

Lecture Notes

CS 160 with Bjoern Hartmann
Spring 2023

Contents

1 Tuesday, January 17th: Welcome to User Interface Design and Development	18
1.1 The dangers of poor UI/UX	18
1.2 Where did we start	18
1.2.1 The Commandline	18
1.3 Course Staff and Communication	18
1.4 Enrollment	19
1.5 This Course	19
1.6 Human-Computer Interaction (HCI)	19
1.7 User Interfaces (UI)	20
1.7.1 Why Study User Interfaces?	20
1.8 Interface Design Cycle	20
1.9 Contextual Inquiry	21
1.10 Rapid Prototyping	21
1.11 Evaluation	21
1.12 Goals of the Course	21
1.13 Teams	22
1.14 Course Mechanics	22
1.15 Office Hours	22
1.16 Sections	22
1.17 Readings	22
1.18 Grading	23

1.18.1	Late Assignments	23
2	Thursday, January 19th: The Design Cycle	24
2.1	UI Critique: Font Selection	24
2.2	BCourses Logistics	24
2.3	Where does design fit into the larger process?	24
2.3.1	Oscillations over Project Lifespan	24
2.3.2	Divergent vs Convergent Phases	24
2.3.3	Waterfall Model (Software Engineering)	25
2.3.4	Agile Software Development	25
2.4	Shopping Cart Video (from abc)	25
2.5	Methods	25
2.5.1	Talking to people	25
2.5.2	Stakeholder Map	26
2.6	Brainstorming	26
3	Tuesday, January 24th: Sketching	27
3.1	Logistics: Discussion Signup	27
3.2	Sketching, Brainstorming, Critique	27
3.2.1	Case Study: Tesla Design	27
3.3	Visions for the future of Car Designs	27
3.3.1	Apple Touchscreen Car Integration (from WWDC)	27
3.3.2	INEOS Grenadier	27
3.4	Sketching	28
3.4.1	Design Journals	28
3.4.2	Storyboard	28
3.5	Brainstorming	28

3.6 Critique	28
4 Thursday, January 26th: Task Analysis	30
4.1 Administrative Details	30
4.2 ChatGPT vs Google	30
4.3 XEROX PARC	30
4.3.1 XEROX 8200	30
4.3.2 Observation Techniques	31
4.4 Task Analysis	31
4.4.1 Case Study: BART Ticket Machine	31
4.4.2 Task Analysis Questions	31
4.5 Old and New Tasks	32
4.6 Learning Tasks	32
4.7 Where is the task	32
4.7.1 Geography of the BART Station	32
4.8 Other Tools	32
4.9 When things go wrong	32
4.9.1 Japanese QR Code Vending Machine	32
5 Tuesday, January 31st: In-Class Team Brainstorm	34
5.1 Logistics: Assignments due!	34
5.2 Team Project	34
5.2.1 Intelligent User Interfaces	34
5.2.2 Designing Inclusive Technologies	35
5.3 In-Class Team Brainstorm: Meeting time!	35
6 Thursday, February 2nd: Contextual Inquiry, Conceptual Models 1	36

6.1	Case Study: Apple Watch	36
6.1.1	Case Study: Calculators	36
6.2	Reminder: Assignments	36
6.3	Contextual Inquiry	36
6.3.1	Goals	36
6.3.2	Context	36
6.3.2.1	Why not just interview folks?	37
6.3.3	Affordances	37
6.3.4	Signifiers	37
6.3.4.1	Case Study: Doors	37
6.3.5	Universal Signals	37
6.3.6	Incorrect signifiers	37
7	Tuesday, February 7th: Conceptual Models (continued)	38
7.1	Make Controls Visible	38
7.1.1	Don't overload the User	38
7.1.2	Make Controls Clear	38
7.1.3	Case Study: Stovetop Controls	38
7.1.4	Use Transfer Learning	39
7.1.5	Provide Feedback	39
7.2	Action Cycle	40
7.2.1	Gulf of Evaluation	40
7.2.2	Gulf of Execution	40
7.3	Interface Languages	41
7.3.1	Semantic & Articulatory Distance	41
7.3.2	Case Study: Adding Autocomplete	41

7.3.3	Spell and Grammar Suggestions	41
7.4	Direct Manipulation	41
7.5	Metaphor	42
7.6	Modes	42
7.6.1	Fixing the Problems with Modes	42
7.6.2	Quasimodes	42
8	Thursday, February 9th: Human Information Processing	44
8.1	Galileo AI	44
8.2	Review: Modes	44
8.2.1	Keyboard Modes: QWERTY, QWERTZ, AZERTY	44
8.2.2	Air France Flight 447	44
8.3	Modeling Human Performance	44
8.3.1	Motivation	44
8.3.2	Model Information Processor Theory	45
8.3.2.1	Perceptual Processor	45
8.3.2.1.1	Michotte Demonstration	45
8.3.2.2	Memory	45
8.3.2.2.1	Attention Span	45
8.3.2.2.2	Long-Term Memory	46
8.3.2.3	Cognitive Processor	46
8.3.2.3.1	Flight Eastern 401	46
8.3.2.3.2	Stroop Effect	46
8.3.2.3.3	Input Stratification	46
8.3.3	Stage Theory	46
8.3.3.1	Recognition over Recall	46

8.4	Decision Making and Learning	47
8.4.1	Hick's Law: Decision Paralysis	47
8.4.2	Power Law of Practice	47
8.5	Pointing	47
8.5.1	Fitts' Law: Distance and Target Size	47
8.5.1.1	Index of Difficulty	48
8.5.1.2	Fitts' Law Tasks	48
8.5.2	The Power of Right-clicking: having options come to you	48
8.6	Bandwidth of Human Muscle Groups	48
9	Tuesday, February 14th: Input	50
9.1	Interface Critique	50
9.1.1	Dual-Screen Devices	50
9.1.1.1	Microsoft Research: Codex	50
9.1.1.2	Microsoft Research: Courier	50
9.2	Administrivia: Midterm 1	50
9.2.1	Scope	50
9.3	Input Devices	51
9.3.1	Text Entry: Keystroke Devices	51
9.3.1.1	DVORAK vs QWERTY	51
9.3.1.2	Mobile Difficulty	51
9.3.1.3	Soft Keys	52
9.3.1.4	Drawing/Handwriting Recognition	52
9.3.1.5	Graffiti	52
9.3.1.6	EdgeWrite	52
9.3.1.7	Stroke writing	53

9.3.1.8	Speech Dictation	53
9.4	Important Device Properties	53
9.4.1	Indirect vs Direct	53
9.4.2	C:D Ratio	53
9.4.3	Device Acquisition Time	54
9.5	Quadrature Encoding	54
9.5.1	Optical Mice	54
9.5.2	Trackpoint	54
9.5.3	Resistive Touchscreens	54
9.5.4	Capacitive Touchscreens	54
9.5.4.1	Exploit the edges	54
10	Thursday, February 16th: Input (contd.) and Prototyping	55
10.1	In the news: Adobe Acquisition of Figma	55
10.2	Input Devices (contd.)	55
10.2.1	Buxton's 3-State Model of Input	55
10.3	Prototyping Theory	55
10.4	The Value of Prototyping	56
10.4.1	Epistemic actions	56
10.4.2	The Value of Surprise	56
10.5	Why Prototype?	57
10.6	Paper Prototyping	57
10.6.1	Wizard of Oz Testing	57
10.7	Testing Device-Based Interfaces	57
10.8	Prototyping in Software	58
10.8.1	Fidelity in Prototyping	58

10.8.1.1 Low-fidelity “Informal” design tools	58
10.8.1.1.1 Advantages	58
10.8.1.1.2 Disadvantages	59
10.8.1.2 High-fidelity visual mockups	59
10.8.1.2.1 Disadvantages	59
10.8.1.3 High-fidelity, fully-interactive prototypes	59
10.9 Video Prototyping	59
11 Tuesday, February 21st: Visual Design	60
11.1 Graphic Design	60
11.1.1 Communication	60
11.1.2 Interpretation	60
11.1.3 History	60
11.1.4 Minimalism	60
11.1.5 Bauhaus Thinking	60
11.1.5.1 Single axis of view	60
11.1.5.2 Grid-based Design	60
11.2 Visual Design	61
11.2.1 Corporate Identity	61
11.2.1.1 Logos	61
11.3 Product Design	61
11.4 Streamlining	61
11.5 Form Follows Function	61
11.6 Simplicity and Elegance	61
11.7 Simplicity	62
11.8 Elegance	62

11.8.1 Reduction	62
11.8.2 Regularization	62
11.8.3 Leverage	62
11.9 Unity	62
11.10 Refinement	62
11.11 Color	63
11.11.1 Color Spaces	63
11.11.1.1 Additive vs Subtractive	63
11.11.2 Perceptual Organization	63
11.11.3 Munsell Color Space	63
11.12 Gestalt Principles	63
11.12.1 Figure/Ground	63
11.12.2 Proximity	63
11.12.3 Similarity	64
11.12.4 Symmetry	64
11.12.5 Connectedness	64
11.12.6 Continuity	64
11.12.7 Closure	65
11.12.8 Common Fate	65
11.12.9 Transparency	65
11.13 Fonts	65
11.13.1 Serif	65
11.13.2 Sans Serif	66
11.13.3 Monospace	66
12 Thursday, February 23th: Midterm 1 Review	67

12.1 Logistics	67
12.2 Composition	67
12.3 Alignment	67
12.4 Common Mistakes	67
12.5 Summary	67
12.6 Review	68
13 Tuesday, February 28th: Midterm 1	69
13.1 Next Class: Thursday, March 2nd	69
13.1.1 Work with your groupmates	69
14 Tuesday, March 7th: Visualization Patterns	70
14.1 Information Visualization, Design Patterns	70
15 Thursday, March 9th: Usability Inspection	71
15.1 Usability Testing	71
15.1.1 Inspection Techniques	71
15.1.2 Cognitive Walkthrough	71
15.2 Example: Find a book in a library	71
15.2.1 Heuristic Evaluation	71
15.2.1.1 Visibility of system status	72
15.2.1.2 Match System and World	72
15.2.1.3 User control and freedom	72
15.2.1.4 Consistency and standards	72
15.2.1.5 Error prevention	72
15.2.1.6 Recognition over Recall	72
15.2.1.7 Flexibility and efficiency of use	73

15.2.1.8 Aesthetic and Minimalist Design	73
15.2.1.9 Help users diagnose recognise	73
15.2.1.10 Provide help and documentation	73
15.3 Heuristic Evaluation Steps	73
15.4 Pros and Cons of HE vs User Testing	74
16 Thursday, March 16th: Empirical Evaluation	75
16.1 GPT4: AI-based Rapid Prototyping	75
16.1.1 Qualitative Empirical Evaluation	75
16.1.1.1 Trends from Qualitative data	75
16.1.2 Quantitative Empirical Evaluation	75
16.1.2.1 Approaches	75
16.1.2.2 Examples of measures	76
16.1.3 Qualitative vs Quantitative	76
16.2 Designing Controlled Experiments	76
16.3 Managing Study Participants	76
16.3.1 The Three Belmont Principles	76
16.4 Ethics	77
16.5 Privacy and Confidentiality	77
17 Tuesday, March 21st: Data Analysis	78
17.1 Managing Study Participants	78
17.2 Treating Subjects with Respect	78
17.3 Data Analysis	78
17.3.1 Bubble Cursor Attributes	78
17.3.1.1 Hypotheses	78
17.3.1.2 Independent Variable	78

17.3.1.3 Dependent Variable	79
17.3.1.4 Control Variables	79
17.3.1.5 Random Variable	79
17.3.1.6 Control Variables	79
17.3.2 Counting Analysis	79
17.3.2.1 Continuous Data	79
17.3.2.2 Categorical Data	80
17.3.3 Cleaning Data	80
17.3.4 Skewed Distributions	80
17.3.4.1 Negative skew	80
17.3.4.2 Positive skew	80
17.3.5 Power Law Distributions	80
17.3.6 Confidence Intervals	80
17.3.7 Hypothesis Testing	81
17.3.7.1 p-values	81
17.4 Statistical tests	81
17.4.1 Between-subject study: Unpaired vs Paired	82
17.4.2 t-Test	82
17.4.2.1 Assumptions	82
17.4.3 Mann-Whitney's U Test	82
17.4.4 Parametric vs non-parametric	82
17.4.4.1 When to use non-parametric tests	83
17.4.5 ANOVA	83
17.4.6 Objective measurements	83
18 Thursday, March 23rd: Native vs Web Apps	84

18.1 Implementing Interfaces, Part 1	84
18.1.1 Xerox Alto (1973)	84
18.2 Native Apps: GUI Toolkits	84
18.3 Native Apps	84
18.3.1 Android SDK Example	85
18.4 Web Apps	85
18.5 Disentangling	85
18.6 DOM (Document Object Model)	86
18.7 Web App with server-side code	86
18.8 Client-side Web Frameworks	86
19 Tuesday, April 4th: Native Events – Widgets, Events, Model-View-Controller	87
19.1 Unexpected Circumstances	87
19.2 Administrative Matters	87
19.2.1 Midterm 1 Regrades	87
19.2.2 Midterm 2 Logistics	88
19.2.3 Scope:	88
19.2.3.1 Readings Scope	88
19.2.3.2 Course curving scheme	88
19.2.4 Next Team Project Assignment	89
19.3 Native App Concepts	89
20 Thursday, April 6th: Browser	91
20.1 Midterm Logistics Review	91
20.2 Implementation	91
20.3 Where are we in the course?	91
20.4 Parsers	95

21 Thursday, April 13th: Collaboration and Social Computing	100
21.1 Post Midterm 2 Feedback Form	100
21.2 Remote Communication	100
21.2.1 Zoom	100
21.2.2 Collaboration Apps	100
21.2.3 Calendars	100
21.3 Challenges to social software	100
21.3.1 Success crisis	100
21.4 Single Display Groupware	101
21.4.1 Roomware	101
21.5 Graph Theory	101
21.6 Power Law Distributions	101
21.7 Time/Space Matrix	102
21.7.1 There goes the neighborhood:	102
22 Tuesday, April 18th: Accessibility	103
22.1 Accessibility Goals	103
22.2 Disabilities	103
22.2.1 Types of Disabilities	104
22.3 Solve for One, Extend to Many	105
22.3.1 Closed Captioning Usages	105
22.4 Specialized Hardware	105
22.5 Visual: Screen Readers	105
22.5.1 Do not cause Seizures!	105
22.6 Navigable	105
23 Thursday, April 20th: Future Interfaces – VR/AR	106

23.1 Virtual and Augmented Reality	106
23.1.1 Changes to costs of VR Hardware	106
23.2 Augmented Reality	106
23.2.1 Video see-through AR	106
23.2.2 HMD-based AR	107
23.2.3 Hardware Complexities	107
23.3 Enabling Technologies/ Open Research	107
23.4 Getting Input	107
23.4.1 3D Glasses	107
23.4.2 Active Shutter	108
23.5 Head-Worn Displays	108
23.6 Audio Displays	108
23.6.1 Auditory Cues	108
23.7 Reverberation	108
23.8 Haptic Displays	108
23.9 Gustation and Olfaction Displays	109
23.10 Professor Hartmann's PhD students' work: Pointing & Selection in 3D	109
23.11 Bi-manual Scaling	109
 24 Tuesday, April 25th: Career Panel	 110
24.1 Speaker List	110
24.2 Midterm #2 Graded	110
24.2.1 Regrades	110
24.2.2 Course Standing	111
24.3 End of the Semester	111
24.4 Design Showcase	111

24.5 Poster Details	111
24.6 Final Demo Details	111
24.7 Q&A	111
25 Thursday, April 27th: Last Class	113
25.1 Final Lecture... but wait there's more	113
25.2 Final Deliverables	113
25.3 Takeaways	113
25.3.1 Why UI is Important	113
25.4 Future Coursework for Undergraduates	114
25.4.1 Graduate School	114
25.4.2 What if I am a Graduate student?	114
25.5 Interested in HCI Research?	115
25.6 Q&A	115
25.7 Course Evaluations	115

Acknowledgements I would like to recognize and thank Professor Björn Hartmann for all information contained in these notes.

Thanks to Minh Phan, Rae Xin, and any others who helped me typeset this.

Disclaimer: These notes are high-level and mainly capture verbal notes not on the slides – there's much more detail on the slides. As such, this represents only a subset of what the class covers.

1 Tuesday, January 17th: Welcome to User Interface Design and Development

1.1 The dangers of poor UI/UX

It is a nice sunny day, you are relaxing on your vacation in Hawaii. Suddenly, you get an alert on your phone that a missile attack is incoming: and mass panic ensues.

But then it is clarified that there was actually no threat and the message was sent in mistake.

How did this happen?

To answer that question, let us look at the UI: We have a dropdown menu with ambiguous acronyms and a confirmation that follows. However the confirmation that follows highlights the “Yes” option as if it is the default.

1.2 Where did we start

1.2.1 The Commandline

In the Command-line, we had a terminal. Now we have a desktop with icons! Modern UI is organized like a physical office: there's a file cabinet with files and a wastebasket.

! But we should not take the metaphor too far:

Making a 3D VR-ish desktop office on your computer has benefits and downfalls. One major downfall is: you are trying to represent 3D via a 2D screen.

1.3 Course Staff and Communication

Here is the CS 160 course staff for this semester!

Bjoern Hartmann: Instructor

Shm Almeda: TA

Peitong Duan: TA

Ace Chen: Reader

Hridhay Suresh: Reader

1.4 Enrollment

This class is oversubscribed with ~100 seats total, split between CS 160 (undergrad) and CS 260A (graduate).

We usually see 5-10% turnover, so if you're in the first 10 positions then you should keep up with the class. Otherwise the chance that you can take the class this semester is unlikely.

Luckily, you can take this class in the summer semester!

Alternatively:

- DES INV 15: Design Methodology
- DES INV 25: UX Design (non-technical)
- IEOR 170: Human Factors
- INFO 213: UI Design and Development
- INFO 114/214: UX Research

are classes you can look into.

! This class has a heavy workload. If you are an EE/CS Major, you already know what upper-division CS classes are like. If you do not think you can handle it, you should drop ASAP to give others a fair chance to get in.

If you are beyond spot 10 on the waitlist, consider leaving now. Likewise if you do not have a seat/cannot see the screen.

1.5 This Course

is about reliably building very good interactive systems.

We focus on **interactions with and through intelligent systems**.

The goal of this class is to build a working **interactive prototype**. We do *not* focus on scaling the class to millions of people, for that CS 169 will moreson help you with that – CS 160 is more for the front-end.

We place emphasis on de-risking via **user testing** and working on rapidly prototyping.

1.6 Human-Computer Interaction (HCI)

Design, prototyping & evaluation of UIs.

Human

- End-user of program

- Others (friends, collaborators, coworkers)

Computer

- The machine the program runs on
- Can be (and often is) split: clients & servers

Interaction

- User tells the computer what they want
- Computer communicates results

1.7 User Interfaces (UI)

What allows people to interact with computers & the computer to communicate results.

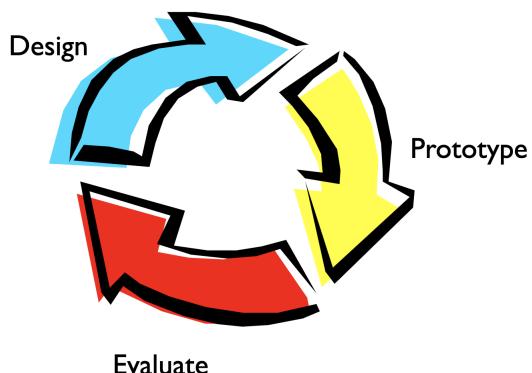
Note that this can include hardware design (i.e. buttons).

1.7.1 Why Study User Interfaces?

- In today's applications, an average of 48% of the code is devoted to the user interface portion
 - It's a major part of work for "real" programs (approx 50%).
- You work on real software intended for others
- Bad UI costs money, lives (see Airplane crash into a canyon due to badly designed UI in autocomplete software that the pilot used), votes, etc
- UI is hard to get right – people are unpredictable.

1.8 Interface Design Cycle

If you take away anything from this lecture, remember this:



1.9 Contextual Inquiry

Observe existing practices.

Create scenarios of actual use.

Create models to gain insight into work processes.

1.10 Rapid Prototyping

Low fidelity can be better as it is:

- Fast, no attachment if you have to scrap it (if you were invested in it then you will stick with a bad UI due to the sunk-cost fallacy).
- No need to debug

1.11 Evaluation

Test with real target users.

Ask other CS 160 UI experts to check a UI for potential accessibility problems using the heuristics and conditions they learned to look for.

1.12 Goals of the Course

Learn to design, prototype, evaluate interfaces:

- Discover tasks of prospective users
- Cognitive/perceptual constraints that effect design
- Techniques for evaluating an interface design
- Importance of iterative design for usability
- Technology used to prototype & implement UI code
- Unique aspects of intelligent user interfaces
- How to work together on a team project
- Communicate your results to a group

These will likely be very important for future jobs.

In other CS classes, you learn algorithms. Here we expect you to know algorithms and teach design.

1.13 Teams

Instructors will form groups of 4-5 students (who are in the same section) in week 2 or 3.

1.14 Course Mechanics

You must be comfortable with programming at the level of CS61B.

Note: Individual programming assignments require you to write code in Javascript, HTML, CSS.

You must be able to attend class and one of the sections each week.

You must commit to working with your assigned team on your group project.

1.15 Office Hours

TBD

1.16 Sections

4 options:

1. 101 – Wed 11am-12pm, 540 Cory
2. 102 - Fri 1-2pm, 310 Soda
3. 103 – Th 10am-11am, 540 Cory
4. 104 – Wed 2 -3pm, 320 Soda

You MUST be able to attend one of the sections.

Section starts next week

1st half of the semester: Lecture material + Tech stack 2nd half of the semester: Design critiques

1.17 Readings

Readings will be posted on bCourses.

Reading responses (recurring assignment) You must post a substantial answer for each assigned reading, **by 10am before class** (so we can review them before class).

Responses are the major factor in your *class participation grade*.

Your first reading response is due next Tuesday by 10am.

1.18 Grading

1. Participation – 10% (Reading responses, class, Ed Discussions)
2. Individual Assignments 20%
3. Two Midterms 30%
4. Group Project Assignments 40%

Note that we generally do the midterms in-class and this may not be accurate – check bCourses.

1.18.1 Late Assignments

- Most assignments will be due before class on the due date
- Individual assignments lose 20% per day (weekend counts as one day)
- Group assignments will not be accepted late

2 Thursday, January 19th: The Design Cycle

2.1 UI Critique: Font Selection

As a designer, one can always benefit from considering multiple interfaces and their respective benefits/tradeoffs.

When choosing what font you want text to be displayed in on Power point, you ahve 2 options:

1. Have a flattened complete list of all combinations of font name, font family, and font weight, rendered in the system font.
2. Have a dropdown that appears on hover with the family and weight, rendered in the font right there for you to see.
 - This gives a visual inconsistency
 - Most will not be used, so slows down the application, with minimal usage for non-graphic designers
 - Does not show what numbers/symbols/etc other visual icons look like

This is not a world-ending decision but there are still tradeoffs. Arguably, neither got it right, which to use depends on the usecase and intended users.

2.2 BCourses Logistics

Make sure you are keeping up with the assignments listed there :D

2.3 Where does design fit into the larger process?

First, realize that there is both stuff before and after Design.

Before: R&D – they create the raw material, look into what matches user needs

After: Engineering (sometimes integrated with design, but sometimes separated), Sales – marketing, feedback loop for re-design

2.3.1 Oscillations over Project Lifespan

The Design Cycle has a dual form which cycles over the number of ideas under consideration; however, this originally large number decays in magnitude to 1, as the project timeline progresses.

2.3.2 Divergent vs Convergent Phases

Divergent: Start with a lot of different ideas/prototypes

Convergent: Over time, you get more narrow and concrete on the concept of what you want to make.

2.3.3 Waterfall Model (Software Engineering)

A linear “waterfall” model where you can only go forward (and therefore don’t allow anything to go backwards) does not work.

This model was used by the Federal Government (specifically wrt contractors) which is probably a key reason behind why their software sucks.

! This may not seem like that big of a deal, but as you get closer to shipping the project, these mistakes cost exponentially more to fix.

Iterative design de-risks your design cycle.

2.3.4 Agile Software Development

In Silicon Valley, you’re more malleable to change. This differs from the similarly essenced iterative design in that this is more technical – this is about writing code.

2.4 Shopping Cart Video (from abc)

In Palo Alto, CA, IDEO is a Design committee looked into making a new, better, shopping cart.
<https://www.youtube.com/watch?v=M66ZU2PCIcM>

Student thoughts on how well they followed the design cycle in the video:

- For the final evaluation video, it seemed kinda rushed.
- Evaluation happened earlier with post-it notes and multiple prototypes

Here Evaluation happened internal to the team (as opposed to user-tested).

They established themes through montages.

2.5 Methods

2.5.1 Talking to people

Talking to people is important, whether it be from asking stakeholders or interviewing experts.

2.5.2 Stakeholder Map

Being visual is important for designers, realize who is at risk as a consequence of your decision(s).

2.6 Brainstorming

1. Sharpen the Focus
2. Playful Rules
3. Number your Ideas
4. Build and Jump
5. The Space Remembers
6. Stretch Your Mental Muscles
7. Get Physical

Aim for quantity, but hope for quality.

3 Tuesday, January 24th: Sketching

3.1 Logistics: Discussion Signup

You can view your discussion assignment here: <https://tinyurl.com/cs160disc-sp23>

3.2 Sketching, Brainstorming, Critique

3.2.1 Case Study: Tesla Design

Tesla Cards do stuff differently:

- They have a large screen in the middle (in portrait mode – vertically long)
- No longer physical buttons – all touch screen

3.3 Visions for the future of Car Designs

3.3.1 Apple Touchscreen Car Integration (from WWDC)

All pixels all the time.

Benefits:

- Shows more information
- Software will get updated
- You don't need to read a complex manual to use the car's basic features
- Cheaper than manufacturing buttons

3.3.2 INEOS Grenadier

All physical buttons.

Benefits:

- More granularity in changing a knob by only a few degrees
- Does not require you to look at it to adjust
- Better in Mountain-like areas without WiFi
- Can be used with gloves/wet hands whereas a touchscreen cannot

3.4 Sketching

3.4.1 Design Journals

Can be a mixture of many different drawings/UI representations/etc.

See attached Drawing Pad for reference.

3.4.2 Storyboard

Disney uses this a lot:

- Combines image frames with text to give a story without committing to all details
- Saves developer time
- Communicates a lot of information/content without being “good” drawings

3.5 Brainstorming

A particular technique for generating a lot of ideas (this is the divergent phase, not the convergent phase).

1. Sharpen the Focus
 - Make sure you have the right focus – not too narrow or fuzzy.
 - Widening your focus can allow you to consider innovative
2. Playful Rules
 - Don’t prematurely reject ideas because they sound too immature or playful
3. Number your Ideas
 - This makes sure that no one person is attached to an idea
4. Build and Jump
 - Exploration vs Exploitation
5. The Space Remembers
 - Use a **lot** of space
 - Giant Post-It Notes
 - External Spatial Memory for your team (i.e. “War Rooms”)
6. Stretch Your Mental Muscles
 - Do puzzles
 - Get immersed in the domain (go somewhere irl)
7. Get Physical
 - Sketch
 - Make Models
 - Act out

3.6 Critique

- ! This is NOT for you to show off how great your project is – you do not learn anything if you are just told that you did a great job.

It is also important not to insult (give feedback on the *design*, not the *designer* dispassionately), ask for specific alternates (instead of suggesting).

This differs from ‘Brainstorming’ as ‘Critique’ is more of an evaluation exercise, and less of a divergent mechanism.

4 Thursday, January 26th: Task Analysis

4.1 Administrative Details

Midterm Schedule set:

1. Tue Feb 28
2. Tue April 11

No Final Exam

But Final Presentations will be on Wed or Thu during RRR week – part of Jacobs Institute Design Showcase! (Schedule TBD)

4.2 ChatGPT vs Google

Similarity & Differences between Web Search (Google) and chat-dialogue interaction (ChatGPT):

- Search engines give multiple results which you can synthesize an answer from, whereas ChatGPT just gives an answer
- Google is helpful if you are in an explanatory phase
- Google gives more sources to cite from
- A singular answer from ChatGPT can be biased, whereas multiple sources from Google can allow us to diversify
- ChatGPT can be more detailed/step-by-step
- ChatGPT can answer questions that it hasn't seen before in its corpus
- ChatGPT has memory, whereas search engines don't explicitly use previously queries
- Google has Images and other integrations, whereas ChatGPT is text-in text-out
- ChatGPT can be confidently incorrect

‘Go beyond intuition, observe target users in context to inform your design’

4.3 XEROX PARC

The first “desktop” computer came from here (IBM and Macintosh were influenced by it), though they were just a printer/copier company.

4.3.1 XEROX 8200

This was sold as a “just click the button and you will be good to go” but in reality, users found that it was too complicated.

Ever very smart people (ACM, Turing Award Winners, Chief Scientists at Powerset/Bing) were unable to operate the machine.

4.3.2 Observation Techniques

User Research:

- Task Analysis
- Contextual Inquiry
- etc (Ethnography/Cultural Probes/Diary Studies)

Goal: Understand User's Activities in Context

4.4 Task Analysis

4.4.1 Case Study: BART Ticket Machine

- Lots of stickers, text, and numbers which overwhelms the user
- People read left-to-right so having earlier tasks more on the left in a noticeable position would be helpful

Solution:

- Stratify into groups (i.e. tourists vs commuters)
- Age varies \implies you cannot make the stand height too high or too low
- Make sure it is accessible to people in wheelchairs

! But this is wrong!

We should actually be talking to real users (found in BART stations) and ask them directly. You are a student so they will be more likely to talk to you than someone who is trying to sell you something; however, you should still make sure to properly compensate them for their time even if it is only a \$5 Starbucks Giftcard.

4.4.2 Task Analysis Questions

1. Who is using the system?
2. What tasks do they now perform?
3. What tasks are desired?
4. How are the tasks learned?
5. Where are the tasks performed?
6. What's the relationship between user & data?
7. What other tools does the user have?
8. How do users communicate with each other?
9. How often are the tasks performed?

10. What are the time constraints on the tasks?
11. What happens when things go wrong?
12. etc

4.5 Old and New Tasks

Now we have clipper cards, Paying with a phone, etc.

4.6 Learning Tasks

1. What do they need to know?
2. Do they need training?
3. Experience, level of education and literacy

4.7 Where is the task

Are there effects of other people (i.e. privacy concerns)

4.7.1 Geography of the BART Station

- Loud
- Privacy
- Lighting is Dim
- Musicians and Rituals

4.8 Other Tools

Smartphones/laptops/Maps are artifacts whose integrations can be relevant for how users communicate.

4.9 When things go wrong

Make sure you have backup strategies

4.9.1 Japanese QR Code Vending Machine

This is a disaster as it required you to type out what product you want on clicking numbers multiple times + receive an email 2FA (requires good internet):

- ! If your machine takes too long to work, then people will not use it. Sometimes people simply cannot use it.

Solution: Make sure tasks are specific.

5 Tuesday, January 31st: In-Class Team Brainstorm

5.1 Logistics: Assignments due!

- Team Brainstorm
 - Aim for ~50 visual ideas!
- Outside of class:
 - Select initial course idea
 - Target specific users (not students)
 - Be creative – your app should not be found in the app store!
- Team Collaborative Plan
 - Meet as a team
 - Figure out team's strength/weaknesses
 - Figure out how/when you'll meet
- Individual Programming Assignment 1: Electric Time
 - Simple web app
 - Deploy, test, debug app locally
 - Document & showcase design + interactive features
 - will be a Personal Transportation Conversion app
 - Section (More HTML + JS) will help with starting this!
- Thursday's Lecture: Conceptual Models + associated Reading Response

5.2 Team Project

Be ambitious, yet realistic.

5.2.1 Intelligent User Interfaces

Example: Audio Fingerprinting a la Shazam.

- Text-to-speech and Speech-to-text
- Translation
- Optical Character Recognition
- Image labeling, face, logo and landmark detection
- Image generation
- Classification and prediction using your own data
- Dialog systems (chatbots)

Example: Automatic Data Entry

Example: Automatic Text Translation

5.2.2 Designing Inclusive Technologies

Abilities change over the course of our lives

Also recall that impairments can be:

- Permanent (i.e. losing an arm)
- Temporary (i.e. arm injury)
- Situational (i.e. new parent with baby in one arm)

5.3 In-Class Team Brainstorm: Meeting time!

For the rest of the class please think of at least 50 app ideas that serve an underrepresented community.

The first group to finish will win free cookies!

The winners were Group 8 consisting of Henry La, Karissa Wong, Priscila Figueroa, Rae Xin, and Rahul Shah.

6 Thursday, February 2nd: Contextual Inquiry, Conceptual Models 1

6.1 Case Study: Apple Watch

You can have a modular home screen or one like traditional watches.

6.1.1 Case Study: Calculators

The 1990 HP-48SX calculator had an android emulator which mimicked it completely with overloaded buttons for easy familiarity at the cost of small hard to push buttons due to the touchscreen.

6.2 Reminder: Assignments

Tue Feb 7:

- Reading Response

Thu Feb 9:

- Team Collaborative Plan
- Team Brainstorm

Mon Feb 13:

- Programming Assignment 1 – Electric Time

6.3 Contextual Inquiry

6.3.1 Goals

- Go to where the customer works
- See what they do
- Talk to them about what they do

This gets you out of your perspective and see their tasks the way they do – a mix between observing and interviewing.

6.3.2 Context

You want details not abstractions – you want to be able to see their work live, in action.

When people are removed from their work environment they can only tell not show.

6.3.2.1 Why not just interview folks?

Folks may be used to inefficiencies and just accept things as “that’s just how things are done”

6.3.3 Affordances

The term affordance refers to the relationship between properties of a physical object and capabilities of a person, that determine how the object could be used.

6.3.4 Signifiers

Signifiers help people figure out the affordances of objects without labels or instructions

6.3.4.1 Case Study: Doors

Q: How to make clear whether you push vs pull to open the door?

A: Make it so that there is only one clear way to do the intended action (i.e. no door handle implies push whereas

6.3.5 Universal Signals

Red means stop;

Green means go.

6.3.6 Incorrect signifiers

! Be careful: Signifiers may suggest affordances that do not exist.

7 Tuesday, February 7th: Conceptual Models (continued)

We start at the **Design Model** and then go to the **System Image** which then iterates back and forth between itself and the **User's Model**.

It is important to realize that the User's Model is *different* than the System Image – the User cannot see your source code!

7.1 Make Controls Visible

Hidden side buttons or controls in hard-to-reach places will make it so that users don't know or won't use features.

7.1.1 Don't overload the User

“Too Much Visibility?” is a thing and will make it hard for the user to find the 1 button they want to press amongst the 200 available choices. Note that we will later formalize this with a function that measures visual search cost (More buttons \implies longer processing time to find a button).

7.1.2 Make Controls Clear

A lot of seat controls have clear mappings from what you want to have happen to the action needed to make that a reality.

7.1.3 Case Study: Stovetop Controls

Which knob controls which burner on a stove is a classic UI question.

Contrary to common opinion, you do not actually need labels to make this unambiguously clear. All you need is to arrange the stoves in a non-linear manner and have the knobs be perturbed in the same manner:

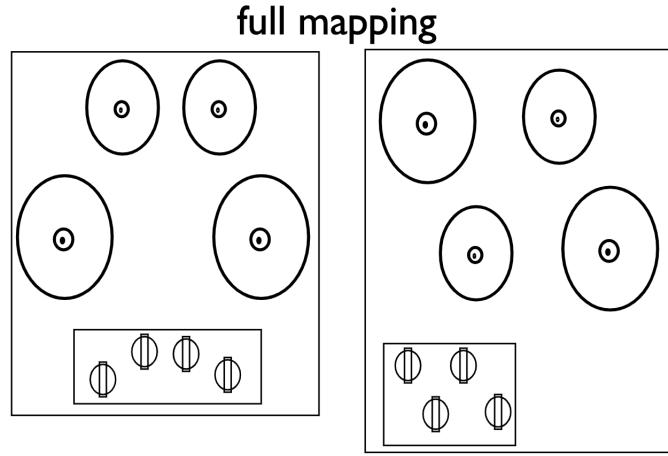


Figure 1: This is unambiguously clear since the knobs are not packed in a horizontal line

7.1.4 Use Transfer Learning

Knowledge from other domains can help users understand how new technologies can work. Laptops have keyboards in a layout that mimics those of typewriters to help people transition to newer technology.

7.1.5 Provide Feedback

Q: Have you ever pressed a button more than once? If so, why?

The answer is likely that you did not receive timely feedback that your input was registered (perhaps you did not press hard enough?).

Making sure you have low-latency is key for good UX.

7.2 Action Cycle

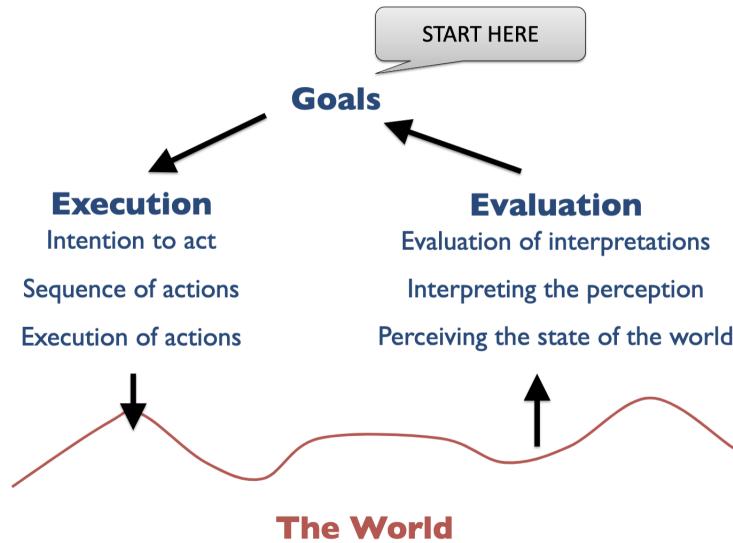


Figure 2: This is a diagram of the Action Cycle

7.2.1 Gulf of Evaluation

Let us say that you are a Data Scientist who is trying to figure out some correlations.

If I can you some pairs of (x, y) in an unsorted order this would be a very non-trivial task. Your first thought may be to re-order the data.

However what if I give you the data as a scatterplot. Then it may be a bit easier to immediately answer as we can use our perceptual systems.

7.2.2 Gulf of Execution

What if I ask you to draw some complicated shapes with a turtle drawing program?

This is non-trivial as you need to calculate some trigonometry. However if I gave you some 'draw some shape' command then it would be much easier to execute this task. Note that the Execution is abstracted – when coding we generally don't think about what is being put into each register in the generated Assembly code.

Gulf of Evaluation goes from *Mental Model* to the *Real World*.

7.3 Interface Languages

An **interface language** is the set of actions a user can take in an interface. These actions have abstract *meaning* and a concrete *physical form*. You can define its input language (provided by user to app) and its output language (provided by app to user)

7.3.1 Semantic & Articulatory Distance

Semantic distance reflects the relationship between the *user's intentions* and the *meaning of expressions* in the interface languages.

Articulatory distance reflects the relationship between the *physical form* of an expression in the interaction language and its *meaning*.

7.3.2 Case Study: Adding Autocomplete

Q: If you added autocomplete to your app, would you be adding Semantic or Articulatory Distance? Would this narrow the gulf of execution or evaluation?

A: This is a trick question – it can be both the gulf of execution (you need to type less keystrokes – lower articulatory distance as you need to do less physical actions) or narrow the gulf of evaluation (as you get to see similar searches – if you don't know how to spell something, your intentions can still be understood and met).

7.3.3 Spell and Grammar Suggestions

Gulf of evaluation (looking at what the UI shows you and interpreting “did this meet my goal or not”) and semantic distances are both lowered since our intention gets matched up to the real world.

7.4 Direct Manipulation

An interface that behaves as though the interaction was with a real-world object rather than with an abstract system

Central ideas

1. Visibility of the objects of interest
 - If you open a phone, your apps are instantly viewable
2. Rapid, reversible, incremental actions
 - You can undo actions
3. Manipulation by pointing and moving

- You can scroll between apps with ease
4. Immediate and continuous display of results
 - There are no hidden states, or compiling, WYSIWYG (what you see is what you get)

GUIs were formed by HCI experts so that people in this room (i.e. non-CS majors) can work with technology.

! Note that if you have used something like L^AT_EX then that is an example of something that is not direct manipulation.

Voice control (like Siri) narrows the Gulf of Execution.

7.5 Metaphor

Metaphor – The transference of the relation between one set of objects to another set for the purpose of brief explanation

This can be useful to help clue users in what the functionality of an interface is and how it might be used.

Not all signifiers are metaphors – metaphors are a type of signifier – all metaphors lend themselves to affordances.

7.6 Modes

This can be useful in applications like Photoshop or Illustrator.

However this can be bad for the User as they need to remember what mode they are in. The SFO airplane crash we talked about in lecture 1 was due to the Pilot thinking they were on autopilot when they were not.

7.6.1 Fixing the Problems with Modes

- Simply do not use modes
- Make it very explicitly clear which mode you are in

The easiest way to go about the second option is to make a redundant UI for the additional modes.

7.6.2 Quasimodes

You have to consciously and continuously take some action to stay in the mode.

The best example of this is the **Shift** key, where if you let go then you exit the mode. This makes sense as you more often than not want to type in lowercase.

This is different from the **Caps Lock** key which completely switches your mode (which can lead to the accident where you write a bunch of text in all caps, essentially unintentionally ‘yelling’ at someone).

8 Thursday, February 9th: Human Information Processing

8.1 Galileo AI

“Galileo AI creates delightful, editable UI designs from a simple text description. It empowers you to design faster than ever.”

8.2 Review: Modes

Last time we went over the Caps Lock key and how it was an example of a mode.

We also talked about quasimodes – the discomfort of having to hold the Shift button down reminds you that you are still in the *uppercase editing* mode.

! If you forget which mode you are in, you will have the source of numerous errors.

8.2.1 Keyboard Modes: QWERTY, QWERTZ, AZERTY

When typing in English vs German, the same buttons can map to letters and muscle memory can work against the User.

8.2.2 Air France Flight 447

In 2009, the plane switched modes due to a broken sensor, without alerting the Pilots. Then when the pilots tried to change the direction, they went into Gimbal Lock, from which they could not recover.

8.3 Modeling Human Performance

8.3.1 Motivation

This is useful to:

- react to situations which we did not see – broader applicability.
- A general solution by falling back on stuff we have seen before.
- Finally, we don't have to have observe things difficulty but we can extrapolate to unseen criteria

So you can react to designs *without actually building them*.

! Note that even today we still don't have a perfectly comprehensive model – we must still fallback upon the **design cycle**.

8.3.2 Model Information Processor Theory

This makes the assumption that humans take in data from a bunch of sensors (embedded systems) that go through a perceptual processor and the brain is just a distributed system consisting of multiple computers.

Note that this is probably inaccurate but it predicts performance well.

8.3.2.1 Perceptual Processor

We take in features that are visual, audio, haptic, etc.

Our attention is selective and is very fast in dropping details. This is done via pre-attentive features (stuff that pop out at you) such as color (most prominently), shapes, curvature, size, convexity, and many more.

! Note that these cannot be combined – pre-attentive features only work in isolation – conjunctions do not work.

Note that the perceptual processor has a cycle time for quantum experience of 100ms.

This is similar to what we see in CS61C w.r.t. how much can be processed, stuff faster cannot be processed.

8.3.2.1.1 Michotte Demonstration

Causality falls off a cliff between 80 and 100 ms and after that, events are seen as independent.

8.3.2.2 Memory

Humans usually repack information as letters or numbers (abstract embeddings with meaning assigned to them) instead of series of pixels with brightness values.

8.3.2.2.1 Attention Span

Humans can usually only remember at most 7 ± 2 things at once.

Attention Span: Interruptions \downarrow decay time.

The more things you are remembering, the longer the access time it will take to remember it will be.

This is analogous to how a CPU only has a limited number of registers for short-term memory.

8.3.2.2.2 Long-Term Memory

In contrast to Short-Term attention span (STM), we also have a longer to access yet bigger Long-Term Memory (LTM).

There is no known limit for the capacity of Human's Long Term Memory.

But it is hard to make sure the semantic encoding goes to LTM and not STM. Remembering/retrieving information is also an issue of concern.

8.3.2.3 Cognitive Processor

Humans are fundamentally not a multi-core machine – you can only focus on one thing at a time – you work in serial (multi-tasking is really just low-latency multiplexing).

8.3.2.3.1 Flight Eastern 401

In 1972, the aircraft crashed due to the crew focusing on a 'landing gear' indicator which overrode the 'crash imminent' indicator to their senses, and made them miss the fact a crash was imminent

8.3.2.3.2 Stroop Effect

The meaning and visual information streams conflict and thus confuse your cognitive processor leading to higher latency.

8.3.2.3.3 Input Stratification

You can calculate the net latency via adding each cognitive cycle for each processor (visual – for reading, processing – for classifying, motor – for pushing a button, etc).

! However this does not work since this is not accurate for complex tasks.

Different tasks have different difficulties. Use pre-attentive features or slow-motion to prolong videos if you want the User to see causality in something that occurs very fast.

8.3.3 Stage Theory

8.3.3.1 Recognition over Recall

Design for recognition over recall (info reproduced from memory).

You have probably heard of Snow White & the Seven Dwarfs but if you are asked to recall the names of the 7 Dwarfs it will probably be non-trivial to do so.

However if I give you a word bank and ask you to choose which are the names, then the task becomes much easier.

8.4 Decision Making and Learning

8.4.1 Hick's Law: Decision Paralysis

The time cost of taking a decision, T , depends on the number of options n :

$$T = a + b \log_2(n + 1) \quad (1)$$

where a, b are empirically derived constants. a = min time it takes to do the task (y-intercept), and b = what human muscle group is being used and how quick that works.

This is why supermarkets make it hard to find products, as well as why they display so many products – to overload your brain into thinking you need more than what you came for.

8.4.2 Power Law of Practice

Task time on the n^{th} trial follows a power law:

$$T_n = T_1 n^{-a} + c \quad (2)$$

Main Idea: **You get faster the more times you do something.**

This means that it will take longer for visitors who are unfamiliar versus experts with the given technology. This also makes it hard to switch to new, unfamiliar, technologies as your muscle memory has already been built.

8.5 Pointing

8.5.1 Fitts' Law: Distance and Target Size

This is a foundational Model for any UI designer to know about.

$$T = a + b \log_2(D/S + 1) \quad (3)$$

where D = distance, and S = size (and $[a, b] = [\text{start/stop time, speed}]$ are empirically derived constants).

8.5.1.1 Index of Difficulty

We can also formulate the Index of Difficulty:

$$ID = \log_2(D/S + 1) \quad (4)$$

This tells us that $T \uparrow$ as $D \uparrow$

and that $T \downarrow$ as $S \uparrow$.

If we have a bigger button then it should be easier and faster to click on. This is due to $S \uparrow$.

8.5.1.2 Fitts' Law Tasks

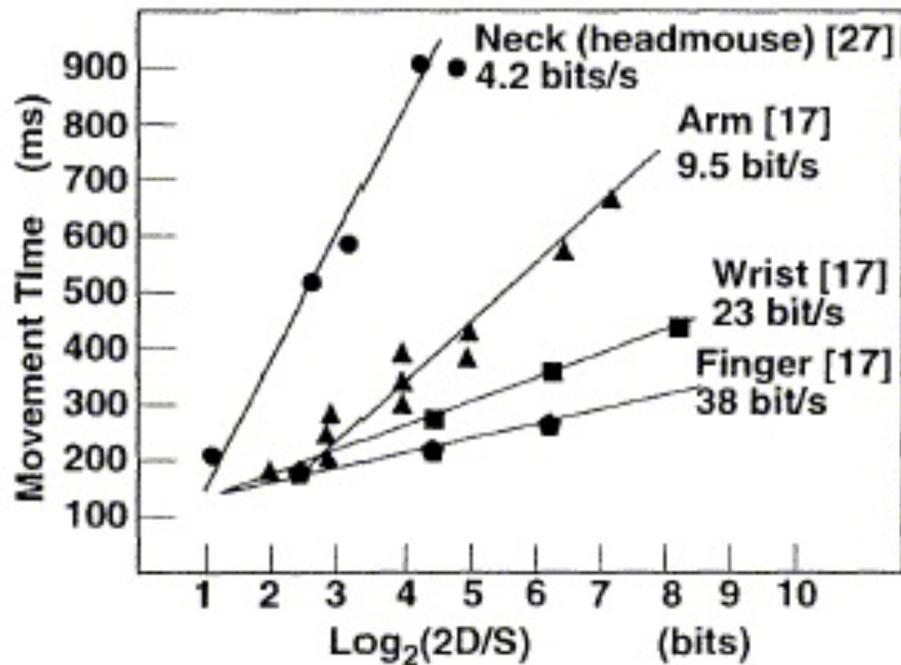
On a touchscreen, tapping and dragging are fundamental Fitts' Law Tasks.

On your laptop, Fitts' Law Tasks are moving your mouse pointer and clicking.

8.5.2 The Power of Right-clicking: having options come to you

Instead of having to drag your cursor across the screen, it is more optimal if you can have the list of options come to you. When you right click, a bar of options appears, this exploits Fitt's Law by making $D \rightarrow 0$.

8.6 Bandwidth of Human Muscle Groups



VR headsets are non-optimal as using fingers instead of neck to click on menus/swipe/move is a 10x speedup.

9 Tuesday, February 14th: Input

9.1 Interface Critique

9.1.1 Dual-Screen Devices

Why should we have these?

- More screen space

Why haven't we seen more of these?

- Moving parts \implies Mechanical Complexity \implies Fall apart quicker
- More expensive \implies lower return on investment
- Aesthetics: there will likely be a crease in the middle
- Not much marginal benefit for the additional pixels

9.1.1.1 Microsoft Research: Codex

This dual screen tablet computer worked well with embedded sensors to detect when the 2 phones were detached.

9.1.1.2 Microsoft Research: Courier

Courier was a prototype concept by Microsoft for a dual-touchscreen tablet. The device was conceived as being a digital notebook, consisting of two 7-inch touchscreens hinged together like a book, and running a custom operating system built primarily around handwriting input and a notebook-like journal for storing notes, images, and clippings from web pages.

9.2 Administravia: Midterm 1

You can take the exam any time in-between 9am Tuesday Feb. 28th to 8:59am Wed, online. You have 90min to complete the exam once it has started (unless you have accommodations).

This will be done via bcourses as a quiz with a mix of MCQ (where you can choose multiple options – select all that apply) and short answer + open-ended questions (can range from a sentence to a paragraph or two).

This gives you an opportunity to show your understanding in a variety of ways.

9.2.1 Scope

Lectures 1 to 11 and associated readings.

Make sure to know basic HTML/JS/CSS concepts + terminology from assignments and sections.

9.3 Input Devices

9.3.1 Text Entry: Keystroke Devices

We have 2 criteria:

1. Fast
2. Low error rate (accurate)

These are good since we can (generally) expect where keys will be on the keyboard,

9.3.1.1 DVORAK vs QWERTY

QWERTY was made to prevent typewriter jams, and was transferred to computers for transfer learning benefits.

DVORAK is not popular as people are used to QWERTY and learning new tech is hard (per the Power law of practice).

9.3.1.2 Mobile Difficulty

If buttons are too small, you run into the “fat-finger problem”

Solutions:

Multi-tap mappings: Press a key multiple times to get a letter

Candybar Phones implemented this (T9 had recognition for 43556 → ‘hello’):



9.3.1.3 Soft Keys

Keys 'f' and 'j' have bumps on a computer keyboard but this tactile feedback information is lost on a touchscreen, which means you **have to** look at the screen to type.

9.3.1.4 Drawing/Handwriting Recognition

Why don't we use Drawing/Handwriting Recognition to input text?

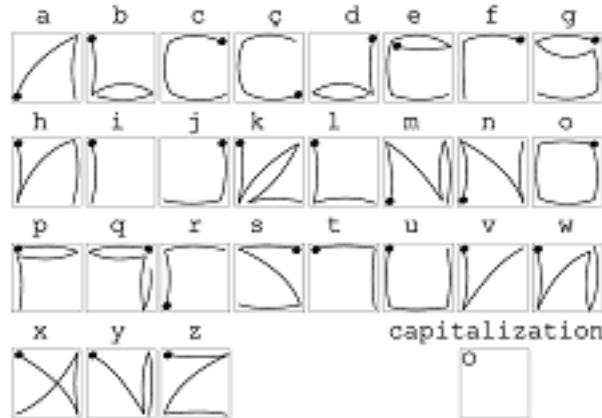
Answer: it's slower than pressing keys

9.3.1.5 Graffiti

Custom alphabet with simplified symbols that are easier to recognize – does not differentiate between lowercase and uppercase though.

9.3.1.6 EdgeWrite

Similar to Graffiti but Corner-based text input technique:



9.3.1.7 Stroke writing

You can also use backend algorithms to make this more accurate

9.3.1.8 Speech Dictation

This is ~ 100 wpm. So why don't we always use this?

- Privacy – anyone nearby can hear
- Not socially acceptable – to interrupt a quiet space with talking
- Can be hard to pickup *only your* voice in a place with multiple people talking
- You have to know what you are going to say, ahead of time and that sort of ‘planning ahead’ is more mentally demanding
 - Editing previous words or even mistakes, is painful and adds more latency than doing the same operation on a keyboard.

9.4 Important Device Properties

9.4.1 Indirect vs Direct

Direct: Input and Output space are unified (touchscreen)

Indirect: You're not pushing a button directly (mouse/trackpad/trackpoint)

9.4.2 C:D Ratio

For one unit of movement in physical space, how far does the cursor travel in display space?

Usually 1:1 for direct touch screen input.

9.4.3 Device Acquisition Time

Time between not using the device and starting to provide input

9.5 Quadrature Encoding

To use sensors for rotary encoding, you need 2 bits of information:

1. Which direction you are moving in
2. How far you are moving in that direction

9.5.1 Optical Mice

The physical mouse has hardware inside (a camera) which detects how much you have moved and which direction – hundreds of times a second.

Fun fact: your mouse can be used as a (very bad) scanner.

It's not good as it doesn't have to be a repeatable proceeding device

9.5.2 Trackpoint

Lenovo Computers are known for having this red dot in the center.

9.5.3 Resistive Touchscreens

Cheap to manufacture but only allows single-touch

9.5.4 Capacitive Touchscreens

Allow multi-touch

- Direct input allows maximal screen space for mobile devices (ocular centrism).
- More degrees of freedom.
- “Virtual input devices” are adaptable.
- No extra pieces to lose or break (stylus!)

9.5.4.1 Exploit the edges

Do not require users to explicitly click on some area/text. Instead allow the user to touch somewhere within some region and allow that to be sufficient to register an input click.

10 Thursday, February 16th: Input (contd.) and Prototyping

10.1 In the news: Adobe Acquisition of Figma

Why did Adobe do this?

- Although Adobe could and did have competing products that accomplished the same task, Figma had already acquired an active userbase
- To consolidate power: Adobe is *the* company for 2-D Graphic Design

10.2 Input Devices (contd.)

! Why don't interfaces designed for one input method work well for another?

Example: Android touchscreen apps on a Chromebook do not support multi-touch.

10.2.1 Buxton's 3-State Model of Input

State	Description
0	<i>Out of Range:</i> The device is not in its physical tracking range.
1	<i>Tracking:</i> Device Motion moves only the cursor.
2	<i>Dragging:</i> Device Motion moves objects on the screen.

One can consider applying Mouse/Touch Screen/Stylus on Tablet to the above.

10.3 Prototyping Theory

There multiple definitions of *Prototype*:

The means by which designers organically and evolutionarily learn, discover, generate, and refine designs. - Lim & Stolterman.

Another definition is:

A representation of a design, made before the final solution exist. - Moggridge, Designing Interactions.

The Industrial Design Process followed this with 8 steps.

This is an example of *Observation and Contextual Inquiry*.

10.4 The Value of Prototyping

10.4.1 Epistemic actions

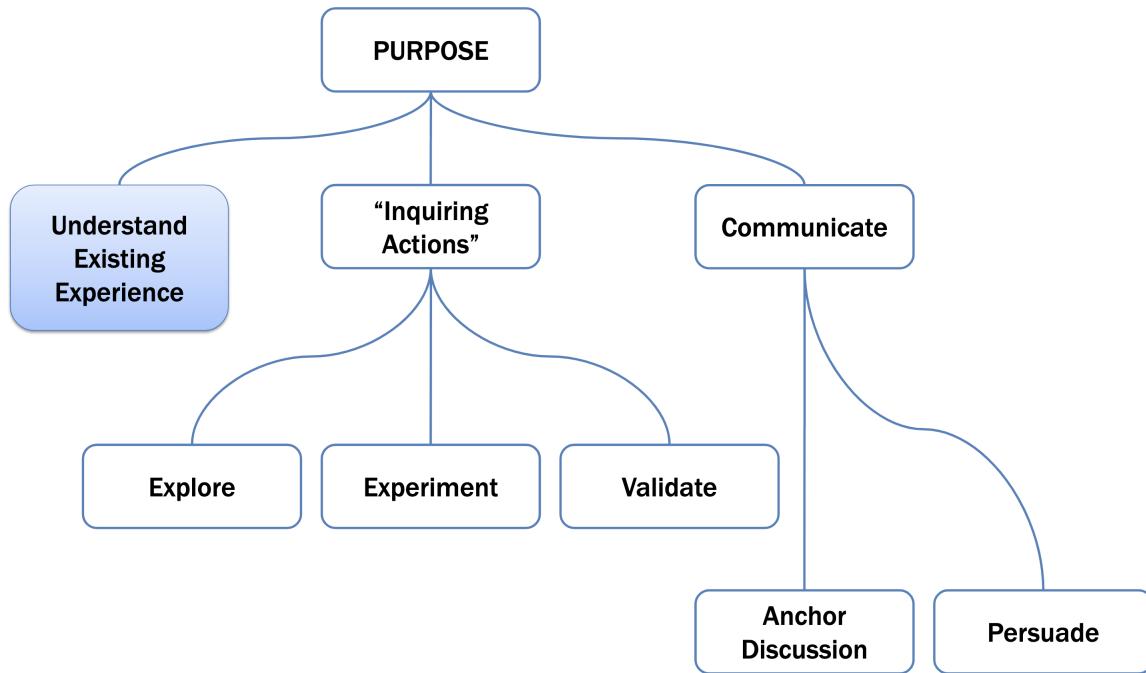
Experts rotate Tetris pieces more than novices as it is easier to visually see what the blocks look like when rotated than to mentally rotate blocks.

10.4.2 The Value of Surprise

The Microwave Oven discovered “by accident” when a networking engineer at Raytheon noticed that the candy bar in his pocket would melt when he would get near to the magnetron radars that produced microwaves.

However the key point here is that the engineer *took action*. They did not just wait for a surprise to come to them, they found something surprising and made a product out of it.

10.5 Why Prototype?



Microsoft had at least 5 different hardware prototypes of the Microsoft mouse, as well as hundreds of paper prototypes before a first version was officially released.

10.6 Paper Prototyping

It's hard to sink a lot of time into paper prototypes, so you won't get super attached to them (they are low fidelity).

Furthermore it is cheap and fast.

10.6.1 Wizard of Oz Testing

Have people test out your prototypes without letting them know what it is, allows you to get real feedback.

10.7 Testing Device-Based Interfaces

When conducting a test, it is preferable to have 3-4 testers.

- **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

A typical session should be approximately **1 hour** as you have to:

- Preparation
- The Test
- Debriefing

! Make sure to record *critical events*.

Critical Events may also be moments when the user

- Got stuck
- Suddenly understood something
- Said “That’s cool”
- Said “Ohhh” etc.

10.8 Prototyping in Software

10.8.1 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.

10.8.1.1 Low-fidelity “Informal” design tools

Examples:

- DENIM (UC Berkeley)
- Balsamiq Wireframes

Goal is to be as rapid and flexible as physical tools.

Add benefits of digital media: Undo, copy+paste, resizing, etc.

10.8.1.1.1 Advantages

- Takes only a few hours – no expensive equipment needed.
- Can test multiple alternatives with the saved time – faster iteration speed
- Can change the design as you test
- Especially useful for black-boxing hard to implement features such as Speech and handwriting recognition

10.8.1.1.2 Disadvantages

- Can be hard to design at the proper scale / complexity / information density of the real UI
- Need to re-create the interface in the target technology
- May not allow real-time end-user interaction

10.8.1.2 High-fidelity visual mockups

- Keynote, Powerpoint
- Flinto+Sketch
- Figma
- InVision
- Adobe XD

10.8.1.2.1 Disadvantages

Distorts 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

10.8.1.3 High-fidelity, fully-interactive prototypes

Should look and behave like the final application.

Takes a lot of effort to build – too little payoff – only use as needed?

Example tools:

- HTML+CSS+Javascript
- Apache Cordova etc

10.9 Video Prototyping

Make sure to demonstrate each of the features/functionality your application will have.

Add structure to better explain content:

- Begin with a title
- Follow with an “establishing shot”
- Create series of closeup & mid-range shots, interspersed with title cards
- Place a final card with credits at the end

Can use stub-motion animation to help do this.

11 Tuesday, February 21st: Visual Design

11.1 Graphic Design

11.1.1 Communication

11.1.2 Interpretation

11.1.3 History

Let's look at the start:

Materials are meant to be read and distributed, which means that you need to make sure texts are laid out in an understandable manner.

Terms such as upper case and lower case came from the separate cases used in printing presses – *terms which are still used today*.

Main idea: history influences the current state of design.

Churches or loyalties – had lots of needs and wanted their words to be replicated. However after the 19th century, people have more disposable income so advertisements appear.

11.1.4 Minimalism

The London Underground is a great example of a simplistic yet ‘does its job’ kind of logo which has withstood the test of time.

11.1.5 Bauhaus Thinking

Lots of Professors from the Bauhaus School of Art fled Nazi Germany and in going to the USA, etc they spread their thinking.

11.1.5.1 Single axis of view

A single vertical axis allows the user to read the text from top to bottom without confusion on where the next word(s) will be.

11.1.5.2 Grid-based Design

Every single element on the page is aligned to horizontal/vertical lines on an underlying grid.

Note that the grid does not have to be square or even axis-aligned, it just needs to be there to give structure.

11.2 Visual Design

11.2.1 Corporate Identity

IBM is a good example of a recognizable identity that is displayed on their posters as a brand mark.

11.2.1.1 Logos

This design of typefaces, size, color, etc is used even in Web Design – think of the BBC news channel: their logo is recognizable and displayed on the top left of <https://www.bbc.com/>

11.3 Product Design

Product Design is about Form and Function.

11.4 Streamlining

This can be helpful but you should not take it too far.

11.5 Form Follows Function

It is the pervading law of all things organic and inorganic,

Of all things physical and metaphysical,

Of all things human and all things super-human,

Of all true manifestations of the head,

Of the heart, of the soul,

That the life is recognizable in its expression,

That form ever follows function. This is the law.

- *Louis Sullivan*

11.6 Simplicity and Elegance

“Good artists borrow (from other artists), but great artists steal!”

- *Pablo Picasso*

11.7 Simplicity

Simple, minimalist, designs are often most effective.

This is likely as there are less ways to interpret them (it has approachability, recognizability, and immediacy).

11.8 Elegance

The scrollbar is a good example of elegant design as it allows scrolling and indicates position in document.

11.8.1 Reduction

Only include essential elements.

11.8.2 Regularization

Use one set of shapes, colors, forms etc.

11.8.3 Leverage

Use elements in multiple roles.

11.9 Unity

One path to simplicity & elegance is through unifying themes:
Forms, colors, components with like qualities.

Think of street signs.

11.10 Refinement

Draw viewers' attention to essential information.

Straighten subway lines to emphasize sequence of stops.

! Mistakes: Clutter & Noise. Don't overdo it (especially with 3-D models on a 2-D screen).

11.11 Color

11.11.1 Color Spaces

11.11.1.1 Additive vs Subtractive

Also known as RGB (red, green, blue) vs CMY (cyan, magenta, yellow).

These are used in Electronic Media and Printed Media respectively.

11.11.2 Perceptual Organization

There are 3-axes of color balance: Colorfulness, Hue, and Lightness. These parameterize our perception.

11.11.3 Munsell Color Space

Perceptually uniform book of painted chips

Pro Tip:
Let Someone Else Pick For You

Some UI frameworks provide default themes... and color resources!

11.12 Gestalt Principles

From Ware's 04 Paper:

11.12.1 Figure/Ground

Relative size.

11.12.2 Proximity

Introduce spacing/underlying grid.

Group related elements – use size and typeface to allow scanning for groups.

11.12.3 Similarity

Can allow you to draw attention to one over the other $\in \{\text{rows, columns}\}$.

11.12.4 Symmetry

Bilateral symmetry gives strong sense of figure.

11.12.5 Connectedness

Connectedness overrules proximity, size, color shape.

11.12.6 Continuity

We prefer smooth not abrupt changes.

Connections are clearer with smooth contours.

11.12.7 Closure

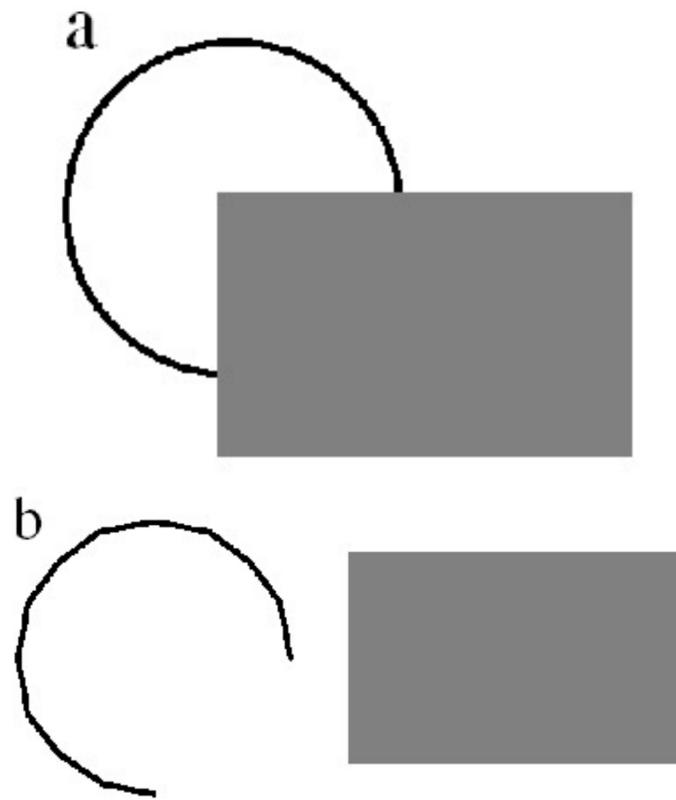


Figure 3: We see a circle behind a rectangle, not a broken circle.

11.12.8 Common Fate

Dots moving together are grouped

11.12.9 Transparency

11.13 Fonts

11.13.1 Serif

Flags/projections jetting off from the top of letters.

11.13.2 Sans Serif

Translates to “Without Flags”

11.13.3 Monospace

A font where the space in between letters are fixed. The only use they have is for code/data/text where vertical alignment matters (i.e. Python code).

! You should not use this for long texts.

12 Thursday, February 23th: Midterm 1 Review

12.1 Logistics

The quiz will be on bcourses under the “Quizzes” section. We will post a clarification form and have times on when we will monitor Ed posted.

12.2 Composition

Use grid systems – what most web apps use!

Note that a text can be multiple grid cells but you mentally think of it wrt multiples of grid cells with whitespace in between.

12.3 Alignment

Every item on a screen has a relationship to the other items. Elements that are almost collinear should be aligned.

Left, right and both-justified alignments create strong boundaries around a piece of text – do not center align text as you do not have an easy side border to read in a deterministic non-causal manner.

Its best to stick with one kind of justification within a page.

12.4 Common Mistakes

- Arbitrary component positions and dimensions
- Random window sizes and layouts
- Unrelated icon sizes and imagery
- Poor alignment

12.5 Summary

What are the 3 stages of design cycle?

The Design Cycle is not everything.

As you go through the design cycle, you consider lots (≈ 50 in brainstorming) but you end up shipping 1 product (divergent \rightarrow convergent phases).

12.6 Review

This has been added to the previous sections, as is relevant.

Good exam questions:

- If I tell you an object, tell me what its affordances are?
- Give an example of a mode that wasn't mentioned in lecture slides.
- Give an example of a task that follows a power law (i.e. typing).

13 Tuesday, February 28th: Midterm 1

There is no class today due to the midterm!

13.1 Next Class: Thursday, March 2nd

13.1.1 Work with your groupmates

There is also no class today to give you an extra chance to work on your project!

14 Tuesday, March 7th: Visualization Patterns

14.1 Information Visualization, Design Patterns

Verbal Notes not given in lecture slides:

- EDA: Exploratory Data Analysis is used for performing critical investigations in data to find patterns
- Start with Color/Orientation/Shape so you're not that constrained for adding O/Q features in the future
- Delta Stimulus
- Loudness = approx 6.5
- Heaviness and taste greater than 1 \implies we overestimate those sensations.
- Certain Visual Variables are better for comparisons.

15 Thursday, March 9th: Usability Inspection

Empirical studies are very expensive in terms of time and money

15.1 Usability Testing

15.1.1 Inspection Techniques

1. Cognitive walkthroughs: put yourself in the shoes of the user. Check to see if the path users want to go is clear and logical.
2. Heuristic Evaluation: assess based on predetermined criteria / heuristics
3. Other non-inspection techniques (mechanical turk)

15.1.2 Cognitive Walkthrough

1. Get concrete goal
2. get actions needed to complete said goal
3. At each step as
 - (a) Will the users know what to do?
 - (b) Will the user notice that the correct action is available?
 - (c) Will the user interpret the application feedback correctly?

15.2 Example: Find a book in a library

1. find the library website
2. find location of said book
3. complete search form
4. parse through to find latest edition
5. click to get to the book's page
6. find library and call number

15.2.1 Heuristic Evaluation

1. Heuristic: rules of thumb that describe features of usable systems
2. Example: Minimize user's memory load
 - (a) get small number (3-5) people together to look at the UI separately
 - (b) give them list of heuristics
 - (c) get them to evaluate your design
 - (d) aggregate findings of the independent evaluators and discuss
 - (a) H2-1: Visibility of system status
 - (b) H2-2: Match system and real world
 - (c) H2-3: User control and freedom
 - (d) H2-4: Consistency and standards

- (e) H2-5: Error prevention
- (f) H2-6: Recognition rather than recall
- (g) H2-7: Flexibility and efficiency of use
- (h) H2-8: Aesthetic and minimalist design
- (i) H2-9: Help users recognize, diagnose, recover from errors
- (j) H2-10: Help and documentation

15.2.1.1 Visibility of system status

1. Spiny ball
2. buffering circle
3. provide redundant information for each decision a user makes (do you want to save changes
⇒ your changes will not be saved if you don't)

15.2.1.2 Match System and World

1. follow real world conventions
2. pay attention to metaphors
3. Ex: used to drag floppy disk to trash to eject it. Did not line up with what trash usually does (data deletion)

15.2.1.3 User control and freedom

1. users don't like being trapped
2. give them freedom to choose to cancel, escape, universal undo (ctrl-z), postpone, etc.

15.2.1.4 Consistency and standards

1. don't violate standards since they will behave according to how they expect things to behave
2. example: put a word in a rounded rectangle implies its a button. People will click it.

15.2.1.5 Error prevention

1. Check for errors (are you sure you want to do that)
2. visual metaphors can help clear up incongruities
3. types of errors: slips (right plan, bad execution) and mistakes (wrong plan, good execution).

15.2.1.6 Recognition over Recall

1. People are good at recognition and bad at recall
2. Example: open recent files tab

15.2.1.7 Flexibility and efficiency of use

1. Experts want to save time
2. Example: keyboard shortcuts, autocomplete

15.2.1.8 Aesthetic and Minimalist Design

1. Don't overload screen with information. Give them "details on demand"
2. Occam's razor: remove or hide irrelevant or rarely needed info
3. Present information in natural reading order (read left to right, top to bottom)

15.2.1.9 Help users diagnose recognise

1. When errors happen, help the users answer the "what do I do next?" question
2. write good error messages that are descriptive and offer solutions
3. Let them undo at every step

15.2.1.10 Provide help and documentation

1. easy to search
2. focused on tasks
3. list concrete steps to carry out
4. not too long
5. examples
 - (a) tutorials
 - (b) reference manuals
 - (c) tool-tips
 - (d) "whats this" cursor
 - (e) search help bar

15.3 Heuristic Evaluation Steps

1. Pre-evaluation training Provide the evaluator with domain knowledge if needed 2) Evaluation Individuals evaluate interface then aggregate results Compare interface elements with heuristics Work in 2 passes First pass: get a feel for flow and scope Second pass: focus on specific elements Each evaluator produces list of problems Explain why with reference to heuristic or other information Be specific and list each problem separately
- 3) Severity rating Establishes a ranking between problems Cosmetic, minor, major and catastrophic First rate individually, then as a group 4) Debriefing Discuss outcome with design team Suggest potential solutions Assess how hard things are to fix

15.4 Pros and Cons of HE vs User Testing

1. Pros: Much faster, doesn't require interpreting user actions
2. Cons: HE is less accurate, may find false positives (things that conceptually could be problems but are not actually), and need multiple evaluations to be done for verification (5 is good)

16 Thursday, March 16th: Empirical Evaluation

Note there was no lecture on Tuesday.

16.1 GPT4: AI-based Rapid Prototyping

GPT4 was recently announced and it can be useful for rapid prototyping. You can say what you want in words and almost automatically view a design render.

16.1.1 Qualitative Empirical Evaluation

Contextual Inquiry: try to understand user's tasks and conceptual model

Usability Studies: look for critical incidents in interface

These Qualitative Empirical Evaluation methods allow us to:

- Understand what is going on
- Look for problems
- Roughly evaluate usability of interface

16.1.1.1 Trends from Qualitative data

Grounded Theory (Glaser, Strauss) is a way to systematically produce insights (hypotheses or theories) from qualitative study data.

Process:

1. Review the collected data, look for concepts or ideas that emerge repeatedly.
2. Coding: Tag these concepts using codes (shorthand descriptions). Codes can be revised throughout the process and can also be hierarchically grouped.
3. Memoing: “The theorizing write-up of ideas about the codes” – relate codes to each other, reflect on the implications

Software tools like MaxQDA can help you here!

16.1.2 Quantitative Empirical Evaluation

Use to reliably measure some aspect of interface

Compare two or more designs on a measurable aspect

Contribute to theory of Human-Computer Interaction

16.1.2.1 Approaches

Collect and analyze user events that occur in natural use

Controlled experiments

16.1.2.2 Examples of measures

Time to complete a task, Average number of errors on a task, Users' ratings of an interface[†]

[†] You could argue that users' perception of speed, error rates etc is as important than their actual values

16.1.3 Qualitative vs Quantitative

Qualitative: Faster, less expensive → esp. useful in early stages of design cycle

Quantitative: Reliable, repeatable result → scientific method
Best studies produce generalizable results

16.2 Designing Controlled Experiments

1. State a lucid, testable hypothesis
2. Identify variables
(independent, dependent, control, random)
3. Design the experimental protocol
4. Choose user population
5. Apply for human subjects protocol review
6. Run pilot studies
7. Run the experiment
8. Perform statistical analysis
9. Draw conclusions

16.3 Managing Study Participants

16.3.1 The Three Belmont Principles

Respect for Persons

Have a meaningful consent process: give information, and let prospective subjects freely chose to participate

Beneficence

Minimize the risk of harm to subjects, maximize potential benefits

Justice

Use fair procedures to select subjects (balance burdens & benefits)

16.4 Ethics

Always ask the question: could the information from a study have been obtained through some other means?

16.5 Privacy and Confidentiality

Privacy: having control over the extent, timing, and circumstances of sharing oneself with others.
What are you asking them to reveal about themselves?

Confidentiality: the treatment of information that an individual has disclosed with the expectation that it will not be divulged.

How are you keeping the information after it has been shared?

17 Tuesday, March 21st: Data Analysis

17.1 Managing Study Participants

It can be a stressful experience for Participants, which is why we follow the 3 Belmont Principles.

What kind of questions are you asking about – is it a sensitive topic that they don't feel comfortable sharing?

Are you going to give literal personally-identifiable info? Or will you deal in *aggregates* (as you should).

Ask yourself the question: Will the person be identifiable from the information shared?

17.2 Treating Subjects with Respect

If you are doing this at a University, you need to go through IRB (Institutional Review Board).

How to let people in the study know:

- Give them a consent form which they must read and sign – you don't need to make this from scratch – most universities have tools to generate such a form with correct legal dictation.
- Keep a copy of the form, while making sure the participant has a copy

Don't waste their time – debug your experiments beforehand & have everything ready.

Make users comfortable (this is a big thing in VR studies where wearing the headsets for extended periods of time can become tiresome).

17.3 Data Analysis

Now that you have the data, how do you analyze?

17.3.1 Bubble Cursor Attributes

17.3.1.1 Hypotheses

- Quicker
- More accurate/Less Errors

17.3.1.2 Independent Variable

Cursor type (bubble vs normal).

Placement of the bubbles (D in Fitts' Law).
Size (S in Fitts' Law).

17.3.1.3 Dependent Variable

1. Time
2. Accuracy

17.3.1.4 Control Variables

Think about this yourself!

17.3.1.5 Random Variable

Think about this yourself!

17.3.1.6 Control Variables

Resulting data: usually in a .CSV file.

Attributes:

- Unique ID
- Cursor Type
- Size
- How many times it took
- Time
- etc

17.3.2 Counting Analysis

We can make descriptive statistics such as:

$$\mu = \frac{\sum_{i=1}^N X_i}{N}$$
$$\sigma = \sqrt{\frac{\sum_{i=1}^N (X_i - \mu)^2}{N}}$$

17.3.2.1 Continuous Data

Central tendency:

- mean
- median
- mode

Dispersion:

- Range (max-min)
- Standard deviation

Shape of distribution:

- Skew
- Kurtosis

17.3.2.2 Categorical Data

Frequency Distributions (Note that there is more you can do here (i.e. error bars).

17.3.3 Cleaning Data

If you don't visualize your data, you may have no idea that you have a bunch of bad data points – that need to be removed.

17.3.4 Skewed Distributions

The Median is more robust to outliers.

17.3.4.1 Negative skew

mean < median < mode.

17.3.4.2 Positive skew

mode < median < mean.

17.3.5 Power Law Distributions

The distribution of photographers contributing photos of the *2005 Coney Island Mermaid Parade* is carried by a few contributors doing 100+ photos, and the rest doing much less (for an average of 26, and median of 11).

$$p(x) \propto x^{-\alpha}$$

17.3.6 Confidence Intervals

95% Confidence Interval: The range of values in which we're 95% sure the true population mean falls.

This can be calculated with the help of the standard error SE.

Standard Deviation: measures variability of individual data points.

Standard Error: measures variability of means

$$SE = \frac{SD}{\sqrt{N}}$$

$$95\%CI = M \pm 1.96 \times SE$$

17.3.7 Hypothesis Testing

Hypothesis: Manipulation of IV effects DV in some way

Null hypothesis: Manipulation of IV has no effect on DV

Null hypothesis assumed true unless statistics allow us to reject it

17.3.7.1 p-values

We need a cutoff for results to be significant $p < 0.05$ usually considered significant (Sometimes $p < 0.01$).

Means that $< 5\%$ chance that null hypothesis is true – Likelihood that results are due to chance variation.

17.4 Statistical tests

- T-test (1 factor, 2 levels)
- Correlation
- ANOVA (1 factor, > 2 levels, multiple factors)
- MANOVA (> 1 dependent variable)

When to use which test:

	Interval/Ratio (Normality assumed)	Interval/Ratio (Normality not assumed), Ordinal	Dichotomy (Binomial)
Compare two unpaired groups	Unpaired t test	Mann-Whitney test	Fisher's test
Compare two paired groups	Paired t test	Wilcoxon test	McNemar's test
Compare more than two unmatched groups	ANOVA	Kruskal-Wallis test	Chi-square test
Compare more than two matched groups	Repeated-measures ANOVA	Friedman test	Cochran's Q test
Find relationship between two variables	Pearson correlation	Spearman correlation	Cramer's V
Predict a value with one independent variable	Linear/Non-linear regression	Non-parametric regression	Logistic regression
Predict a value with multiple independent variables or binomial variables	Multiple linear/non-linear regression		Multiple logistic regression

17.4.1 Between-subject study: Unpaired vs Paired

Between-subject study: We cut our participants into 2 groups, which each group trying a different cursors. This is an unpaired group.

If everyone used both cursors, then it matters that these measurements both came from the same person as that would be paired groups.

17.4.2 t-Test

1 independent variable with 2 levels,
1 dependent variable of type Q (quantitative/interval)

Compares the means of 2 groups
Null hypothesis: No difference between means

17.4.2.1 Assumptions

- **Samples are normally distributed**
- Very robust in practice
- **Population variances are equal (between subjects tests)**
- Reasonably robust for differing variances
- **Individual observations in samples are independent**

t-Test is somewhat robust to variations in skew as long as the distribution looks somewhat normal.

This is a parametric test.

17.4.3 Mann-Whitney's U Test

1 independent variable with 2 levels,
1 dependent variable of type O (ordinal), or type Q when assumption of normal distribution does not hold.

Compares the ranks of values of 2 groups

This is a non-parametric test.

17.4.4 Parametric vs non-parametric

t-Tests are parametric as it makes the assumption on the data's distribution (that it is normal).

Non-parametric test are when one does not rely on a particular distribution.

17.4.4.1 When to use non-parametric tests

Preference data is the most frequently produced ordinal data (Strongly Agree vs Agree vs Neutral vs Disagree vs Strongly Disagree – so one should use a non-parametric test here).

17.4.5 ANOVA

Single factor analysis of variance (ANOVA):

Compare means for 3 or more levels of a single independent variable.

Multi-Way Analysis of variance (n-Way ANOVA): Compare more than one independent variable

Can find interactions between independent variables

ANOVA tests whether means differ, but does not tell us *which means* differ – for this we must perform pairwise t-tests.

17.4.6 Objective measurements

- Good internal validity → repeatability
- **But, real-world implications may be difficult to foresee**
Significant results doesn't imply real-world importance
3.05s versus 3.00s for menu selection

18 Thursday, March 23rd: Native vs Web Apps

18.1 Implementing Interfaces, Part 1

We have talked about the design process as the cycle that happens in the red phase. In the programming assignments, you have been in the “Engineering” phase.

Before we got to GUIs, everyone interacted with computers through command-line prompts, a model where interaction is done via the system.

18.1.1 Xerox Alto (1973)

The system is waiting for an interaction and responds once that has happened. This was one of the first interactive programs.

One way you can do this is with a switch statement in an infinite while loop but this only works if your computer is only running 1 process. The reason for this is because they were written by different people.

18.2 Native Apps: GUI Toolkits

Today most native apps are written via toolkits such as QT, Cocoa, Java Swing, GTK, etc.

18.3 Native Apps

These are self-contained programs made to run directly on a target OS (Windows, Linux, MacOS, Andriod, iOS, etc).

There are a multitude of UI toolkits even if there is a ‘canonical’ toolkit for the OS, and third-party toolkits can help you get closer to the platform-independence that web apps. However this has the cost of not looking like it was *meant* for the current OS.

An advantage of Toolkits over Web Apps is that they are generally quicker in updating to allow using new hardware.

Note that Programming Language is orthogonal to the toolkit. You can use almost any language for almost any toolkit.

! This means that you will have to have an ‘Android Team’ and an ‘iOS Team’ – even if you share some code, the frontend UI will be split across 2 codebases with completely different code.

18.3.1 Android SDK Example

- Highest Level

View Class

Activity System

Native Libraries + Android Runtime

Linux Kernel

- Lowest Level

18.4 Web Apps

This semester we are working on web applications which are delivered by a *web server* and run on a *web browser*.

A *Client* requests resources from server.

A *Server* transfers data for UI and interaction logic to client.

A *Browser* on the client parses received data, builds & renders the UI and processes user input events

You can refer to names in the DOM in JavaScript – a language for defining interactive behavior.

Note that with different OS/vendors, you may see some slight differences.

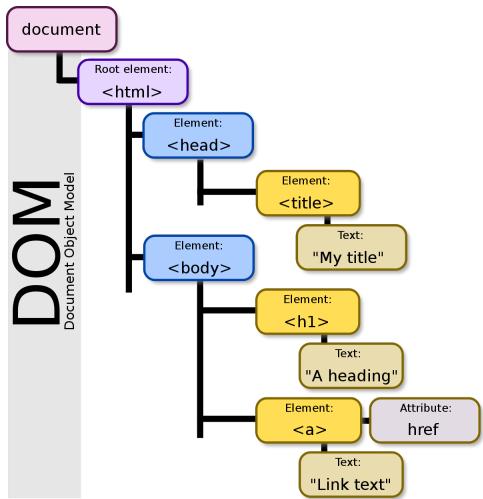
There are Pros/Cons to using Web & Native Apps and these can be seen in the table in lecture slides.

18.5 Disentangling

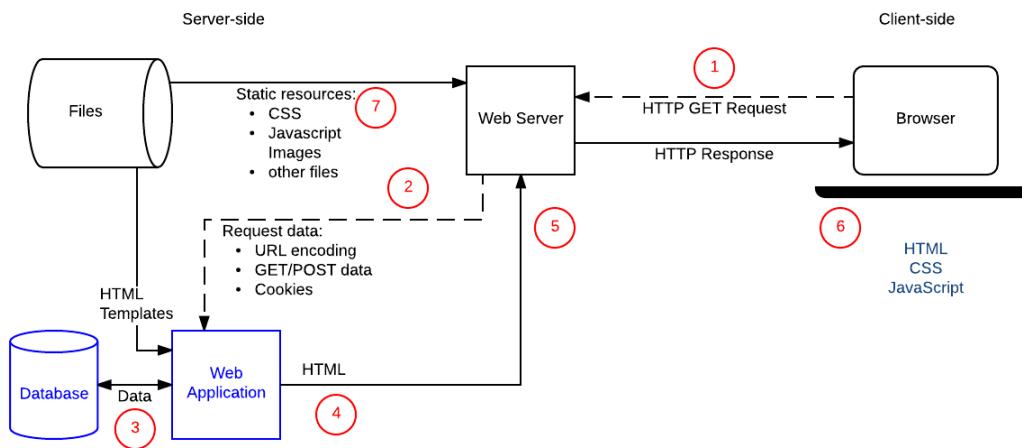
We usually separate HTML DSL from CSS/JS code.

Instead of using `<style>` and `<script>` tags, we designate explicit `.css` and `.js` files for CSS & JavaSctipt code respectively.

18.6 DOM (Document Object Model)



18.7 Web App with server-side code



18.8 Client-side Web Frameworks

HTML was made for *Documents* – not UI.

- React JS (Facebook)
- Angular JS (Google)
- JQuery

Note that JQuery is still considered a Client-side Web Framework even though it does not really change how your server communicates like the others.

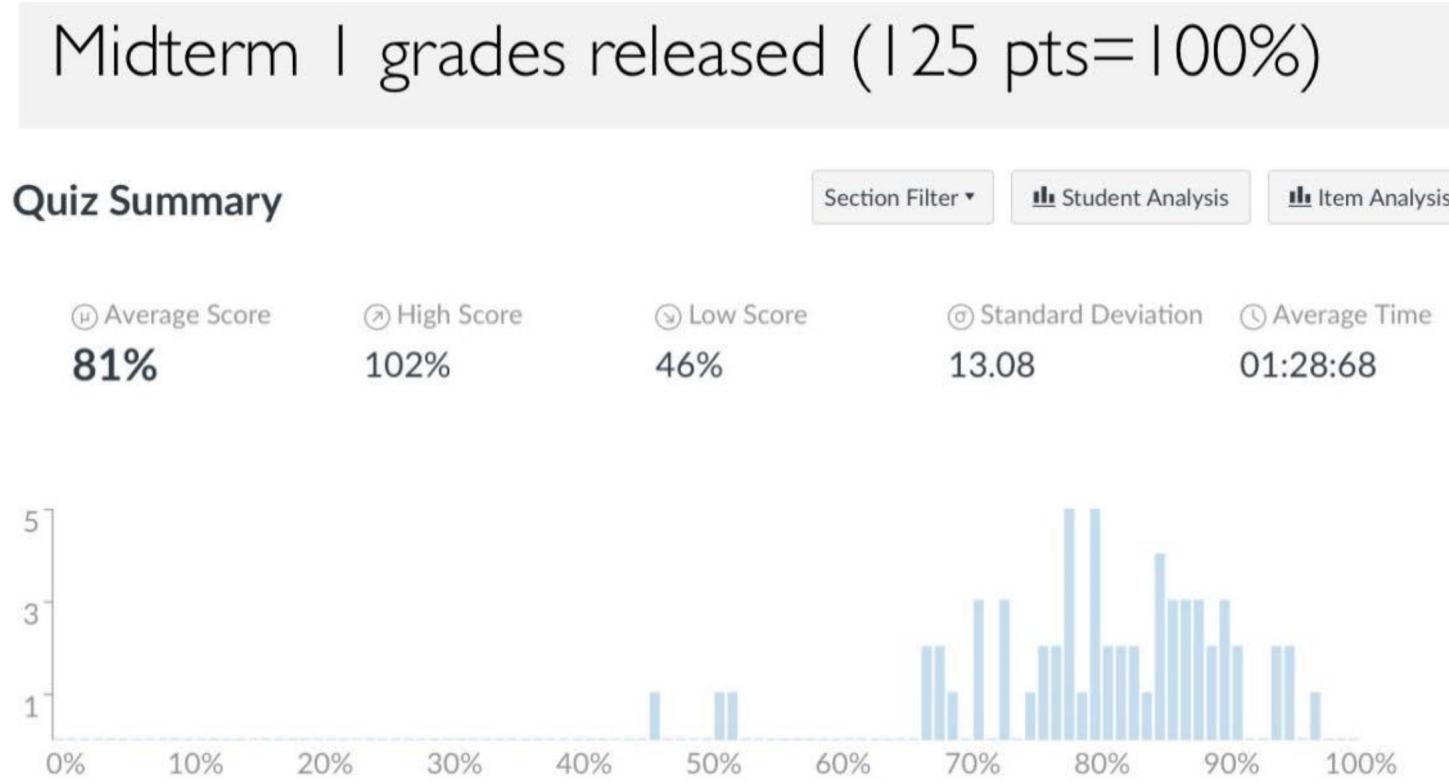
19 Tuesday, April 4th: Native Events – Widgets, Events, Model-View-Controller

19.1 Unexpected Circumstances

The Professor tested positive for COVID, and thus class will be virtual for today and Thursday.

19.2 Administrative Matters

Midterm 1 grades have been released. The midterm was out of 125 points total (though there were up to five extra credit points). The following statistics are about normal for an exam in CS 160.



19.2.1 Midterm 1 Regrades

Standard regrade policy applies here to request a regrade, please write a detailed explanation for your request and submitted via the regrade request form that's on the syllabus for the course. Please do that within one week. Do not email directly – then Course Staff just lose track of all the requests. Make sure to do this especially if the point total on the question does not match the feedback you've been given. Or if we just forgot this to grade something, we're fallible, this

happens, sorry, we will fix that. Please do not argue with us about partial credit. Credit on a given question. Since like one person consistently graded a poll question for everyone in the class. And so we tried to be very consistent in how we assign partial credit. So your number is in similar in relation to other students who provided a similar answer.

19.2.2 Midterm 2 Logistics

Midterm #2 will take place next week, Tuesday April 11th 2023.

The format will be the same as midterm #1 – it will be released the midterm online through bcourses at 09:00 A.M.

You will have 90 min within a 24-hour time, 24 hour time window to take the exam which will once again have a mix of multiple choice and open answer questions.

- Midterm #2 is not cumulative. This is **not** a final exam. We will only ask about material that was not yet covered in midterm #1.

19.2.3 Scope:

This means the material we'll cover is that from lectures 14 through 20, information visualization, usability inspection methods, usability evaluation, data analysis, and three lectures on implementation of which we are in the second right now.

19.2.3.1 Readings Scope

Readings: we will not ask you about information visualization readings since we made that optional. We will ask you about the two Martin chapters on experiment design interpreting data. The heuristic evaluation links provided in the heuristic evaluation assignment are also fair game – they should pretty much match what we covered in class. The last reading on how browsers work is also in-scope.

19.2.3.2 Course curving scheme

Question: Is this class curved?

Answer: This class is not usually curved. Usually we get just about what we want in terms of a an upper division or a master's class great distribution by just taking straight values. Usually the exams tend to be a little lower than the project assignments and the homework assignment grades. We will definitely not curve down. We can take a look if all the values are significantly below where they should be. If so, then we curve up but haven't had to do that in the past – so I would not expect that to happen this year either.

19.2.4 Next Team Project Assignment

The next team project assignment is a heuristic evaluation. Now, heuristic evaluation is a discount usability method as we discussed, that should be quick to do. That's why we only were only assigning one week. And in fact, you will do most of the work for this assignment. In section this week. You will conduct a heuristic evaluation on another team's Figma prototype and vice versa. Another team will evaluate your Figma prototype. This will happen in sections, so it's important to attend section. You will then share the report of what juristic violations you found with the other team within 24 h of section. And you will receive your report and then you as a team, we'll discuss, well, what should we do, what should we redesign based on these usability violations that were found? Briefly write that up, and then submit that to us. So I would say the majority of the work for this assignment happens in section. And then maybe there's another hour or so a team meeting that you should have afterwards to interpret the results. So in the larger scheme of things, this is a much smaller team assignment than the ones you've done previously. If since you submitted your Figma prototypes to us, you've had other ideas, your thinking has evolved. Feel free to iterate and change your Figma prototype. Before this week's section. If you think it's fine, you don't need to do anything.

Following this week, we will have three weeks of weekly implementation check-ins in sections. So you will submit a brief progress updates to us. And then in section, meet with your team to work on your project and also check in with your GSI to discuss your progress, what you've accomplished, what remains to be done, what your plan is. So we want you to every week for the following three weeks basically make for here on progress. So you're not keeping it all for a rush at the end. That takes us through the last week of regular classes. During RRR week. We then have our scheduled demo session, that is Wednesday, May 3rd at 02:00 P.M. That's where we will you will show your final demo to class staff and the public. And you will also have a poster about your project. And then during finals week, one week afterwards, we give you one extra week to put together final video and team evaluation.

19.3 Native App Concepts

- Native applications provide a “cleaner” and simpler architecture for constructing interactive applications bc that SW stack was built for interactivity
- Web apps had interactivity added on top of a document-centric, distributed client-server system
- GUI Toolkits
 - We don't write the main function
- Widgets
 - Modular components that get reused again and again
 - Ex. Android lists or date/time pickers, buttons, toggles, text fields, etc
- UI Components
 - Each component is an object with:
 - * Bounding box
 - * Paint method for drawing itself
 - * Something else
 - 2D graphics model
 - * Typically top left to bottom right

- Sizing
 - * Widgets are not in control, bounding box is controlled by Layout Manager
 - * Component has to know how to draw itself at a size provided by LM
- Working with Widgets
 - Make common case fast, uncommon case possible
 - Common case: assemble standard widgets into a layout
 - * Instantiate class, provide configuration parameters
 - Uncommon case: write your own widget
- Absolute layout
 - Good idea? Provide absolute coordinates for everything
 - * Bad idea in general outside of prototyping, hardcoding parameters only works for a given screen size
- Event Dispatch Loop
 - Mouse moved = Event Queue (sorted queue of input events) → Event Loop (runs in dedicated thread)
 - * Event Loop:
 - Remove next event from queue
 - Determine event type
 - Find proper components
 - Invoke callbacks on components
 - Repeat or wait until event arrives
 - → Component (invoked callback method, update application state, request repaint if needed)
 - Model-View-Controller
 - * Model
 - * View
 - * Controller
 - Receives input events from user, decides what to do by talking to view to determine which objects are being manipulated
 - Calls model methods to make changes on objects
 - Model then notifies views to change
 - Why MVC?
 - * Combining MVC into one class will not scale

20 Thursday, April 6th: Browser

20.1 Midterm Logistics Review

This is the last class before our next midterm – so we will start by reviewing the midterm info again: Midterm #2 is next Tuesday, April 11th. The same format as midterm number one, which means you don't have to come to the classroom. You can if you want, but the midterm is happening online, will release it at 09:00 A.M. you then have 90 min or 90 min times here DSP exam time within a 24 hour window to take the exam. It is on bcourses as a quiz. We once again have a mix of multiple choice and open answer questions. Most importantly, it is not cumulative. Only material that's not been covered in midterm, one will be on the midterm too. So what is that material? Well, it's lectures 14 through 20, information visualization through number 20, that's today, browsers. And the associated readings, which include the two readings, experiment design and interpreting data. The readings on heuristic evaluation, which will lengthen your heuristic evaluation assignment and then how browsers work. Reading that we assigned. You don't have to submit an answer, but it's a part of it.

We had some questions about practice midterms. So there are two practice midterms on bcourses under Files, midterms. Those were both examples of second midterms from CS 160. And just the caveat, that material has slightly changed. So we will not ask you about the specifics of Android. You might see some questions in the practice midterms about that because that is no longer the focus of the class. We will also not ask you about midterm one material. So some of the practice midterms, e.g. have questions about color spaces, which we already covered in midterm. Number one. So take it as a guideline of the types of questions we'll ask, the format of the, of the questions, but we will tailor it to the actual lectures we covered this semester. It will once again be open book, open notes, but you shouldn't be talking to other students in the class about the exam, just like midterm number one question, will there be coding? We might ask you some questions that where we show you some code and ask questions about it. Like we might show you HTML, CSS, and JavaScript, but we will not ask you to write code. Yeah, So just to repeat, we might show you code, but we don't ask you to write code to answer the questions.

20.2 Implementation

Today will probably be a shorter lecture partially because it's such a complex topic – and we should just focus on the the high-level takeaways.

20.3 Where are we in the course?

We talked a lot about the design phase and right now we're in the kind of what's the engineering of user interfaces look like. That's what you're programming assignments were about, that's what your implementation of your team project is about. And then in the lectures that we're currently having, we're looking like one layer beyond that because the code you write, What's the system underneath it that works with that code to actually render interactive interfaces to the user. Okay, so here's the quick review of what we talked about last lecture. So native application concepts,

right? What did we talk about? We talked about key concepts or widgets or controls. Those are the templates for buttons, date pickers, drop-down menus, radio buttons, etc. And how are they architected? Well, they implement drawing, bounds management and event processing. So they know how to draw themselves. They know how to draw themselves when they get resized. And they can process input events. And some of that processing comes kinda out of the box, right? So a button automatically renders itself as depressed when you press the mouse button over it. And then some of the event processing is basically gives the structure for you to add your own code. So your event handlers so that your own code can execute when that button gets pressed by the user. We talked about that widgets are laid out in a containment hierarchy, right? So we have this tree where all the internal nodes of the tree, our layout nodes and all the leaf nodes are the actual widgets. And the layouts determine how the children of the layout note get arranged on screen. And so parents allocate space for children. And so the layouts allocate space and then tell the widgets, Hey, you have the following bounds. Draw yourself. We've talked about most modern UI toolkits support declarative layout of UIs. So basically, this is usually done in some XML dialect. Right here. What happens in native apps looks kind of like what happens in HTML. You have some XML dialect that specifies at compile time what the structure of the Uighurs.

However, we also want this to be flexible. And so usually they're also, you can encode also directly manipulate that, that structure to dynamically change the tree at runtime. Then we also talked about events. And events are how we deliver information about user input. Two components and the user interface. And they are the key diagram we talked about was the event dispatch loop, right? Where you have a Q input events go into that queue. The main event loop in your UI toolkit pops these events off one-by-one, figures out what UI components to deliver them too, and then calls the appropriate event handlers on those UI components and delivers the event to them. Then the last thing we talked about was Model-View-Controller. As a way when you do object oriented programming, how do you write the classes that make up a user interface in a way that is extensible and maintainable. And so the key concepts there's you have model classes that just capture all of the information, right? So in a drawing app that would be describe the shapes that are on screen. How big are they? What are their positions? What are their colors and other attributes? Then we have the view classes, which are responsible for rendering a model onto screen. And then we have the controller classes which take input from an input device and then update the model. Then when the model updates, right, the associated views update. And so that was the cycle of how we go from input to output. But in a structured way that is easily extensible and maintainable. So we can add information to our model without having to touch any of the view code. We can add extra views to our application without touching any of the model code. All right?

Here are a couple of things we already talked about. Event handling, part of the model or controller. So great question. Event handling is not part of the model. It's part of the controller. The controller receives the input, such as I drag on the handle of a shape that's onscreen. The controller may then interpret this as 0. That means I should the user just re-size this shape. So let me find the model class and update its internal width parameter, which in turn then leads to a request to redraw the screen. But event handling happens in the controller. But as a result, we usually then in the controller update the model. Another way to think about this is that if you're assembling your UI from pre-existing widgets and controls, right? Event handling happens in Callback functions on that, on that widget. So that's a control class. Usually you're the internal, how you store information about the application. Those are classes that you create that are not subclasses of widgets and controls, right? Those are model classes. And so event handling happens in the controller, but it often updates a model class as a result. Okay? And then I wanted to go over

these slides again. So we talked about web versus native apps, some pros and cons when you would choose which makes a compatibility on the web is great. You need one code base. We're native apps, it's platform specific, and you might need multiple code bases if you target multiple mobile or laptop operating systems. Software updates and web apps are of course, incident. As soon as you push new code to the server on native apps, that requires reinstallation of the application. But web apps by themselves cannot run offline. Native apps can. However, these days can all the data usually involves some web service. And so even a native app. What good is Netflix? If you can start up the Netflix app, but you can't access any movies. Not that useful. In terms of functionality we talked about that web apps are often kind of one generation behind what is available natively. And that when a new piece of hardware comes, comes around, like a depth camera or another new type of sensor. That usually, there's usually a native application framework first for using making, taking advantage of that hardware. And then performance-wise, usually web apps have lower performance because there's the additional complexity of the browser and we tackle some of that complexity today. Whereas the native applications you can write. Timing critical components in some low-level compile fast language. And then we also did some mapping of concepts onto each other, right? So how do you declaratively define what the UI structure looks like? It looks like in a native app that some XML layout file in the web app. It's HTML. And we'll talk again how today, how that is not a true XML file, even though it looks like one. Visual appearance rules, we use CSS and Web Apps. And there's usually a notion of styles and themes and Android or other native apps. CSS is an especially complex way of implementing visual appearance rules. We'll talk about that a little bit today as well. Application logic is usually written in some native language, right? And UI toolkit in a native application, in a web app gets split across client and server.

We usually have some way of bridging between the application logic and that declarative UI structure. And how do you define different screens? Well, in a native application that is usually some class, whereas in a web app is just you navigate to a new URL. One last piece of review is what happens when we request a web page from a server. So basically this is the quick review of how do we get all the HTML, CSS, and JavaScript to the browser? And then the rest of today's lecture is, once it's in the browser, what do we do with it? So remember that for web apps, we need a web server. The web server has access to files and then our computer, the computer of the user in this case is the client. It sends a request for a particular resource to a web server. So that happens for web apps over HTTP, Hypertext Transfer Protocol Addresses are encoded as URLs, Uniform Resource Locators. The web server receives the request and then maps that through its internal logic to, okay, so what files do I need to retrieve from my file system? What HTML, CSS, JavaScript should I retrieve? It can of course construct these on the fly as well with computation. It then delivers them back to client and the client. The browser then renders the files as a web app user interface. All right, so then today, the topic is once we've received those types of files like HTTP or HTTPS and the browser, what happens next? And so I'm just going to walk through some of the key concepts in the reading that I've assigned for this. Alright, high-level structure of the browser. The browser's main components are the user interface of the browser as well itself. That is, everything that's not the browser window, right? It is the menu bar. For the browser. It is the URL bar where you enter the URL. It's the backwards button, it's the bookmarks, etc. So everything that's not the requested page itself. Then there's a pair of a browser and a rendering engine. The rendering engine is the key most complex component for displaying the requested content. So if you request it an HTML document, it's responsible for parsing the HTML and the CSS and displaying that on screen. We will mostly talk about that today. The browser engine is basically the layer in-between. Like that, uses what you do in the browser user interface to send commands to the rendering engine. There's a networking component, which is basically the

previous three slides, right? How do we actually get the documents from the network for requested URL? And then for all the other documents that might be LinkedIn, it is a JavaScript interpreter, which is usually separate from the rendering engine. So here's a difference between native apps and web apps. And native apps were the part of interpreting the UI and where you put the callbacks, that's part of the same UI toolkit in the browser, we have separate render, rendering engines of JavaScript interpreter's. The browser also then has a UI backend. We're basically saying, Okay, I've received all of this HTML and CSS. When it comes time to draw it on the screen, it's still uses the underlying graphics primitives of the platform it's running on. So just like a UI toolkit in the end will resort to 2D drawing primitives like put text on the screen here, draw a line, draw a rectangle. The browser in the end uses those same calls as well. Then the last component is there's usually a data persistence layer where e.g. all the cookies that you receive gets stored. And HTML5 also has kind of a local storage component where data that's local to the user can be stored in the browser.

All right, So that's the high-level structure of the different components of a browser. And then most of today we're going to spend on an overview of how rendering and JavaScript works. So, okay, question here was, what about the direct arrow from user-interface to UI backend? When does this happen? So think about the browser Chrome, like thinking about the address bar and the back button in the browser, that has to get drawn on screen as well, right? And so there, the user interface component of the browser uses the UI backend to use those low-level drawing primitives to put the buttons and menus on screen. Thanks for the question. All right, let's keep going here. Okay. So high-level overview, what the rendering engine does, it is it gets the HTML and the CSS. It has to parse them to understand their structure. So HTML goes kind of video content tree, CSS goes into a style tree, and then that information has to be fused together to determine for a given element what style should we render it in? That's called constructing the render tree. And then the render tree has all of the components that should be displayed on screen and now has their visual styles. And then there's a layout pass where you determine, well, where, how big is it, where should it go on screen? And then once you've done a pass or that, then painting happens, which is using the UI backend. You just draw stuff into the viewport. On the JavaScript engine, we get the JavaScript, we parse that, we compile it and execute it. Alright, so let's delve into that. Let's start with the rendering engine. Okay? So the rendering engine, that's the piece of code that's responsible for displaying the received content in the browser window. Most of the code and rendering engines focus on HTML CSS, because that is what the width is built on. But rendering engines can also have plug-ins and extensions to handle other formats, right? E.g. most browsers can directly render a PDF. So there are some extension for rendering PDF structured data. But most of the core of the rendering engine is for HTML and CSS. Now, different browsers use different engines. And one thing that's interesting here is that basically what we use today is all, all goes back to open source software. So for those of you who are using Linux, you might have at some point used a browser called Konqueror. And that had a rendering engine called KHTML.KHTML was particularly standards compliant. And then pretty much everything else that the commercial companies use these days somehow goes back to that open source code base. So Apple Safari browser, the rendering engine is called webkit and webkit at some point was forked from KHTML. Google Chrome uses a rendering engine called Blink. Blink at some point was forked from webkit, or more precisely, it's web core component. So also goes back to HTML. Interesting detail is if you use Chrome on iOS, on an iPhone, then it actually doesn't use Blink, it uses webkit because that's an App Store policy in iOS. So their Chrome actually uses different rendering engines based on the platform that you use it on. Microsoft Edge, Microsoft used to use it, right, its own rendering engine. They've now abandoned that and they're also using Blink. And then the only other kind of independent project that has significant traction is Mozilla Firefox. They use gecko and of course,

Firefox also open-source project. Let's look in one level more detail how this rendering engine flow works. So we start by receiving HTML and style sheets. Then we parse the HTML and the output of that is a DOM tree.

So the Document Object Model tree, style sheets, we parsed and we get a tree of style rules. And then there's this merging process where we attach style rules to particular elements in the DOM. And that yields a render tree. The render tree has to go through a layout phase and then goes to painting to draw. Once layout determines the sizes and positions of elements, we painted onto screen, onto the display.

20.4 Parsers

Alright, so let's talk for a minute here about parsers, how that works. So let's maybe get a show of hands.

Who here has taken CS 164 or a version of it – a PL (Programming Languages) course.

Today we are gonna to see the high level intuition of how basically HTML and CSS are custom domain specific languages that for you to program a document structure and visual attributes. And here's like one-six-four in a nutshell of how you parse and then interpret a custom domain specific language. If you want to know more about it than I highly recommend you take that course probably with Sarah Jason's or Zen. Super important can a basic topic in computer science. Okay? So we just received a document which is basically a bunch of characters and we want to understand the meaning of that document. So a common pattern, Which mode, which we use to interpret most programming languages and most domain-specific languages is. We go through a two-step process of first flexing and then parsing. And so lexical analysis, we usually, we turn the document that we've received into a stream of tokens. And so a token could be, this is a string, this is a number. This is a special character. Usually we define what a token is, two regular expressions. So we passed all the characters we received in a document through a set of regular expressions. See which ones match, and then emit a bunch of tokens. In the next step, we then consume the stream of tokens and try to build a hierarchical representation. So a parse tree out of this tokens. The way we use that we do that is usually we define a context-free grammar that defines the structure of legal statements and expressions in our language. The usual tools that you, there's lots of tooling around that. And so traditionally in Unix used to be lex and yacc, those tools are now called flex and bison. So one is a lexer, the other one is a parser. There's also, by Professor at the University of San Francisco, wrote a tool that has a great community called antler, which is a parser generator. And so what you usually do is you provide a definition of for Alexa, you provide a definition for a parser. And then a parser generator will generate the code that can then process a document and give you a parse tree back. So let's look at a really simple example. Let's look at here's a snippet of valid XML. Tag, attribute equals value, some text in between and then end tag. How do we create a parse tree for that example? So here's what a simplified XML lexer and Parser would look like. That could deal with such input. So first you would define a lexer. So here what you see is basically on the left-hand side, you see a definition like I define a token like tags start open. That's the name of my token. And what is it? Well, it's the string Open tag. Tag closes the string close tag. Then we have something like a digit. And now this is a regular expression. It's a digit 0-9.

Letter is between lowercase a to lowercase z or uppercase a through uppercase Z. Whitespace is a tab. New line or a space. I'm an identifier is a letter or some sequence of letters and digits, right? So I define a bunch of regular expressions and give them names. And once I have these in place, right, I can just scan like start scanning the text I'm given and say, Oh, here's a tag, start open. So I admit that I emit that token, does. Next thing that matches what I defined as generic ID is another generic idea. Here's a tag close. Here's something that's not within a tag in XML that's called PC data. Parsed character data, sorry. And so we just turn this document we received into a set of tokens. And then the next thing you do is once I have that set of tokens, I define a grammar. Where the grammar can have I defined rules that match some sequence of tokens and other rules, right? So e.g. two, start, the start tag in XML. Is we have that. Sorry, let me try to get to. This is just this character, right? Generic ID, that is the name of the tag. Then we close the tag, that is this. And then we have zero or more attributes that can be in there. What's an attribute? Well, this is a different rule. An attribute has some name, equals, that's the equal sign. And then it has a value. And then by the end, so we define these rules. And then we say an element in XML has a start tag. Inside of it. It can either have other elements or just some texts. Zero more of those. And then we have an end tag. So now we've defined a grammar over these tokens. And what I can now then do is construct a tree based on matching grammar rules, right? So we have a top-end element that is this whole string. Within that. We have start tag that goes from here to here. We have some text inside of it that is from here to here. We have N tag that goes from here to here. And then within the star tag, this matches the name of the tag itself. And then we have an attribute node, which has an ID, which is this part, and a value which is this part. Of course, inside of this tag, we might have other tags in here, right? And then the tree would just get more complex. So through this two-state stage process, we started off with just some character data that we received in the browser. And then we now turned it into a tree representation of that domain specific language. Okay? So unfortunately that is, it would be great if we, we're done here. That's the basic approach of how you parse CSS and JavaScript. For both CSS and JavaScript. We can define a set of lexer rules. We can define a context-free grammar and we can parse it in that form. Unfortunately, that does not work for HTML. And other question is, why not? HTML looks like XML, right? Shouldn't just be an XML dialect. And unfortunately, HTML takes some of the rules of XML in a couple of ways. So examples of valid HTML that are not valid XML, an image tag, the image source URL here. Why is this not valid XML? Because it doesn't add any point have a closing tag. Same thing for a new line, a line break. What's missing? We don't have a closing tag. Another example, you can write input type equals checkbox. But based on the rules that we defined, every attribute, the value has to be in quotes. So there's a bunch of ways in which you can write HTML that's valid HTML. But it's not valid XML. As a result, you cannot use standard context-free grammar based parser. It can deal with these edge cases. The other big reason is lots of people write HTML by hand and they make a bunch of mistakes. And browsers want to be able to render HTML even in the presence of errors. They don't just want to throw up an error set malformed HTML. I'm not going to show you anything. Let's talk about that for a moment. Okay, here's some HTML. Look at it and tell me in the chat what is wrong with the following snippet of HTML? There are multiple errors in here. Look at it, think about it, and then tell me in the chat what you think is wrong. Alright? Says nested tags. In particular, what's, what's wrong with the nesting? Yeah, okay, farm Rahul and rationality all have the right idea, right? As these two go together. And these two go together. This breaks the nesting relationship. The P has to be closed within the div to be valid, right? But people do this, people write stuff like this. So the p, This would need to move down here. Okay. Anything else you see that is wrong, right? My tag that doesn't exist, that is not a valid HTML tag, not part of the HTML spec. So what should a browser do when it receives such HTML? To just throw basically a compile-time error was like, I'm

not going to render this document. In general, browsers say no, that's not what we should do. We should try to fix the errors that are here and just do a best effort at rendering.

Okay? So what happens is HTML parsers as implemented in browsers, they still have a tokenization and a tree construction phase. But we don't just use the standard lexer and a parser. There's like more complicated code that runs that e.g. checks for all kinds of custom error conditions and then corrects the HTML's ago, we think maybe you meant this. So we're not gonna go into the details of how that happens, but you can look it up into HTML spec on this link, which suggests the structure of such a parser. So high-level structure of the parts are still the same, but at a low-level would actually happens under the hood is way more complex. Okay. Now we get to parceling CSS. Css is much more well-behaved. Css. The language definition for CSS is really clean and can be parsed with the standard lexer and Parser. So we take something like the character stream up here and then we produce a tree of CSS. Alright, so we've passed our HTML with a much more complex parser, constructed a DOM. We parsed CSS and we now have our CSS tree. What do we need to do next? Well, what we want to end up with is a render tree. So a render tree has the visual elements in the order they will be displayed on the page. And each node or render knows how to layout itself and its children. So that's a render tree, kind of like containment hierarchy, right? Each node knows how to draw itself. The renderers and the render tree kind of correspond to DOM elements, but the relationship is not one-to-one. E.g. the head element in HTML, anything that's in the head doesn't get rendered in the viewport. Elements with a CSS style of display, none are not rendered. On the other hand, we have complex elements like a drop-down on a form that might need multiple renderers. So we end up with some tree here that has a relationship to the DOM, but it's not necessarily a one-to-one relationship. The next important thing we need to do is we need to match which visual styles that are defined in CSS and HTML we should attach to each of the nodes in our render tree. Visual styles. There's a bunch of different ways in which you can define visual styles. So you can author CSS rules. Css document. You can also in the HTML L itself, have CSS style attributes. In addition, HTML has visual attributes, like setting the background color on a paragraph, which is not CSS syntax, but also defines a visual style. We have three different sources of where style rules come from. And then for the CSS rules, they actually can be many different style sheets that a single UI loads it. So in particular, there are not just the page style sheets. These are the ones that as an application developer, you define. But the browser also has a default style sheets. So if there are no CSS rules, what font you use, how big should it be? That's a browser default style sheet. And then each user profile in the browser can also have style sheets where you just set your preferences. I want everything rendered at 150 per cent. I prefer the following background. And then each style sheet can have many rules. Now there's a question of, well, how do we look up which rule to apply when this is where the cascade order comes in, which is the C in CSS, right there. Not just style sheets, but they are cascading style sheets. So what does the cascade order mean? You can look it up in the specification for CSS. It basically says there is an important, there's an order of importance. So in ascending order, the least important rules are user agent that those are the browser rules. Browser user normal declarations, these are the user's preferences. Author declarations are the author of the web app, right? So this is web app. And then there is a level of importance. So if you have a exclamation mark important rule, then it kinda supersedes others. So you first look up, is there a user important declaration? If not, let's go through the Web App declarations. Then let's go through the user's normal preferences and then the browser preferences. And basically the higher, the higher ones take precedence over the low ones. So that's what we mean by the cascade order in cascading style sheets. And then the other part that happens is we have a tree, right? We have a tree of HTML. Elements are dumb. Next thing that happens, let's say we're at this element down here. What do we do? We look up the

cascade. So these were the rules I just talked about. Can we find a value for background or text color? If so, we use it. If not, we go up the tree and we check, did this node define background, color and font? If so, we use it from here. If not, we go yet further up the tree. So the upshot of all of that is actually matching styles two nodes is not a trivial process at all. You need to consider multiple sources and different preferences, which makes the whole thing kind of more complex than if you just have a style parameter in a native UI. Alright? But at the end of this render tree construction, we basically have everything that gets rendered on onscreen and we have the right CSS style attributes attached to each of the nodes. The final thing we have to do is now we have the styles, but we don't yet have the sizes. And so the final path that happens is a layout stage, where we basically go from Herron down the tree to children. We determine the width of an element that could come e.g. from a CSS style rule. Then we recurse on the children, call land on the children if needed. Maybe calculate their height and then from all of those heights, we calculate our own heights. So it's kind of a panel down, bubble up algorithm where based on available width, the calculate heights, and set those for the whole tree. And then we have basically the bounds for each of the boxes. And once we have those box bounds, we can just call paint. And then paint. It now has all of the absolute coordinates that it should use and abuse this drawing primitives like put texts at x location x, y with the following color and the following font. But this is now this uses the same underlying primitives as UI toolkit in its draw method. So we have access to 2D drawing primitives, lines, boxes, texts, colors. So one way to look at it is that this whole rendering engine, I'm in the browser does the work of the UI toolkit. It just does it in a kind of a more complex way because we have to deal with the vagaries of HTML. We have to be very error permissive. Whereas a UI toolkit, if you give it a malformed XML, let's say error can't deal with this. And on the other hand, we have this really flexible Cascading Style System, which is also more complex than style definitions in native UI Toolkit. Okay, then just briefly, so we just covered what is happening up here. And then just briefly wanted to touch on what's happening down here, but not in a lot of detail. Which is we also get a bunch of JavaScript from the web server, right? So what do we do with that? So the first part is we parse it. So a JavaScript parser. This here is kind of what we just talked about in the parsing S aspects. So we'll have a lexer and a parser. And then in the end, we end up with an abstract syntax tree, which is a tree presentation of the code we just received. Now, JavaScript is an interpreted language. But interpreting all of the JavaScript that we received would be too slow. That's what web browsers did in the very early days of the web. This is not what we do any anymore. Instead, we compile JavaScript using a just-in-time or a JIT compiler into machine code. And then. Execute that to speed it up. And in fact, just this box of how do we make JavaScript faster has been endless engineering on that part. And what's actually happening in browsers these days under the hood is not just we don't interpret, but it's more complex than that. So here e.g. is an example of what the V8 JavaScript engine does in Chrome. And it basically has two pathways. He has a compiler that this is the fast pass. So we get our source code, we compile it into bytecode. We have a bytecode interpreter and executed. And while this execution is happening, if we have spare cycles, will say, Well, we tried to go through this path as fast as possible. But as a result, this is not highly optimized. It's way better than interpreting it, but it's still kinda slow. So that's in the background. Launch another path. And try to optimize this bytecode that we received with an optimizing compiler. And once that is ready, replace this binary with an optimized binary and start executing that. Kinda way more complex. Again, then what's happening in a UI toolkit, right? We're, we're just, we're compiling the code that you gave us and we're running it at runtime. Based on the design decisions that were made early on is that we attach this JavaScript. We're just kind of by accident, this interpreted language took over. And now we have to deal with the fact that we have this kind of not very optimal language. But we really want fast application performance. And so we need way more machinery on the backend to

get something like native performance out of it. Okay, so there are tons of details like if you want to become a browser engineer. So Irina asks, I'm curious why not just do the optimization first, then render the optimized code. Because this path down here is fast to compute. Slow to execute. This path up here is slow to compute and then passed to execute.

So the desire is that like your web browser is loading and this JavaScript code, we want to start executing it as quickly as possible, right? So from the time, from the time that we receive this document of JavaScript, we want to start executing that code right away. So you want to do the fast just-in-time compilation and start executing it. Then in the background, say, well, yeah, but now this execution, this bytecode we got, we can probably do better. But it'll take us a couple of hundred milliseconds to maybe optimize it. So we optimize it behind the scenes and then replace it. And yes, either could be faster as the final result. You don't know, right? Because you're just getting this JavaScript code stream then in synchronously in real time. Great question. Okay. That brings us to That's kind of the level at which I want. However this, so if you want to become a browser engineer, that's like being a highly skilled software engineer in all the vagaries of implementing either the rendering engine or the JavaScript engine. I think those are very highly sought after skills, but only a couple of people focused on that. But hopefully, even as a application developer, even if you're never touching the source code of the rendering engine with the JavaScript engine, it's good to have the conceptual overview of what happens underneath the hood and an understanding of it.

21 Thursday, April 13th: Collaboration and Social Computing

21.1 Post Midterm 2 Feedback Form

Please fill out the form pinned on EdStem if you have anything to share!

21.2 Remote Communication

21.2.1 Zoom

We all are familiar with online meetings over Zoom.

21.2.2 Collaboration Apps

- A lot of big companies fell behind because they did not allow for collaboration (ex. Microsoft Word falling behind Google Docs)
 - The companies have tried to make up for it (Word now allows collaboration) but it hasn't been enough – the software wasn't **designed** for collaboration *from the start*.
 - This was an easy way to make a lot of money in the early days of (EE)CS – make a new product which is just an existing product but with collaboration

21.2.3 Calendars

Who here uses a Google Calendar (or some equivalent) to manage your schedule?

As expected, almost everyone in the room raised their hand.

The ability to anonymize what you are doing (and thus not have to worry about people scheduling events they think are ‘more important’ over your existing events) was a key factor in its popularity.

21.3 Challenges to social software

21.3.1 Success crisis

Whether your software scales will only become apparent once you are successful, and then it's often too late to change the architecture (see Friendster)

This is somewhat reminiscent of “Design security in from the start” from CS 161.

21.4 Single Display Groupware

21.4.1 Roomware

Example: AT&T Network Center Control Room – has everyone in the same room at the same time, even if they have different responsibilities. This helps develop a shared sense of purpose for the end-goal and allows people to be up to the date wth the overall project status.

21.5 Graph Theory

- # of edges in a fully connected graph: $n * (n - 1)/2 \sim n^2$
- # of edges in a tree: $n - 1$

Coasean Floor: The point below which the transaction costs of a particular type of activity are too high for a standard institution to pursue. - Clay Shirky

21.6 Power Law Distributions

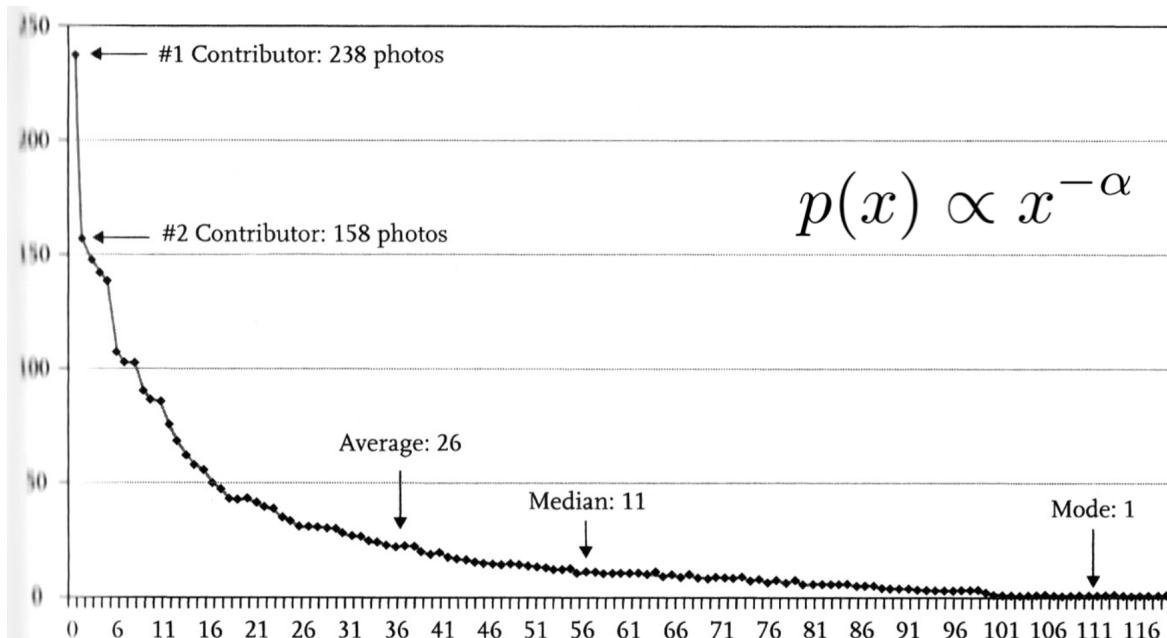


Figure 5-1: The distribution of photographers contributing photos of the 2005 Coney Island Mermaid Parade.

From C. Shirky, *Here Comes Everybody*

$$p(x) \propto x^{-\alpha}$$

21.7 Time/Space Matrix

	Same time (synchronous)	Different time (asynchronous)
Same place (co-located)	Face-to-face interactions	Continuous Task
Different place (remote)	Remote interactions	Communication + Coordination

21.7.1 There goes the neighborhood:

Early adopters are often a self-selected, homogenous group; therefore utility in early stages is not indicative of the “steady state” once successful.

When new people use your app, they may say ‘there goes the neighborhood’ to express how they feel new users will be detrimental to the app, due to them using the app in unintended ways.

22 Tuesday, April 18th: Accessibility

22.1 Accessibility Goals

We should strive for design which is as inclusive as possible.

Taking a quote from physical furniture design:

There is no such thing as an average person – everyone has some different kind of arm length, leg length, etc.

Instead we strive to include as many *percentiles* as possible: we should be able to accommodate someone from the 5th %ile as well as someone from the 95th %ile and everyone in-between.

22.2 Disabilities

Disability is not a ‘personal health condition’ but a *mismatched human interaction* – it is something we can and should design to accommodate.

22.2.1 Types of Disabilities

	Permanent	Temporary	Situational
Touch			
			
	One arm	Arm injury	New parent
See			
			
	Blind	Cataract	Distracted driver
Hear			
			
	Deaf	Ear infection	Bartender
Speak			
			
	Non-verbal	Laryngitis	Heavy accent

Note that there are different types of Disabilities which have different extents.

Making sure audio can be heard for people who just are hard of hearing due to an ear infection and not only accommodating for people who are completely deaf is a necessity for accessible design.

22.3 Solve for One, Extend to Many

Curb Cuts (when the elevation in sidewalks goes down to meet the road) are helpful for people in wheelchairs, but are also helpful for new parents who have a baby in a stroller.

22.3.1 Closed Captioning Usages

- Started for deaf people
- Good for listening to accents
- Allows you to watch videos in public spaces where it's too loud to listen to audio
- Helpful for new language learners who do not know word pronunciations but know of the word in its spelling

22.4 Specialized Hardware

This can help with accessibility: Braille Display is an example of this (bumps on surfaces for blind people to feel what the display is saying).

22.5 Visual: Screen Readers

How do we make sure to not include the footer when reading a webpage out loud?
How do we ‘read’ out loud images?

We can set the `alt` attribute of `` tags to set what the image reads out loud. Note that this extends to give an alt text in case the image cannot load.

About 8 percent of men (one in 12) and 0.5 per cent of women (one in 200) have some form of red-green colour deficiency.

22.5.1 Do not cause Seizures!

Do not flash anything more than three times in a row.

22.6 Navigable

You can set the `tabindex` of `<div>` elements to set the sequential ordering that you navigate in when pressing the Tab key on your keyboard.

23 Thursday, April 20th: Future Interfaces – VR/AR

23.1 Virtual and Augmented Reality

VR has been around since the 1950s with Heilig's Sensorama.

23.1.1 Changes to costs of VR Hardware

However VR Arcade Games in the 1990s costed tens of thousands of dollars – but improved technology are much cheaper now (in the hundreds of dollars or less).

! If the costs are not low enough, some usages of technology will not be discovered. This is yet another reason why making sure hardware is accessible is important.

The most obvious example of the above statement is if you want to be able to draw with brushstrokes in 3D. Would you pay \$20,000 for that? Now what if it is $\approx \$0$, then would you consider using it?

23.2 Augmented Reality

Definition:

AR is a computer technology that augments a physical, real-world environment directly or through its indirect view with computer-generated information, including graphics, video, and sound.

AR may alter a user's view of reality, and may also enhance one's perception of reality.

Note that just like VR, there was a wave of AR in the '90s. The thought was that it'd see usage in car repairs/maintenance among other applications.

Microsoft Hololens2 (2019) showcases this amongst other applications (primarily in games).

23.2.1 Video see-through AR

Pokemon Go is the most well-known example of this. *But* it did not really try to understand the scene – it just placed a Pokemon somewhere even if it's not a realistic horizontal surface or sensible with the current lighting.

23.2.2 HMD-based AR

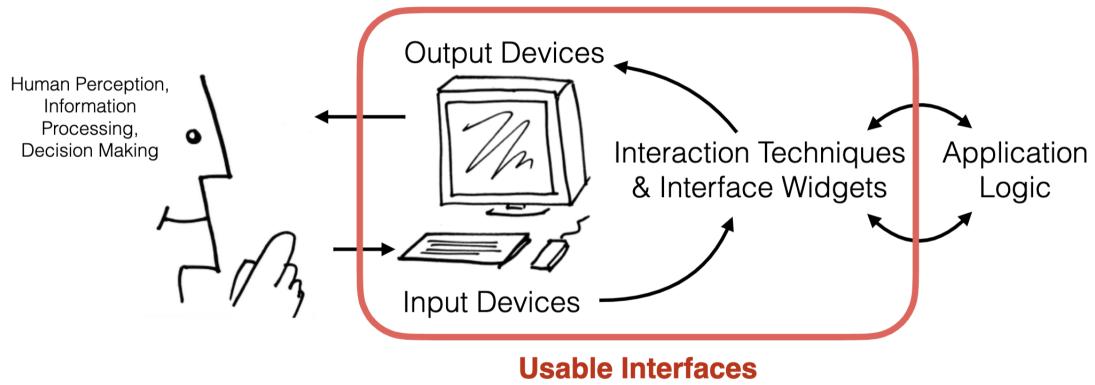
Head Mounted Display is when you have a virtual display mounted on your head. Sometimes this is paired with actual hardware, which the headset then draws on top off (in real-time), to make a more immersive experience.

23.2.3 Hardware Complexities

There are lots of sensors and hardware parts, especially in HMDs.

23.3 Enabling Technologies/ Open Research

- Near-Eye Displays and Optics
- 3D Localization
- 3D Content Capturing
- New Human-Computer Interfaces



23.4 Getting Input

There are 3 important inputs to a VR/AR application: **Head, Hands and Limbs**.

23.4.1 3D Glasses

Red-Blue 3D glasses use light subtraction to make images seem as they come out to you – but in the process you lose many red and blue colors and get a grayscaling effect. This is known as an Anaglyph (spectral) approach.

However there is also the “Polarization multiplexing” approach which is much like polarizing (anti-glare) sunglasses, but with a polarizing filter on each projector.

23.4.2 Active Shutter

These were also glasses you could wear to see 3D video from your TV – these were popular 10-14 years ago.

23.5 Head-Worn Displays

These should allow seeing-through so that you can be aware of your environment.

23.6 Audio Displays

23.6.1 Auditory Cues

- Localization: psychoacoustic process of determining direction and location from which sound emanates
- Binaural cues:
 - ITD interaural time difference
 - IIT - interaural intensity difference
 - HRTF - head related transfer function (physical shape of ears, ear canals)
 - Reverberation
 - Intensity

$$I \propto \frac{1}{r^2}$$

where I is the sound pressure level, and r is distance.

HRTF measurement: For far objects, radius less important. Function of sound frequency and position: azimuth, elevation, radius.

23.7 Reverberation

Early reflection: localization

Long tail late reflection: room characterization

23.8 Haptic Displays

- Simulate the physical interaction between the user and objects
- Encompasses sensations of force, touch, vibration, temperature

Core challenge: cues for particular categories are very different from other categories.

23.9 Gustation and Olfaction Displays

Can we simulate smell and taste?

Current answer: No.

Sound: Air pressure variations over time (which is 1-dimensional). Vision: Red-Green-Blue pixels (which are 3-dimensional).

But smells are a much more complicated dimensionality and carrying a backpack with thousands of smells in jars is not really feasible.

With taste, we have a subset spanned by descriptions (such as salty, sour, sweet, etc) but this does not generate a basis.

23.10 Professor Hartmann's PhD students' work: Pointing & Selection in 3D

How do we select stuff that is further away:

- Shrink stuff down
- Cast a ray into the scene

There has been success in making arms that reach out beyond an actual human's reach – much like Inspector Gadget's mechanical arms.

Remember that since we are doing a *simulation*, we can alter the laws of physics that are written in the code.

Results: you can control multiple bodies at the same time – with a natural feeling, despite it being an action that can never happen in reality. Cognitive Science has yet to give an answer to why that is.

23.11 Bi-manual Scaling

Pinch-to-zoom but with 2 controllers – allows us to scale the world up or down.

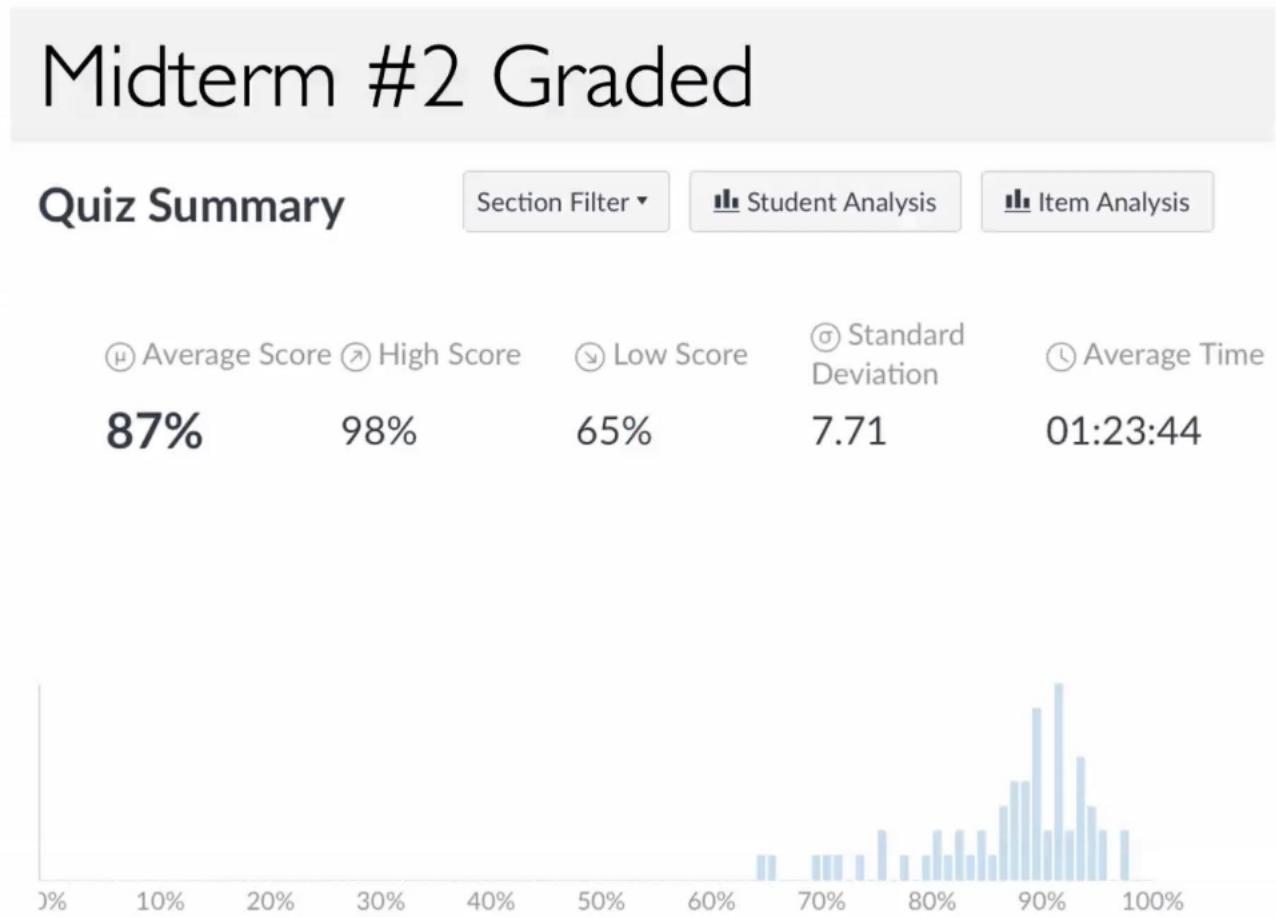
24 Tuesday, April 25th: Career Panel

24.1 Speaker List

We have Speakers from:

- Heidi Dong ([LinkedIn](#)), Software Engineer, Frontend at The New York Times
- Janaki Vivrekar ([Personal](#), [LinkedIn](#)), Software Engineer at Amplitude
- Sonia Uppal ([LinkedIn](#)), Software Engineer at Meta
- Emily Pedersen ([Personal](#), [LinkedIn](#)), SW Engineering Program Manager at Apple

24.2 Midterm #2 Graded



24.2.1 Regrades

Same as midterm 1.

24.2.2 Course Standing

If you have any concerns about your standing in the course and your ability to graduate, please get in touch with us proactively via private message on Ed Discussion.

24.3 End of the Semester

Course summary and next steps on April 27

Section: Submit update including demo video this week **before section!**

24.4 Design Showcase

Design Showcase runs Wed-Fri of RRR week every semester. The Jacobs Institute will promote all courses and projects to the public.

Our presentation slot:

Design Showcase Session C:

Wednesday, 5/3 - 2-3:30pm / Jacobs Hall Room 310

Poster: Due Monday 5/1 11:59am (noon) if you want us to print it!

24.5 Poster Details

See [bcourses](#).

24.6 Final Demo Details

See [bcourses](#).

24.7 Q&A

How has ChatGPT changed the way you worked?

Most companies do not allow you to use ChatGPT for obvious reasons but on the side it has been cool to play around with.

What courses were most useful?

I work in TypeScript which was not taught in any class at Cal but look into taking DeCals and others, see:

- <https://www.figma.com/proto/LIWnxQaKzQkYD5k5wpP8oA/classes-i've-taken>
- <https://vivrekar.medium.com/discovering-human-computer-interaction-at-uc-berkeley-9fc211145638>

25 Thursday, April 27th: Last Class

25.1 Final Lecture... but wait there's more

CS 160 has their Jacobs Spring Design Showcase at Studio 310 on Session C (2-3:30 PM).

Note that this is the same classroom where regular lecture is held but the date is different: Wednesday May 4th.

25.2 Final Deliverables

What should you submit on **May 10th**?

- **Writeup:** Now that you have a working app, get feedback from **3** users, write up what you learned and would change.
- **Video:** Record a demo video of the final application.
- **Source Code:** Remind us of the link to your repo, make sure you have a `README.txt`.

25.3 Takeaways

25.3.1 Why UI is Important

Major part of work for “real” programs

- Approximately 50%

You will work on “real” software

- Intended for people other than yourself

Bad user interfaces cost

- Money ($5\% \uparrow$ satisfaction \implies up to $85\% \uparrow$ profits)
- Lives

User interfaces hard to get right

- People are unpredictable

So what should you do then?

Answer: **Iterative Design**

Finally, don’t use intuition for design – instead observe the user in context.

Also separate the designer’s conceptual model from the users’ (which is from experience & usage).

25.4 Future Coursework for Undergraduates

BCDI: Berkeley Certificate in Design Innovation

- Interdisciplinary certificate (like minor)
- 4 courses, CS160 counts as one of them – you are 25% done!
- Look at DES INV courses (they all count)
- <https://bcdi.berkeley.edu>

Decals and student Orgs:

- Web Design Decal <https://wdd.io>
- InnoD Decals <https://www.innovatedesign.club>
- Berkeley Innovation: <https://www.berkeleyinnovation.org>
- Extended Reality @ Berkeley: <https://xr.berkeley.edu>

25.4.1 Graduate School

Thinking about graduate school? Check out

- Berkeley MDes – Master in Design
- Berkeley MEng – Visual Computing
- Berkeley MIMS
- Berkeley PhD in CS; iSchool

Other Universities with top HCI programs: CMU, U Dub, Stanford, GTech, MIT, UMD, etc.

25.4.2 What if I am a Graduate student?

Relevant Graduate Courses at Berkeley

- CS294-137 Immersive Computing (Yang, Hartmann)¹
- CS294-184 Building User-Centered Programming Tools (Chasins)²
- NW MEDIA 203 Critical Making (Paulos)
- CS260B Topics in HCI Research (Paulos, Hartmann)
- INFO 214: User Experience Research (Fadden)
- INFO 247: Information visualization and presentation (Hearst)
- INFO C262: Tangible User Interfaces (Ryokai)
- INFO C265: Interface Aesthetics (Ryokai)
- INFO 290: Human-Centered AI (Salehi)
- and many more!

! INFO 213 is similar to CS160 – only take one

¹only ugrads allowed will be those who have taken CS 160 – and by petition

²only for PhDs

25.5 Interested in HCI Research?

If you did very well in the CS160 midterms and individual projects and want to get involved as an undergraduate research assistant, send a message on Ed Discussion and/or approach the following HCI faculty about research opportunities:

Profs: Eric Paulos, Sarah Chasins, Aditya Parameswaran, Marti Hearst, Niloufar Salehi, Kimiko Ryokai (EECS & ISchool).

Summer has been mostly planned but there are openings for Fall 2024.

25.6 Q&A

Q: What is the best way to get into SWE as an M.Eng.

A: Take CS 169

25.7 Course Evaluations

Please fill out <https://course-evaluations.berkeley.edu/> so that the class can improve!