The End

15.SECT.goodbye

- What makes public interest tech unique?
- How do we find our own philosophy?
- How and when to stay current.
- Management
- Lessons from one guy’s career

Course Objectives

- Develop fluency in common software architectures and patterns, and a sense of when they are and aren't appropriate to use.
- Gain practical experience working on a team, building a project with enough complexity as to help demonstrate the benefits of the practices discussed in class.
- Begin to develop "taste" when it comes to software design & interface choices. What makes an interface or API "nice to use" and how can you ensure others feel that way about code you write?
- Understand some special considerations that come with working in civic tech/public interest tech, from audience considerations to working within certain common constraints.

Key Lessons

- Prioritize communities, then users, then everything else.
- Small teams need autonomy mixed with accountability.
- Writing software is a form of communication: of theory, of values.
- There are infinite ways to solve a problem: your job is to find one that is clear, reliable, and efficient.

  (Usually in that order.)

Technology in the Public Interest

- Usually means fewer people, reduced resources, tougher requirements.
- A team of 4 can outperform a team of 400 if they are building the *right thing*.
  - *Not unique to civic tech: startups often outmaneuver giants in niche markets.*
- Leverage benefits of small teams:
  - Agility
  - Trust
  - Unified Vision

Staying Current

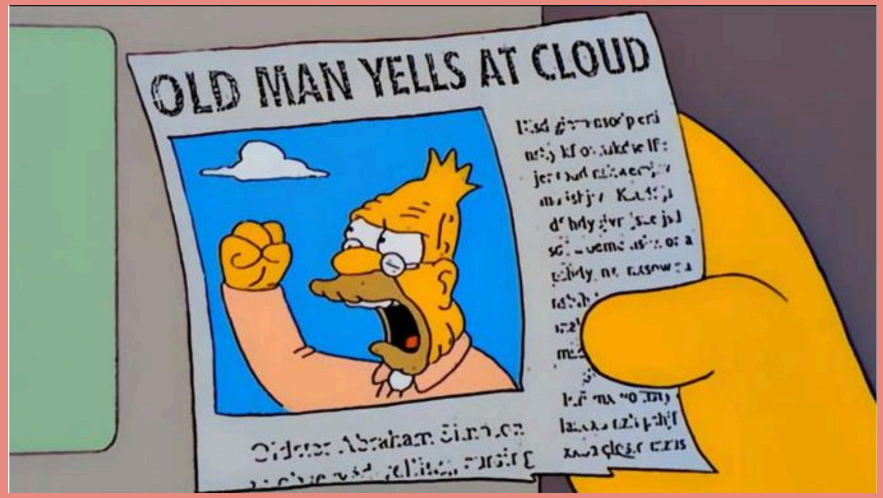
- blogs/newsletters
- federated social media
- aggregators (https://lobste.rs)
- side projects/personal itch

Read & engage with people who do work that you enjoy. That's part of why they work with the garage door up.

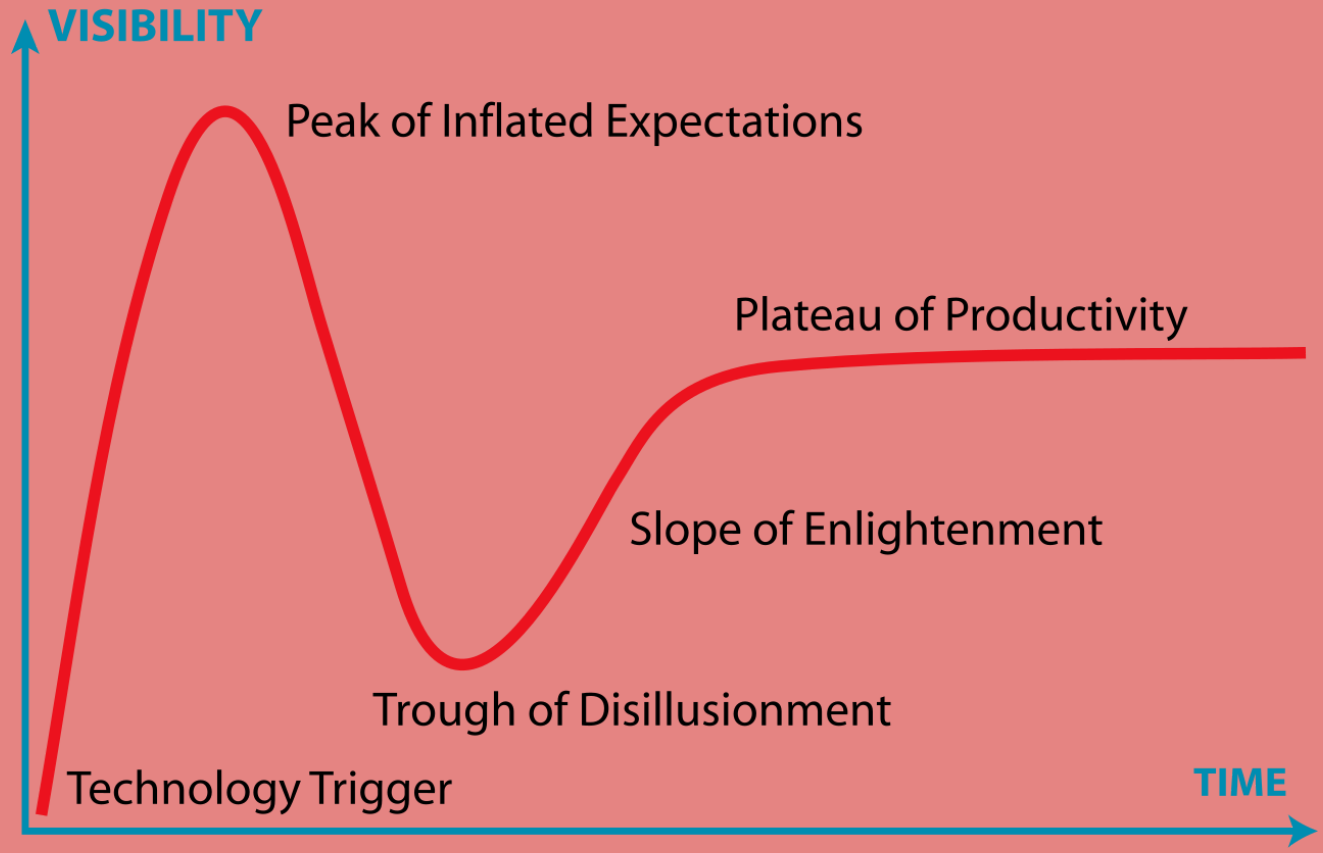
How Current?

The "old ways" will continue to work, newer isn't always better.

There are benefits, especially in public interest environments, to not being on the *cutting edge*.



Gartner Hype Cycle



https://en.wikipedia.org/wiki/Gartner_hype_cycle

Hype Cycles

1990s

- OOP design patterns
- "Java on every device"
- everything-is-XML

2000s

- AJAX
- semantic web (RDF/XML)
- service-oriented architecture

2010s

- NoSQL databases
- "big data"
- blockchain
- microservices
- containerization (Docker)

2020s

- return of strong types (TypeScript, Rust)
- edge computing
- Generative AI

Depth vs. Breadth

A personal choice, but early in career I'd favor *breadth*.

Stay curious.

Tip: When you encounter something for the Nth time, read more about it.

20% learning, 80% doing

Lots of ways to get there:

- 1 day a week
- a new job every ~4 years
- *make things*
- *experiment at the edges, stability in the core*

LLM Usage: CAPP Alums (2016-2024)

~15 participants: half from pre-2023

- Two reported next-to-no usage. (one from 2023, one from ~2020)
- Two reported near 100% use (one from 2024, one from ~2020)

Not all happy with usage, but belief/acceptance it is a reality of the job, likely here to stay in some form.

Points of Agreement

- Unanimous–First year CAPP CS curriculum *should not remove things*.
  - Important skills remain the same.
  - Possibly more emphasis on reading code, architecture.
- Useful for tedious parts of the job.
  - Config, tests (but be sure to read them!), HTML/CSS utilities
- Architecture/interface design important, ability to implement less-so.
- Many seeing juniors struggling with basic tasks, even with LLM assistance.
  - Several vibe-coder colleague horror stories. "An entire app in a language he didn't know"
- Concerned about keeping skills sharp/regression to mean in tech choices. Mentorship. Job losses.

Divided on usage for learning.

- More willing to try new things.
- Less need for peers' time
- Less interaction with peers
- *Overconfidence* of LLM, convincing people they were incorrect when it was wrong.

Other Bits

- *My boss thinks it can replace me, I had to justify my continued employment.*
- *It is overused and I want to leave my job but don't know where it isn't.*
- *After months of informal usage, adopted a policy last month effectively banning it*
- If they don't feel like they could manage 2-3 interns yet, they shouldn't be using agents.

LLM Usage

- Environment
- Equity
- Centralization
  - Market can only support 3-4 big players, possibly fewer
  - Anthropic buying up projects reminiscent of Google's *spending spree*
- Destruction of F/OSS Ecosystem
- Slop
- Brainrot
- leaving only the worst parts of the job
- Increased performance pressure, stagnant pay
- Market Reality
- Mostly destroys language barrier?
- Writing YAML is worse
- Ability to prototype quickly, less attached to v1
- Potentially enables single-actors
- Maybe it is time to just end copyright as we know it?
- Much less concerned about LLMs as data processing tools/etc., fancy autocomplete of boilerplate, etc.

What's Exciting (to me) in Tech

- Hypermedia Systems: merging power of open web with modern needs/browser features without complex JS build chains.
- CRDTs: Conflict-free replicated data types. Algorithmic approach to cross-device data syncing without need for central authority.
- Rust and other memory-safe languages with powerful tooling.
- NixOS declaratively-defined Linux systems. (You're looking at one!)
- Fediverse: decentralized social media built on *protocols*.

Other Advice

- Not everything needs to be in your head, the next couple of years of growth come through repeated exposure.
- Software engineering is much more than just programming. At the end of the day, code is just a tool. The context matters a lot especially in orgs with predominantly non-technical people. Understand them as much as you understand the code.
- If you can read other people's code, you can write almost anything.

via Michael Plunkett & Divij Sinha (CAPP '24)

Praveen: Ed Post

Using and/or creating Free/Open Source Software

Know your licenses: MIT, BSD, Apache vs. GPL/AGPL

Pick a license you can live with. Resist temptation to write your own for anything serious.

Freedom Zero / "Do No Evil"

Take time to understand your dependencies license (and community norms).

General Rules

- Disclose usage. Preserve license notice.
- Publish changes: required by GPL, best practice by convention.

Your Own Philosophy

- What kinds of tools do you like working with?
- How do you approach a new problem with an unknown solution?
  - Plan or Prototype?
- How do you architect your own code?
  - Functional? OOP? Procedural? Declarative?
- How do you think about abstraction?
  - Locality of action vs. General-purpose code.
- How do you prefer to work on a team?
  - Defined role? Generalist?

What is your *least favorite* thing about your *favorite* tool?

Your Own Philosophy, continued

- What motivates you?
- What did you come here to do?
- How does technology inform your solutions?
- What *won't* you work on?

> *With great power comes great responsibility*

-(Uncle) Ben Parker

Managing Up

- Take time to understand their goals. Walking into a new team and making too many changes right away typically won't be well-received, you are likely to not yet understand.
  - Ask *why*, but mean it, have an open mind.
  - Take notes, what was hard to onboard to? What doesn't make sense?
- On the other hand, **you are part of the team**, they hired you because they want you to help build it.
- I have spent time learning how to be most effective and it is when I ...
  - Use certain tools.
  - Have uninterrupted time.
  - Learn things my own way.
- Take 1:1s seriously. Look for tech mentors that do the same.
  - *Ask for mentorship.*
- Labor has power in unity. Support your colleagues and ask for support.
- Save "going around/above" for situations with little left to lose. But be ready to part ways if it fails.

Bosses I Have Had (2006-2020)

- #1: *Do you like solitude?*
- #2: *Speak up.*
- #3: *Why did you do that if it wasn't in the issue tracker?*
- #4: *I pit smart people against one another.*
- #5: Tell me what you need to grow and be successful.
- #6: Aren't you *loyal to this place*?
- #7: 2 weeks.
- #8: Good, but clearly wanted to be an IC again.
- #9: *Why would we hire a woman with an English degree?*

Peter Principle: *People who are competent are promoted until they find themselves in a job they aren't well-suited to.*

A good manager helps you understand what you are building and why. *How* is your job-- but don't be afraid to ask for help!

Being a Manager/Tech Lead

- Pure IC paths exist, management not the only route to promotion (in every org).
- If you are going to lead. Lead. Don’t accept a position and then be afraid to use it.
  - Take it seriously. You are a big part of organization’s success, people’s lives. Make work a nice place to be.
  - 1:1s matter, invest time in growing your teammates’ skills.
  - Practice giving feedback the way you’d want to hear it.
  - Foster an environment where people can tell you when you are wrong.

People, not companies or institutions.

Imposter syndrome comes with the field. Things change fast.

You have built real things. You have climbed the steepest part of the learning curve.

Take risks, continue to bet on yourself.

Work that matters still requires rest and relaxation. **Burnout** is a major risk.

This can be especially hard when it *matters* to you.

Value your time and labor.

A frustrating reality of our world is that work *in the public interest* does not pay as well as selling ads or building weapons.

At the same time, don't let people take advantage.

Discussing salaries is legal in the US. Doing so is often one of the only ways to ensure equity.

Find & cultivate community.

Python, Django, Civic Tech, Open Source, CAPP, ...

Questions?

anything

Thank You

...for your participation in this class.

...for every conversation, all the hard work, and helping make this a better place these last two years.

...for the work you will do after leaving here.