in computational form
   * Basic terminology and approaches used by development
   * Basics of human perception and cognition as it relates to UX design
   * Design and conduct informal user tests of prototypes to find flaws with interface
   * Future of HCI and UX design
2. Interactive computing today: desktop, web, and smartphone
3. Three phases of computing
   * Computation
   * Communication
   * Ubiquitous computing, pervasive computing, internet of things
4. Some new UX design challenges
   * Autonomous vehicles
   * Artificial intelligence: how to create best combination of people and computers
   * Conversational UIs: how to interact with voice only, what happens when intelligent agents are everywhere
   * Information visualization: how to understand millions or billions of data points

Lecture 2: What is Design?

1. Design principles
   * Flow from left to right
   * Clear starting point: or else eyes cannot distinguish where to go first, and can’t differentiate what’s important from what’s not
   * Should fit into a grid layout
2. Good design is important
   * ~50% of system implementation
   * Bad user interfaces will lead to money (reworking), reputation (brand loyalty, trust), time (wasted effort and wasted energy), security, and lives cost
     + Congressional file server: security breach when computer files accessed improperly
     + Therac-25: computer-controlled radiation therapy machine that led to fatal radiation-over dose incident because of programming errors
   * Some characteristics: emotional connection, efficient, learnable, fun, strong and clean value proposition
   * Common themes: solves a problem within a set of constraint, useful, usable, desirable
   * Note: good design doesn’t always need to be “intuitive” (ex: bike)
3. Design: human power of conceiving, planning, and making all the products that serve human beings in the accomplishment of their individual and collective purposes
4. Myths about good UX design
   * Good design is just cool graphics: graphics is part of communication, but we also need to consider what people are trying to do and how to improve on those criteria
   * Good design is just common sense: still a lot of bad design
   * The interface can be fixed at the end: changes incur costs; people’s needs should be addressed from the start with good design
   * Marketing takes care of understanding customer needs: does not address behavior, gap between what people say vs what they do and what they actually need
     + Ford – faster horse
     + Apple – it’s not the consumers’ job to know what they want
5. Users don’t know what they want
   * People are good at identifying problems they face, but not necessarily solutions
   * Not familiar with what’s possible with technology with design constraints, good design, security and privacy
6. Major considerations of UX design (3 tips of the design triangle)
   * Technology
     + Constantly changing
     + Building and testing software is hard
   * Humans
     + Considerations (accessibility, education)
     + Need wide range of knowledge and skills to account (color, layout,
   * Tasks
   * Encompassing all: organization and social
     + Cybersecurity and legal constraints
   * Beyond the diagram: market force
     + Standards, internationalization, performance needs, multiple platforms, compatibility, time to develop and test
   * Complexity: good design might not fit in ecosystem (ex: Google Glass – hard to predict emergent patterns)
7. Participants of user interface development
   * Graphic designers, UX designers, technical writers, marketers, test engineers, usability engineers, software engineers, product managers, users
8. Application development processes
   * Waterfall: requirement specification, design, coding, integrating and testing, operation and maintenance
     + Cannot move on to the next stage before completing the previous one, can’t go back
     + Cons
       - Requires getting design right on the first try
       - Can’t retract on process: stuck with unrealistic/infeasible designs, can’t adapt
       - Slow to learn what should be built: lack of user feedback until after building software
       - Example: Heathcare.gov, based on bureaucratic waterfall process with Gantt chart
   * Iterative design: short cycles of design, prototype, evaluate
     + Quickly validate assumptions, fail fast, **progressive refinement** – the more cycles, the better
     + Links well with agile methods: short sprints with a lot of feedback
9. Design
   * Driven by core needs and not exactly implementation method, as they are many ways to achieve the same goals (android phone and mobile apps)
   * Simplified representation of desired artifact (ex: text description, screen sketches or storyboards, flow diagrams, prototypes)
     + Site maps
     + Storyboards
     + Schematics
     + Mockups
10. Usability goals: to ensure progress in UX
    * Really depends on what you are trying to accomplish
    * Learnable, memorable, flexible, efficient, pleasing, robust, out of box experience (oh that’s so cool)
    * Halo Game Testing
      + Log pattern data, generate map
      + Players ran out of rounds because they were firing too soon from far away, used target indications
11. Mindset for design
    * The designer is not the user
    * Important to be empathetic
      + Understanding another person’s point of view
      + Appreciating these problems
      + Sympathizing with pain points
      + Phil Agre “How to Help Someone Use a Computer”
12. Understanding users through task analysis: set of questions to understand users and context, helps realize what you know and don’t know, quick focus to validate assumptions
    * Initial task analysis, then validate by triangulating with multiple methods (natural setting, interview, etc.)
    * Sample
      + Who is going to use the system?
      + What tasks do they perform?
      + How are the tasks learned?
      + How often do they perform tasks: frequency helps remember more detail
      + What other tools do they have? (subway map, person behind counter)
    * Simple things should be simple, complex things should be possible
    * When choosing tasks
      + Choose real user task
      + Mixture of simple and complex tasks
    * When testing,
      + Write up a description of tasks that tells story and makes sense to design team (index card) > focus on what, not how
13. Design from data through users input (if busy, buy their time with tshirts or find substitutes) or testing early prototypes
14. Google designer tested 41 shades of blue to find 2

Lecture 3: The Whys and Hows of Prototyping

1. Prototyping: building simplified versions of the system
   * Get feedback faster
     + Validate assumptions faster
     + Fix problems before code is written
     + Saves time + money by finding bugs early
   * Experiment with alternatives
   * Design centered on user by testing and observing
2. Iterative design: more cycles, the better
3. Fidelity level of detail
   * High fidelity: prototype looks like final product
     + Prototype > evaluate > iterate
     + Why not use
       - Wrong types of feedback > focus on unimportant details versus foundation
       - Avoid in early stages
       - Tools require precision and waste time on small details
       - Less creativity: narrow in one design instead of broad
   * Low fidelity: rendition with many (unimportant) details missing
     + Simulate prototype in early stages
     + Sketches > evaluate > iterate
     + Sketches act as prototypes
     + Why use
       - Little skills needed: non-programmers participate, same page, team building
       - Little time
       - No expensive equipment
       - Can test multiple alternatives
       - Fake interactions
     + Problems
       - Hard to implement some functionality (animation, drag, etc.)
       - End-users can’t use by themselves
       - Can’t show to clients > use medium fidelity
   * When paper prototyping
     + Materials (paper, index cards, posits, tape, pen and markers, transparencies, scissors)
     + Set deadline
     + Draw window frame on large paper
     + Focus on interactivity
     + Ready response for any user action
     + Photocopiers
   * Wizard of Oz: faking interactions
     + From the man behind the curtains
     + Tradition in computer industry because hard to implement certain features (ex: speech and handwriting, augmented reality)
     + Vs. Lo-fi
       - Lo-fi: act of creating rough and ready prototypes
       - Wizard of Oz: process of faking interactions to do early-stage testing with users
       - Complementary
4. IDEO: skilled in process of designing stuff, nothing in particular
5. Advanced prototyping concepts
   * Bodystorming: physically go around and pretend what it would be like (ex: pda palm)
   * Three concepts for faking it at a lean startup
     + Value hypothesis: test whether a product or service really delivers value
     + Minimal Viable Product: version of product that works with minimum effort and dev
     + Build-Measure-Learn
6. User Test
   * Steps
     + Select people representative of target users and use incentives
     + Prepare tasks typical of the product during actual use
     + Practice
   * Roles in user testing
     + Greeter: put users at ease
     + Facilitator: gives instructions, encourage thinking out loud
     + Computer: acts as WoZ > stimulates feedback and response
     + Observer: take notes and recommendations
   * Additional
     + 1 hour, record and take notes
     + instructions
       - purpose of evaluation: product, not user
       - can quit anytime
       - demonstrate equipment and how to use
       - explain how to think aloud
       - can’t help much, explain just the basic concepts
     + ethical considerations
       - distressing, responsible to alleviate
     + debriefing at the end + questions about process
   * Evaluating results
     + Priorities observations: the most important, most frequent
     + Create a written report on findings
     + Make changes and iterate on design

Lecture 4: Usability Engineering

1. Kinds of errors
   * Slips: right goal, wrong action
     + Typos, hitting the wrong menu item
     + Automatic behaviors
     + Types
       - Capture error: frequently done activity takes over
       - Description error: flipping wrong switch
       - Data-driven errors: numbers on mind
       - Associative activation: office phone rings, but say come in
       - Loss of activation: go into room and forget why
       - Mode error: user performs an action appropriate to one situation in another situation > same input but different outputs depending on mode (ex: enter and caps)
         * Airbus A320 crash in 1992 because entered 33 instead of 3.3
         * Pros

User has to keep track of mode changes

Mode escape is not always clear

* + Mistakes: conscious deliberation, wrong goal

1. Designing for slips
   * Confirmations don’t help much > habituation
   * Two strategies
     + Prevent error
       - Selection rather than fill in
       - Forcing functions: physical constraints (ex: car key)
     + Improve detection and responses
       - Reduce severity of errors: cancel and undo
       - Make errors more obvious
2. Heuristic Evaluation
   * Developed by Jakob Nielsen
   * Cheap and fast method for finding problems
   * Basic idea
     + Compare each screen against a link of heuristics
     + See where interface does not comply
   * Samples
     + Visibility of System Status: keep users informed what is going on (ex: response time)
     + Match between system and real world: speak the user’s language, follow conventions
     + User control and freedom: make it easy to fix mistakes (undo, redo)
     + Consistency and standards: consistent with self and platform standards
     + Error prevention
     + Recognition over recall: make objects, actions, options and directions visible or easily retrievable (menu, retrieval cues)
     + Flexibility and efficiency of use: accelerators for experts, users to tailor frequent actions
     + Aesthetic and minimalist design: aligned and grouped elements, no irrelevant info, use grid
     + Help users recognize, diagnose, and recover from errors: error message in plain language, suggest a solution
     + Help and documentation: easy to search, focused on users’ task, not too long
   * When to use HEval?
     + All phases, but especially before a user study
   * Conducting a HEval
     + 3-5 evaluators examine UI
       - single evaluator doesn’t find every problem
       - good evluators find both easy and hard ones
     + Independently check against heuristics
     + Aggregate findings afterward
     + Can perform on UI or sketches
     + At least two passes for each evaluator: get feel for flow and scope of system, focus on specific elements
     + Phases
       - Pre-evaluation training: give evaluators needed domain knowledge and information on scenario
       - Evaluation: individuals evaluate and then aggregate results
       - Severity rating: determine how sever each problem is > individually then as a group (frequency, impact, persistence) > calcaulted after all HEvals > 0-4
       - Debriefing: discuss outcome with design team
         * Discuss general characteristics
         * Suggest potential improvement
         * [h1-4 consistency][severity 3][fix0]
   * HEval vs. user testing
     + Much faster (1-2 hours)
     + Doesn’t require interpreting user action
     + User testing is more accurate > HE may find false positives
     + Good to alternate between HE and user testing

Lecture 5: Wireframes and Design Specs

1. Design specs: good documentation of design
   * Purpose
     + Designs user experience
     + Details user interface
     + Offers design rationale
     + Additional
       - Gives overview of the problem
       - Describes mental model and a bit of the workflow
       - Describes the challenges and design considerations
       - Show how it’s done today and pain points
       - Presents a high level site map of proposed design
       - Presents key features, gives design rationale, and highlights the main ideas
       - Shows details of individual screens (med-fi) and gives expected use, highlights useful features (greys out unimportant)
       - Shows small sequence of events for important interaction
   * Usually at mid or end stages
   * Why Design specs
     + Forces thinking about important issues (consider error cases and edge paths)
     + Assures quality in visuals and behaviors of app
     + Helps get design team on the same page
     + Reference document for overview
     + Provides detailed documentation for implementation
   * Not
     + Technical specification: internal implementation
     + Business plan: former statement of goals and other background information
     + Design spec is a part of the three that makes up a good product
   * Can include
     + Change log: what was edited in each version
     + Legend: what symbols and codes mean
     + Overview
     + Key interactions
     + Main workflows
     + Key screens
     + Edge cases (errors)
   * Tailor
     + Task flows
     + Behaviors
     + Theories or unifying principles
     + Client understanding
     + Error path
     + Gaps (slow network, different screen size)
   * Good design specs
     + Easily approachable
     + Offers guidance for developers
     + Type, color, layout, behavior, interaction, patterns > doesn’t have to be new
       - Compiles with standards (ex: iOS design guidelines)
       - Specifies and respects any toolkit

Lecture 6: Advanced prototyping and evaluation

1. Prototyping beyond desktop-orientated: focus on augmented reality/IoT
2. Prototyping wearable technology
   * Helios: NASA wearable that helps NASA engineers (on ground) and astronauts (in space) to track and share information about tasks
     + Paper prototype on wrist
     + Augmented reality mockup using transparencies and a hat > hard to see annotation and what’s not
     + Mockup with space helmet + iphone duct taped to the wrist
   * Rats in Space
     + AR mockups with transparencies
     + Bike helmet with phone
     + Helmet with tablet reflection
   * Boeing
   * Bosch 2014: Internet of Things for Homes: make it easy to manage and connect IoT devices with each other in home
     + Created fake house, foam core and cardboard
   * Google signifiers.io
     + Guidelines on how to have physical devices offer privacy feedback
     + Use light to indicate that visual sensor is recording, but only when absolutely necessary, shutters are noticeable from device
     + Clay and physical feedback mechanisms
   * Other IoT devices
     + Motion: mirror that becomes more frosted, lights that follow user, fan spin faster when motion
     + Data transfer: colored lights to indicate status
     + Audio: equalizer that goes up and down based on volume, lights in shape of ear that would dim and brighten
3. Diary studies and experience sampling
   * Diary studies: understanding behavior over a long period of time
     + How do people spend their time, with whom do they socialize, what do they eat, how often do they feel bored, what info do they need when mobile
     + Study of mobile information needs presented in class
     + Types
       - Interval based: recorded at regular times
       - Signal-based: recorded when prompted
       - Event-based: record when events happen (rare: enter password)
       - Retrospective: at the end of each day, how well your plan matched your planned schedule
     + Pros: own environment, no Hawthorne effect, insights not from labs
     + Cons: low compliance, based on recall (fuzzy memory), diary might change behavior
4. Quantify vs. quality: the more product, the more learn about mistakes
5. Parallel design vs. serial approach
   * Parallel design: multiple designs at once, get feedback, finalize design > explore options, learn from mistakes, is better
     + Alternatives facilitate user feedback
   * Serial design: cycle of prototype and feedback
     + Fixation: stuck with the same idea
     + Serial ads are more similar
   * Conclusion: more is better
     + Better comparison
     + More individual exploration
     + More feature sharing
     + Increase in group rapport
     + More conversational turns
6. A/B Testing: comparing versions of a web page to see which ones does better
   * Pros
     + Easy to compare options
   * Cons
     + Premature optimization
     + No insights to why > but know it’s real user behavior
     + Misleading if bad comparisons
     + Need a lot of people
7. Big data now allows collection of large amounts of data with web-based apps
   * Good for UX evaluations
   * Intuition of design can often be wrong
   * Small changes in UI can have massive effects on use
   * Constant feedback in web-app world allows for faster iteration and evolution
8. Cultural probes: small packages of artifact (camera) along with task to elicit emotions in people.

Lecture 7: User Interface Design Patterns

1. Don’t need to reinvent design patterns; use well-known good design solutions
   * Ex: navigation bar: top of page, shows major subsites, logo at top-left
   * Design patterns communicate design problems and good solutions
     + Explains what the problem is
     + Why the solution is done the way it is
     + How to reuse the solution?
     + Connects to other patterns
2. Patterns
   * Good solutions to common design problems
   * Not too general, but not too specific
   * Don’t copy, but take inspiration
   * Useful because
     + Designers know it’s a similar concept
     + People are familiar with common conventions
     + Patterns are likely to be tested designs (ex: large companies spent a long time)
     + Can raise level of abstraction for communication > gives common vocabulary for team
   * Can also
     + Give common vocabulary (abstraction)
     + Help with finding usability problems
     + Offer inspiration for design ideas too
   * Some patterns
     + Footer (don’t want to overstuff nav bar), search bar, my account, shopping cart, clickable logos, content modules, multiple forms of navigation, obvious links (although these days it is more correctly guessable with flat UI), auxiliary information
3. Patterns vs. Guidelines
   * Patterns tend to be nouns (ex: nav bar, shopping cart, obvious links)
   * Patterns are more specific and generative : capable of production
     + Guidelines offer high level suggestions, but no concrete design solutions for problems
4. Pattern groups in the design of sites (Jason Hong)
   * Grouped by letter than number (ex: site genre, home page, basic ecommerce)
   * Pattern name and number
   * Example
   * Background
   * Problem Statement
   * Forces
   * Solution
   * Solution Diagram
   * Related patterns
   * Example
     + Problem: customers need to complete highly specific tasks on web page, but crowding of tangential links can prevent them (ex: Amazon checkout page)
     + Solution: minimize number of steps, provide progress bar, remove unnecessary links and content, use floating windows
5. Pros
   * More efficiency because new users are also familiar with patterns
   * Don’t have to reinvent, inspiration for ideas
   * Save money and time
   * consistent look and feel
   * Common vocabulary
   * Tested designs
6. Cons
   * Something that works is not necessarily good > sometimes conventions are not good > applying old patterns to emerging techs
   * Stifles creativity/copying everyone else > Jason says, most formatting is stable, so spend more time as UX designer on creative parts rather than the well solved parts
   * Takes time to find appropriate design pattern
7. Current design: flat and image-heavy design
   * Half.com: get books and dvds, music for cheap
8. Good artists copy, great artists steal – Steve Jobs or Picasso?

Lecture 8: Web

1. Format
   * HTML: content of the web
     + Basic behaviors (links, buttons)
   * CSS: presentation (layout, colors, size)
   * JavaScript: programming for sophisticated behaviors beyond the basic ones in HTML
2. Document Object Model (DOM): how programs can access and modify HTML
   * HTML DOM views an HTML document as a tree structure
   * Each node in tree is an html attribute
     + Each html tag gets “parsed” (read and processed) into an element into the DOM tree
   * Nodes can have attributes
     + HTML: attributes
     + DOM: properties (id, class, parentNode, innerHTML)
   * Not static, live and up to date
     + Modifying the DOM means that the web page that is displayed is also modified
     + Heart of making dynamic and interactive web pages
3. JavaSciript
   * Some things are functions, some are variables > hard o know, just have to try
   * JavaScript often fails silently, can make debugging difficult
   * Need to define variables
   * Case sensitivity
4. Web programming ips
   * Build things up from small parts first
     + Test things inside console one piece at a time
     + Put together into actual code
   * Look up Application Programming Interfaces (APIs)
   * Use Google, Overflow
   * Save early and often and with revision control
5. Graphical User Interfaces (GUI)
   * Early GUI flow: Input > App Interface > application > app interface > output
     + Tight coupling: changing UI requires changing input and output code
     + Not modular
   * Key idea one: example is widget: highly reusable interactive components (checkbox, radio, date, password)
     + object-based organization: every element on the screen is an object (early GUI flow on a smaller scale)
     + each object has its own behaviors and states
       - Can draw itself (checkbox that looks like checkbox)
       - Can handle its own input (checkbox can handle presses)
       - Has internal state (checked or not)
     + Creating GUI
       - Selecting appropriate widget
       - Positioning widget in window with HTML
       - Modifying widget property for look and feel
       - Adding code so widget does right thing
     + Widgets
       - Pros
         * Loose coupling, easy to modify: changing one button does not impact other widgets
         * Modular
         * Easy to reuse
         * Clean and higher-level abstraction: reduces complexity
   * Key Idea 2: component tree
     + Every GUI has a tree data structure similar to the DOM, and everything is done through this tree (build interface = build tree, change interface = change tree)
     + Position of child nodes are relative to parent node (top left of parent is (0, 0)
       - Child node position can be relative or absolute
   * Key idea 3: model view controller
     + Split widget into three separate objects, and each part handles different aspect of widget
     + Model (HTML): handles related data > updates view
       - Can have more than one view and controller
     + View (CSS) : handles output
     + Controller (HavaScript): handles input > manipulates model
     + Useful for small and large scale, but when programming, won’t be using small scale since widgets take care of it, but use at large scale for organizing code and overall structure
6. Event driven programming: all input from user to computer done via events (mouse, items, keyboard)
   * Mouse events: click, dblclick, mouseup, mousedown
     + Target: which HTML element
     + Button: which buttom
     + clientX/ clientY: relative to current view
     + pageX/ page Y: relative to top of page
     + screenX/ screen; relative to screen
   * All generated events go to a single event queue
     + Ensures correct temporal ordering of events
     + Allows us to hack on things (ex: combine all mouse motions)
     + Offers level of indirection: can support other inputs in the future
     + Offers loose coupling, so apps don’t need to know what input devices are inputting
7. Modern GUIs
   * Widgets/ component trees/ MVC
   * Loose coupling/ separation of concerns/ reuse
   * Event-driven programming
   * A lot of options, but adaptable
8. Layers of UI software
   * Application, high level tools (libraries for game engines, advertising, facebook, etc.), toolkit (handles widgets), window system, os, software

GitHub

* Mkdir folder\_name
* Cd folder\_name
* Git init
* Git clone /path/to/directory
* Git add <filename>
* Git commit –m “commit message”
* Git push origin master
* Git pull
* Go back previous version: git checkout [revision]
* Replace local changes: git checkout --<filename>--

Lecture 9: Web Programming II

1. How developers work
   * Try
   * Look up documentation
   * Make guess to right approach
   * Look up error message
   * Test code to see what it does
   * Repeat
2. Principles and patterns
   * Separation of content: don’t intermingle things that don’t need to be
     + CSS for style, HTML for content
     + MVC: data from presentation and UI
   * Loose coupling: components should depend on other components minimally
     + Should not know internal of others
     + Should only interact through well-defined APIs
     + VC only interact with M loosely > easy to replace, easy to modify a single part
   * High cohesion
     + Simple but sensible organization
     + Each object and function have simple goal
     + Goal should be predicable based on names
3. Origins of Hypertext: text displayed with references (hyperlinks) to other texts that can be instantly accessed
   * Vannevar Bush: WWII scientific effort
     + Social contract for science
       - Government funds university
       - Universities do research
       - Research bolster economy and national defense
       - Basis of National Science Foundation
   * 1945 “As We May Think”: we have too much information
     + Inventions: wearable cameras, automatic transcripts
     + Memex: first idea of hypertext to link articles together
   * Ted Nelson: coins hypertext and docuverse
     + Micropayments
     + Transclusion, combining of multiple documents
     + StretchText: text is stored as a stream; extra text, invisible at one level, pops in and out of desired altitudes at user control
   * Doug Engelbart
     + Development of hypertext: click on “links”
     + Invented the computer mouse
4. WWW = hypertext + networking
   * No support for graphics until Mosaic by Marc Andreesen
     + <img> tag implemented
     + standardized blue links > hard to read
   * HTML forms later invented: buttons, checkboxes, radio buttons
   * Mostly static pages
   * SQL is for querying databases
   * Difficult to create web pages on the fly
     + Had to use code that generates HTML
     + New system uses HTMl that contains code when page is requested
   * Layers
     + PHP, Python: templating
     + MySQL: dataabsing
     + Apache: web server
     + Linux, Windows: operating system
5. Browser wars > people started inventing new tags
   * + WWW Consortium founded in 1994: establishes standards for compatibility
6. HTTP originally had no notion of continuity: use of cookies (small pieces of data that can be set by server and sent back to server) to keep track of history
7. CSS and JavaScript at the same time
   * CSS: styling separate from content
   * JavaScript: TEMPORARY NAME, HAD NOTHING TO DO WITH JAVA
   * Web Browser (CSS/JS) > Internet (URL/HTML) > Server (PHP, MySQL, Apache, Linux)
8. Content Management Systems (CMS): avoid updating page by finding file and editing it
   * (ex: Drupal, Jakarta, Joomla, Wordpress)
9. Content Distribution Networks (CDN) to alleviate traffic through same channel
   * Global network that puts content geographically closer to users, faster to download
10. Screen scraping: using computer program to copy data from a website
    * Effective but brittle
    * Hard to guess where desired content was
    * Software broke if website changed HTML
    * REST: architectural style to build lightweight, maintainable, scalable web services
      + Service based in REST is called RESTful services
      + HTTP as primary protocol
      + REST-based APIs
11. XML vs. JSON: two common ways of representing data on web; personally equivalent
    * Extensible Markup Language (XML):
      + Similar to HTML, but no predefined tags
      + Formatting has to be 100% correct
      + No semantics > XHTML is XML formatted in HTML
    * JavaScript Object Notation (JSON)
      + Simpler to read than XML
      + Easier to parse in JavaScript
    * Functionally equivalent: JSON preferred, easier to read, more compact
    * Asynchronous JavaScript and XML (AJAX): new technique
      + XHTML for content
      + CSS for style
      + DOM and JavaScript for dynamic content display
12. Evolution based on WIMP: windows, icons, menus, pointer
    * New interaction models coming: speech, gesture, AI systems

Lecture 12: Properties of People

1. Color
   * Mapping of color to reality
   * Adjustments in saturation (purity) or value (brightness) is easier to see than hue (rough color)
   * RGB: technology-centered color: used by LCD monitors
     + Easy to program
   * HSV (Hue, Saturation, Value)
     + Hue: shade
     + Saturation: purity (ex: red is more saturated than pink)
       - Color is mixture of hue and achromatic (white, gray, or black)
     + Value: lightness
     + Uses people intuition, direct conversion to RGB
   * CMYK (Cyan, magenta, yellow, and black): for printing
2. Human Visual System
   * Light passes through lens and is focused on the retina
   * Retina is covered with light-sensitive receptors
   * Rods
     + Night vision and perceiving movement
     + Sense intensity of shades of gray
     + Can’t discriminate between colors
     + 75-150m rods (more rods than cones)
     + Edge of retina: for detecting motion
   * Cones
     + Used to sense colors
     + Gathered at center: allows for high focus
3. Photo pigments used to sense color: blue, green, red/yellow
4. Design tips
   * One: don’t rely on blue for small objects
     + More photo pigments for red than blue
     + Few blue cones in fovea
     + Blue hyperlinks is default worst choice
   * Two: blue not for older users
     + When age, lenses yellow and absorbs shorter wave lengths, blue even harder
   * Three: Minimize saturated colors
     + Greater saturation, more refocusing necessary
     + Use saturation when want something to stand out
   * Four: don’t rely only on hue as cue: use mixture of colors and contrast
     + (stop signs use words and shapes)
   * Five: Design UI in grayscale first: focus on value over hue or saturation
     + Keep luminance/intensity/value same when moving to color
     + Can do in reverse
     + To evaluate UI for people with poor vision, convert screenshot of UI to grayscale
5. Color deficiency
   * 8% males, 0.5% females
   * Two types
     + Different photo pigment response: reduce capability to discern small color differences
     + Red-green deficiency: lacking red or green photo pigment
       - More rare are blue-yellow or total
       - Ishihara Test for color blindness
6. Perception of color is not the same as displayed color
7. Colors is 24 bit for web today (8 each) No 64 big color because humans can’t really see much more colors
8. Human physiology drives design
   * Visuals
     + Number of colors
     + Pixel size
   * Audio
   * Cursor
9. Fitt’s Law: intuitively, things that are closer and/or bigger are faster and easier to hit, and vice versa
   * Tells us about difficulty for pointing and selecting tasks
   * Predicts time to make a movement > A and B are empirically derived constants
     + Time to move the hand depends only on relative precision required
     + Dist and Size
   * Subtle source of usability bug
   * Precision is a variant of Fitt’s law
   * Improve visibility by making objects have bigger target sizes
     + Putting things in corner
     + Making entire label clickable in HTML
     + Or can prevent errors by making target size smaller or further away: increasing difficulty of hitting target, more precision
     + Flow
10. Design of Everyday Things
    * Interface cycle: System > Display > User > Input Devicess > System
    * Bridging users and systems
      + The Gulf of Evaluation: gap between user’s understanding of system and actual system state
      + Gulf of execution: Gap between user’s desires and how to do it
      + UX design is to shrink these gulfs
    * Gulf of evaluation: gap between user’s understanding of sytem and actual system state
    * Gulf of execution: gulf between user’s desires and how to do them

Lecture 13: Properties of People: Addressing gulf of evaluation and execution

1. A more accurate mental model (fridge example)
   * Mental models: describes how a person thinking something works
     + From past experiences
     + Generally, approximations
     + No necessarily one correct model (ex: scientist and native American and fish)
   * Solution
     + Make controls map to user’s model
     + Make controls map to actual system
   * Every system has three different mental models
     + Design model: how you intend the system to work
     + System image: implementation
     + User model: how users think the system works
       - People only interact with system image, so not sure how design model works
2. Affordances: flues to operations that are apparent (often visual, could be speech)
   * Allows and promotes certain actions: door knob > turning
   * Vs. signifiers: affordances are relationships, signifiers are visual cue
   * Virtual
     + Knurling: small ridges on knobs increases friction, affords grip
     + Knurling for a corner for mouse to grip onto
   * Design implications
     + Make sure interactive objects have affordances
     + Many GUI have standard affordances
   * Increasing issues with affordances for sensor-based problems
   * Flat design/material design makes it difficult to know what’s clickable
3. Feedback: response by system to action of user
   * Important for cause and effect, forming mental models
   * Make system state visible (button goes down)
   * Humans can perceive up to <20ms 1/50 sec discrete flashes > don’t need to be faster ~100ms for GUIs
   * <1-2 seconds good response time, ~5 needs something to keep conversation alive, >10-15 bad response time
     + Memory decay effects
     + Need progress meter
     + If can’t meet goal, manipulate user expectation
     + Consistency and predictability of response is more important
   * Response times
     + Programmers intuition is often wrong
     + Perceived response times important too
     + Elevators: mirrors, notification of which floor elevator is on
   * Application to web: people will visit website less if slower by 250ms
   * Feedforward: what will happen if you execute action
     + Web page mosueover
     + Label on a button
     + Help people predict what will happen
   * Short term memory 7+-2 chunks
     + Long term memory is huge but slower access time
     + Chunk menu bars
   * Stroop effect: interference of color word (two or more strong cues I working memory > can lead to slip)
   * Map interface controls: controls should mirror real world
4. Visual Grouping/ Separation
   * Proximity, similarity, connected
   * Shape, color, size
5. Making things distinct: size, value, orientation, texture, shape, position
6. Preattentive processing: ~200 msec to see diff, based on simple and not multiple diffs
7. People have change blindness: subtle changes over time, distractors, discontinuity
8. Metaphor: transference of relation between one set of objects to another set for the purpose of brief explanation
   * To-do lists, calenadrs, drawers (amazon book store)
9. Andy Rubin: Android

Lecture 14: Interaction Techniques

1. Skeuomorphic interfaces: derivative object that retains ornamental attributes from the original
   * Don’t want to be artificially constrained by physical limitations too
   * Tony Fadell (ipod) founded Nest Labs
   * The Danger Sidekick (eventually bought out by Microsoft)
2. Interaction technique: ways of doing interacting with a computer
   * Historically keyboard and mouse, today siri and other voice-based interfaces
   * NLS system: funded by NASA, used hyperlink, mouse, etc., two handed interaction
3. Interaction today
   * Changing contexts: no more office, no more sitting down, mostly quiet
   * Changing users: disabilities, young, elderly
   * Changint asks: fun, creative, on the go
4. Pie Menus: The Sims, Counterstrike
   * Hard to implement: non rectangular harder until mid 1990s
   * Does not scale past a few items: no hierarchy
   * Unfamiliar to people
   * Can block display
   * Relatively small overall gain > use menus a lot
5. Four ways to beat Fitts’ law with pointing tasks
   * Decrease distance
   * Incresae size
   * Do both (pie menus)
   * Avoiding pointing task altogether
6. More ways to beat Fitts’ law
   * Increase size, decrease size, both, avoid
   * Keyboard shortcuts
   * Snapping
7. Drag and drop: predict what person wants, bring target closer
8. Object pointing: cursor jumps to next object in direction of movement with object pointing
   * EdgeWrite: physical overlay and new unistroke alphabet makes it easier for people with disabilities and mobile users
9. Bubble cursor: manipulates distance with resizable cursor so something is always selected
10. Marking menu: pop up menu that lets you do menu selection by making a straight mark in direction of desired menu item
11. Magic lense: see through menu overlaid
12. SHapewriter: gestures + dictionary > easy to use, efficient