Why Software Engineering?

SECT.00.2026

- What is hard about writing good software?
- What is software engineering?
- How does civic tech influence this course’s perspective on software engineering?
- How will the course & project work?

What is hard about writing high-quality software?

- syntax?
- algorithms & data structures?
- finding a scalable solution within constraints
- understanding & solving the right the problem (design)
- maintaining ability to iterate as understanding of problem evolves (maintenance)

What constitutes high-quality software?

- Reliable
- Secure
- Maintainable
- Upgradable
- Empowers people*
- Affordable*

Requires recognizing needs through the full life cycle: design, development, testing, and maintenance.

**Not broadly accepted.*

Programming as Theory Building

> Accepting program modifications demanded by changing external circumstances to be an essential part of programming, it is argued that the primary aim of programming is to have the programmers build a theory of the way the matters at hand may be supported by the execution of a program. Such a view leads to a notion of program life that depends on the continued support of the program by programmers having its theory.

—Peter Naur’s 1985 paper: https://pages.cs.wisc.edu/~remzi/Naur.pdf

Writing code isn’t the hard part, instead we spend most of our time building mental models of the system.

(This is the thing that AI does not help you with and where AI-written programs typically hit a wall.)

> Non-trivial software changes over time. The requirements evolve, flaws need to be corrected, the world itself changes and violates assumptions we made in the past, or it just takes longer than one working session to finish. And all the while, that software is running in the real world. All of the design choices taken and not taken throughout development; all of the trade-offs; all of the assumptions; all of the expected and unexpected situations the software encounters form a hugely complex system that includes both the software itself and the people building it. And that system is continuously changing.

—Jennifer Moore, https://jenniferplusplus.com/losing-the-imitation-game/

A Brief History of Software Engineering

Some of the first "large" programs were written in the 1960s as part of the space program.

The term *software engineering* is widely attributed to Margaret Hamilton, lead of the on-board flight software for Apollo.

Programs are mostly written on punch card, software is almost entirely written for-purpose.

Image: *Margaret Hamilton with Apollo code*

1970s: "Software Crisis"

> The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

—Edsger Dijkstra, The Humble Programmer (EWD340), Communications of the ACM (1972)

Can software be *engineered*?

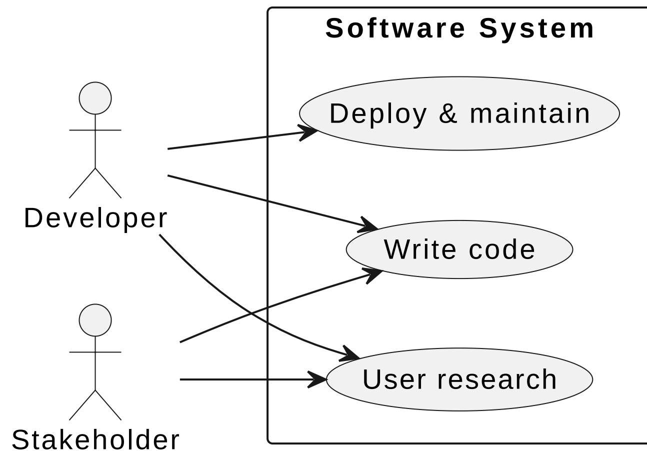
> When I first came up with the term, no one had heard of it before, at least in our world. It was an ongoing joke for a long time. They liked to kid me about my radical ideas. It was a memorable day when one of the most respected hardware gurus explained to everyone in a meeting that he agreed with me that the process of building software should also be considered an engineering discipline, just like with hardware. Not because of his acceptance of the new 'term' per se, but because we had earned his and the acceptance of the others in the room as being in an engineering field in its own right.

—Margaret Hamilton

The term is not universally accepted today, most other engineers require *professional certification*. ^1

1990s: Formal *Software Engineering*

- A renewed attempt to make programming into a formal discipline with standards.
  - Software Engineering Body of Knowledge
  - UML
- not what we're doing



Software Engineering
TENTH EDITION
Ian Sommerville

Software Engineering Body of Knowledge 4.0 (2024)

A 411-page manual that attempts to document how software is made & prescribe solutions to problems.

> Section 3.4 **Problem Resolution**
> The objective of this operations process is to minimize disruption to the business through the identification and analysis of the cause of software and system incidents and problems. This approach may require the involvement of a multidisciplinary team, whose software engineers and operations engineers investigate, for example, recurring production problems that might have an underlying cause in software infrastructure and system components. This might require monitoring, logging and profiling the software and its infrastructure behavior.

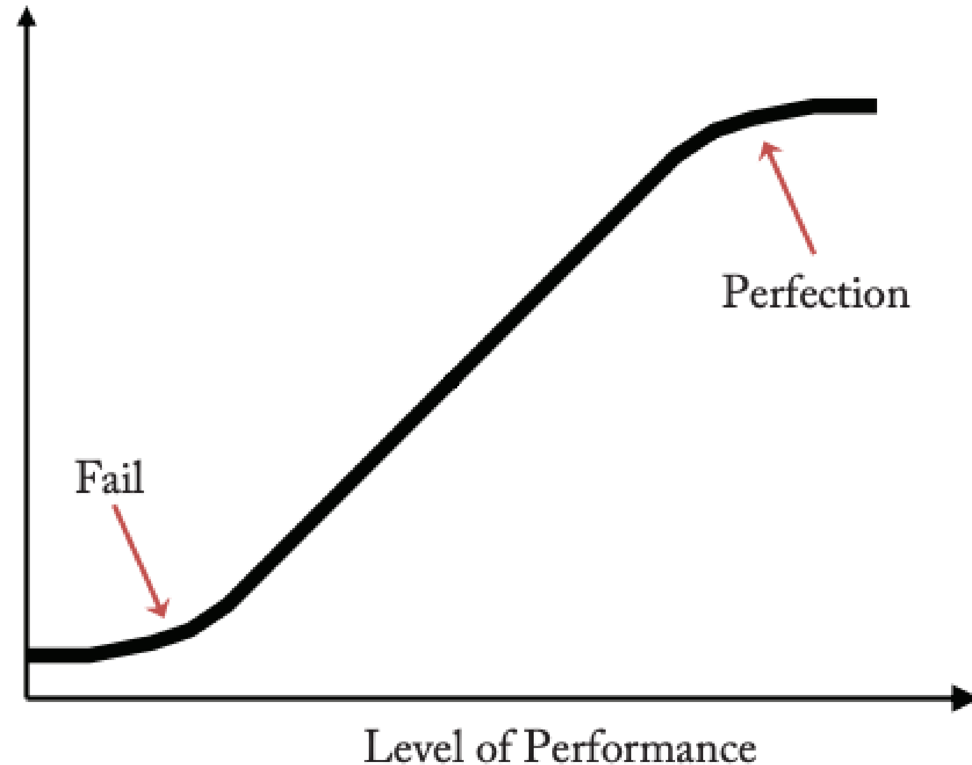


Figure 1.4. Value as a Function of Level of Performance

~2000s

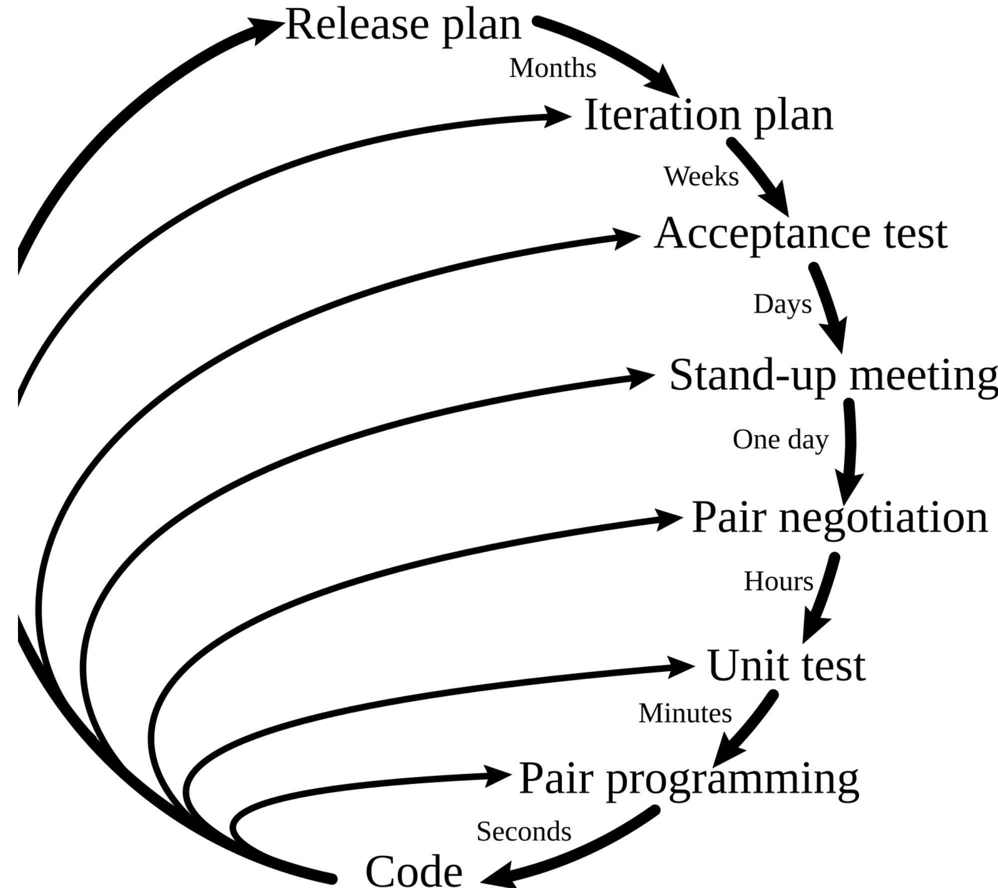
Software (or at least patches) increasingly delivered over the internet.

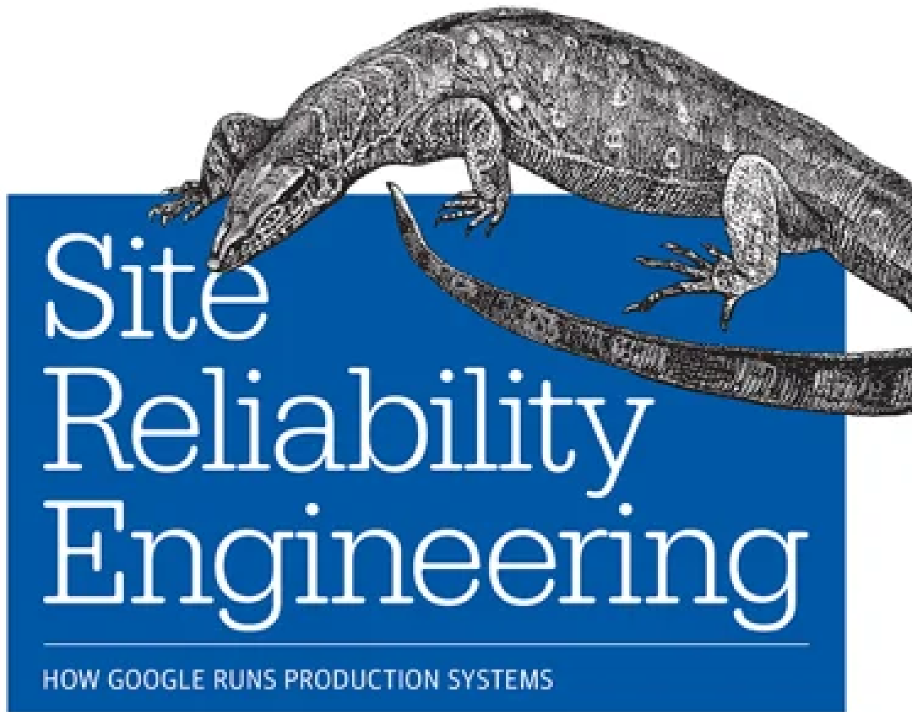
Extreme Programming

- Code review → Pair programming
- Testing → Test-first
- Iteration → Short Release Cycles
- Customer feedback → "customer on the team"

Agile Manifesto

> Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan[...]





2010s: Site Reliability Engineering

- Google founds SRE team in 2003, SRE book 2015.
- Inspires broader industry trends:
  - DevOps: combined developers/ops engineers
  - Highly-automated development and release processes: CI/CD.
  - Formalized reliability goals and processes to follow on failure.
  - Emphasis on *observability*.
  - Often associated with *microservices*.

Software Engineering for…

(civic|public interest|small|humane|liberatory) tech.

What’s Different?

Who is building?

Nonprofits, government agencies, startups.

Typically far more resource-constrained than corporate environments. They aren’t alone in this, so is *almost every one else*.

What is being built?

Often essential services: public services, license registrations, democratic infrastructure.

Accessibility & reliability are non-negotiable. Bugs/downtime can have major repercussions.

Who is the audience?

Perhaps the biggest difference: almost any startup will *narrow* their initial audience. (We only support users with the latest iPhone & 32GB of RAM. Our target audience is single women under 40.)

Instead, typically need to build for an entire community or country. Everyone.

So, harder goals, wider audience, more constraints?

Most challenges scale with team size:

- All members of the team need to have compatible mental models of the problem. (theory-building)
- Process complexity is needed to solve coordination among dozens/hundreds.

The magic of small teams:

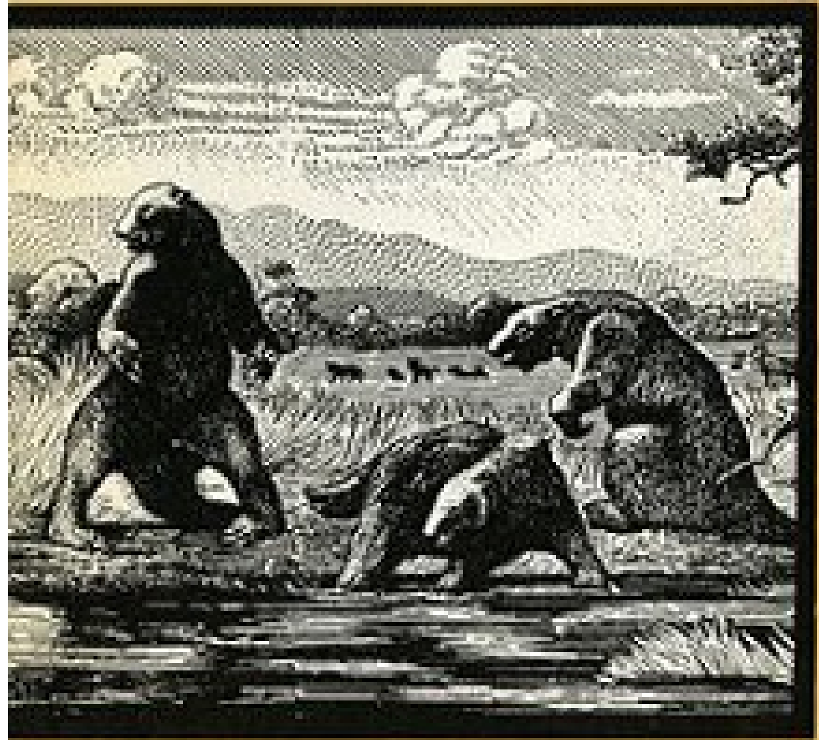
"2024 Google couldn't build 2004 Google"

Brooks's Law: Adding manpower to a late software project makes it later

Also popularized ideas of "surgical teams" and 10x programmers.

mythical man-month

Essays on Software Engineering



healthcare.gov launch

- Affordable Care Act: Barack Obama’s major domestic goal.
- October 2013 - 27,000 signups in first month, a disaster
- There are reports less than 1% of people that tried to use the site were able to.
- Other reports cite the number of people that could sign up on the first day as 3. Three people.

What went wrong?

- Latency: 8-120s for a page to load
- Internal errors at integration points between vendor systems.
- *No Monitoring*
- Woefully complex system awaiting those that came to fix.

Why?

- Dozens of contractors & vendors
- Enormously long dev cycles
- Poor communication
- Irrational technology choices

These are human problems, not code.

This is why we'll spend a significant portion of this course talking about how to work on a team, how to make good technology choices, and what it takes to build a resilient system.

Obama’s Tech Surge

Interview with Paul Smith, co-founder of Everyblock, founder of AdHoc.

Small team, given lots of autonomy, working in short sprints. ~2 months until peak.

April 2014 - 8 million healthcare.gov signups six months later.

https://www.wired.com/2014/08/healthcare-gov/

Our Course

- **understanding what to build**: user research, planning, iteration
- **working effectively with a team**: planning, documentation, processes
- **writing reliable software**: design, testing, robust code
- **managing complexity**: architecture, documentation

Course Philosophy

- Keep humans/users at the center of what you build.
- Leverage benefits of small teams.
- Take what works for you from XP, Agile, SRE, OSS, etc. Drop the rest.
- Write high-quality code that is tested and extensible.
- Focus on the simplest solution that works: don’t follow trends blindly.

My Experience

- Prototyping/rapid-response "newsapps"/civic tech projects.
- Maintained Open States code & infrastructure for 13+ years.
- CTO to team of 21 devs/designers.
- Tech lead of two backend teams at PBS; led org through major tech transition. (Offshore team in Romania.)
- Principal Architect for two academic projects
- Public Data Lead @ Startup
- 4 months interning at defense contractor breaking radios; 18 months on interoperability between Google/Microsoft/Facebook elections teams; 3 weeks at a very dysfunctional health care startup

Notably Absent

- Working on teams larger than ~20 devs.
- I’ve always chosen to work *outside* government, not within.
- I’m notoriously and irrationally averse to selling things.

Elements of My Philosophy

- Most software is complex for bad reasons, simpler is usually better.
- Model data, then straight to prototype for most projects. (Helpful to have something in front of people.)
- heavily influenced by Open Source/Hacker Ethic:
  - Sharing & openness should be defaults.
  - Decentralization is an important safeguard against authority/corruption.
  - *"You can create art and beauty on a computer"*
  - Computers can change the world for the better

Course Goal: begin to develop your own philosophy

Key Course Expectations

Come prepared to participate:

- Several in-class activities.
- Required & recommended readings
- Discuss what’s happening on your projects

Team expectations:

- ~10 hours/week on your project sustained through quarter. (This is not a Week 7-9 project!)
- Communicate issues early. It is hard to build in 9 weeks, every day counts!
- Team-decided guidelines should be treated like course policy. **Escalate to course staff if resolution needed!**

Think of this course as one software 'company'.

- James: CTO
- Praveen: Sr. Dev
- 4-5 product teams, but we’re all in this together

Syllabus/Projects

https://notes.jpt.sh/30320

Next Time: Team Brainstorming/Pairing

Before: Fill out survey if you haven’t already!

Be here on time & ready to participate: participation point!

Extra Session: Weds after class? Friday?