Last time

Visualizations can be represented as **encodings** that map from data to marks & visual attributes based on data types.

Our **cognitive and perceptual systems** determine which **encodings are effective**: we (mis)read data if encoded poorly.

Active research at frontiers investigating **how users can create effective visualizations and how readers take information away from them**.
cognitive models
visualization
(and don’t forget the design cognition that we already covered)
Software

Unit 6

Software, Tools and Toolkits
Content Creation
Today

Threshold and ceiling
Changing problem representations
Learning programming
Software engineering is a highly complex task, a microcosm of many challenges in HCI.

Making software engineering more accessible could empower millions to customize applications and write programs.
Programming ain’t easy

Developers struggle to recover others’ implicit knowledge by inspecting code [LaToza, Venolia and DeLine 2006; Ko, DeLine and Venolia 2007; Ko et al. 2006]

Developers rarely hold all information needed for the task, and often must turn to the web [Brandt et al. 2009]

Just-in-time learning of new skills, clarifying existing skills

Reminding themselves of details

Barriers span from conceptual (how is this even possible to code?) to pragmatic (how do I express this?) [Ko, Myers, and Aung 2004]
How do we aid programming?
Threshold and Ceiling
What is your programming intervention actually doing?

What is Github Copilot’s design goal? How do we know if it’s succeeding at that design goal?

Are some programming languages “better” than others? How would we know?

Is the VSCode plugin helping? With what?
Threshold/Ceiling Diagram

Threshold: Difficulty to use (semantic distance, often in gulf of execution — sometimes in gulf of evaluation)

Ceiling: Sophistication of what can be created (higher expressivity)

Are you trying to lower the threshold, or raise the ceiling?

- C++
- Node, Python
- HTML, CSS
- Figma
Lowering the threshold

**Goal:** reduce the effort and cognitive complexity of creating software artifacts
How to lower thresholds

One approach is to reduce the ceiling (expressivity) in exchange for smaller semantic distances in gulf of execution or evaluation.

Regular expressions are simpler to understand than context-free grammars, but also less expressive.

No-code or low-code front-end web frameworks can be fast to get off the ground but limited in what you can create.

Python manages memory and garbage collection for you, but also trades off some manual optimizability of memory to achieve it.

But, not all lowered thresholds require lower ceilings — we saw last time how representations shape cognition (e.g. number scrabble).
Asking ‘why’ questions of code

[Ko and Myers CHI ’04, ICSE ’09]

Debugging problems often reduce to “why” questions, but these questions are challenging to answer (=high threshold)

Analyze program traces to answer many unanswered “why” and “why not” questions about what just happened
Data science notebooks

Automatic cleanup of Jupyter notebooks by tracking provenance across cells

[Head et al. 2019]
Programming by demonstration (PBD): teach a computer a program by doing it yourself while it watches

Challenges

There is an infinite, and hugely branching, space of programs that might be inferred

Inferred macros can be extremely brittle
PBD on the desktop

[Cypher 1991]

Infer a macro by watching the user’s behavior
Modern PBD: Excel flash fill

[Gulwani 2011]
Develop a domain-specific language of string transformations, and learn from examples how to decompose it into subproblems.

Machine learning ranks between all possible valid programs.
Raising the ceiling

Goal: increase expressivity (range and sometime complexity of what can be created)
How to increase the ceiling

Identify opportunities for **untapped expressivity** in the current language, and position the software to expose that level of expressivity.

This is not about “adding knobs”: it’s about (metaphorically) providing new paint colors in the palette.
Non-programming examples

Engelbart’s chorded keyset
[Engelbart 1968]

Musical instruments: the goal isn’t to reduce the threshold to playing the piano — it’s to enable high musical expressivity
Programmable artist brushes

[Jacobs et al. 2018]

Attaching computational functions to brushes enables new forms of artistic expression.

Dynamic Brushes is a programming and drawing environment for creating procedural drawing tools.
Programming as problem representation
Domain-specific languages

DSLs, or domain-specific languages, are programming languages that are tailored to a specific domain

- SQL (databases)
- d3 / Vega Lite (visualization)
- pytorch, keras, tensor flow (machine learning)

Successful DSLs reshape the cognitive representation of the task, reducing the gulfs of execution and evaluation and empowering development in their application domain.
Data science representations

I have too much data to fit in my computer. How do I count the number of times the word “HCI” appears on the web?

**Representation:** *Map-Reduce* [Dean and Ghemawat 2008]

First, run a *Map* phase that runs a simple function over each webpage. That function outputs the number of HCIs, and can be run completely in parallel across every page on the web.

Second, run a *Reduce* phase that collects the outputs from the Map phase and aggregates them: here, via a sum.
Representations for vis

[Bertin 1983; Mackinlay 1986; Satyanarayanan 2016]

How do we tell a machine to create this? Paint pixels?

It’s extremely challenging until we adopt a representation that visualizations are encodings of data types into marks.

```r
vl.markPoint()
  .data(data2000)
  .encode(
    vl.x().fieldQ('fertility'),
    vl.y().fieldQ('life_expect'),
    vl.size().fieldQ('pop').scale({range: [0, 1000]}),
    vl.color().fieldN('cluster')
  )
  .render()
```
Learning programming
Constructionist learning:
learning happens most effectively when people are making tangible objects

Lego Mindstorms followed this mold and was named after it
Scratch
[Resnick et al. 2009]

Inherited from Logo:
Block-based programming of simple animations and games as a gateway to programming for children
Online python tutor

[Guo 2013]

Embeddable Python data structure visualization

Over 200,000 users and a dozen universities using it

```python
1 def listSum(numbers):
  2     if not numbers:
  3         return 0
  4     else:
  5         (f, rest) = numbers
  6         return f + listSum(rest)
  7
8 myList = (1, (2, (3, None)))
9 total = listSum(myList)
```
Codeopticon

Watch many learners code and debug in real time.
Successful programming tools shift our cognitive problem representations to make the task more readily solvable.

Tools for learning programming help externalize our cognition to better understand what code is doing (or ought to be doing).
References


References


