CSE 403
Software Engineering

Pragmatic Programmer Tip: Care about Your Craft

Why spend your time developing software unless you care about doing it well?

What is software engineering?

- Software engineering ≠ programming
- Software engineering ≠ computer science
- **Software engineering**: Creating and maintaining software applications by applying technologies and practices from computer science, project management, and other fields.
- Software engineering is about people working in teams under constraints to create value for their customers
- Software engineering is a discipline.

The first step toward the management of disease was replacement of demon theories and humours theories by the germ theory. That very step, the beginning of hope, in itself dashed all hopes of magical solutions. It told workers that progress would be made stepwise, at great effort, and that a persistent, unremitting care would have to be paid to a discipline of cleanliness. So it is with software engineering today. -- *Fred Brooks*

Aspects of software engineering

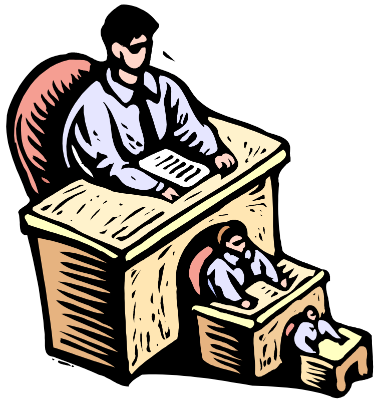
1. Processes necessary to turn a concept into a robust deliverable that can evolve over time
2. Working with limited time and resources
3. Satisfying a customer
4. Managing risk
5. Teamwork and communication

Ties to many fields

- **computer science** (algorithms, data structures, languages, tools)
- **business/management** (project mgmt, scheduling)
- **economics/marketing** (selling, niche markets, monopolies)
- **communication** (managing relations with stakeholders: customers, management, developers, testers, sales)
- **law** (patents, licenses, copyrights, reverse engineering)
- **sociology** (modern trends in societies, localization, ethics)
- **political science** (negotiations; topics at the intersection of law, economics, and global societal trends; public safety)
- **psychology** (personalities, styles, usability, what is fun)
- **art** (GUI design, what is appealing to users)

Necessarily “softer” than other parts of CS; fewer clearly right/wrong answers

Roles of people in software



- **customer** / client: wants software built
  - often doesn't know what he/she wants
- **managers**: make plans, coordinate team
  - difficult to foresee all problems and issues in advance
- **developers**: design and write code
  - it is hard to write complex code for large systems
- **testers**: perform quality assurance (QA)
  - it is impossible to test every combination of actions
- **users**: purchase and use software product
  - users can be fickle and can misunderstand the product

Making software is hard – Pitfalls to avoid

People	Process	Product	Technology
• Undermined motivation • Weak personnel • Uncontrolled problem employees • Heroics • Adding people to a late software project • Noisy, crowded offices • Friction between developers and customers • Unrealistic expectations • Lack of effective project sponsorship • Lack of stakeholder buy-in • Lack of user input • Politics placed over substance • Wishful thinking	• Overly optimistic schedules • Insufficient risk management • Contractor failure • Insufficient planning • Abandonment of planning under pressure • Wasted time during the "fuzzy front end" • Shortchanged upstream activities • Inadequate design • Shortchanged quality assurance • Insufficient management controls • Premature or overly frequent convergence • Omitting necessary tasks from estimates • Planning to catch up later • Code-like-hell programming	• Requirements gold-plating • Feature creep • Developer gold-plating • Push-me, pull-me negotiation • Research-oriented development	• Silver-bullet syndrome • Overestimated savings from new tools or methods • Switching tools in the middle of a project • Lack of automated source-code control

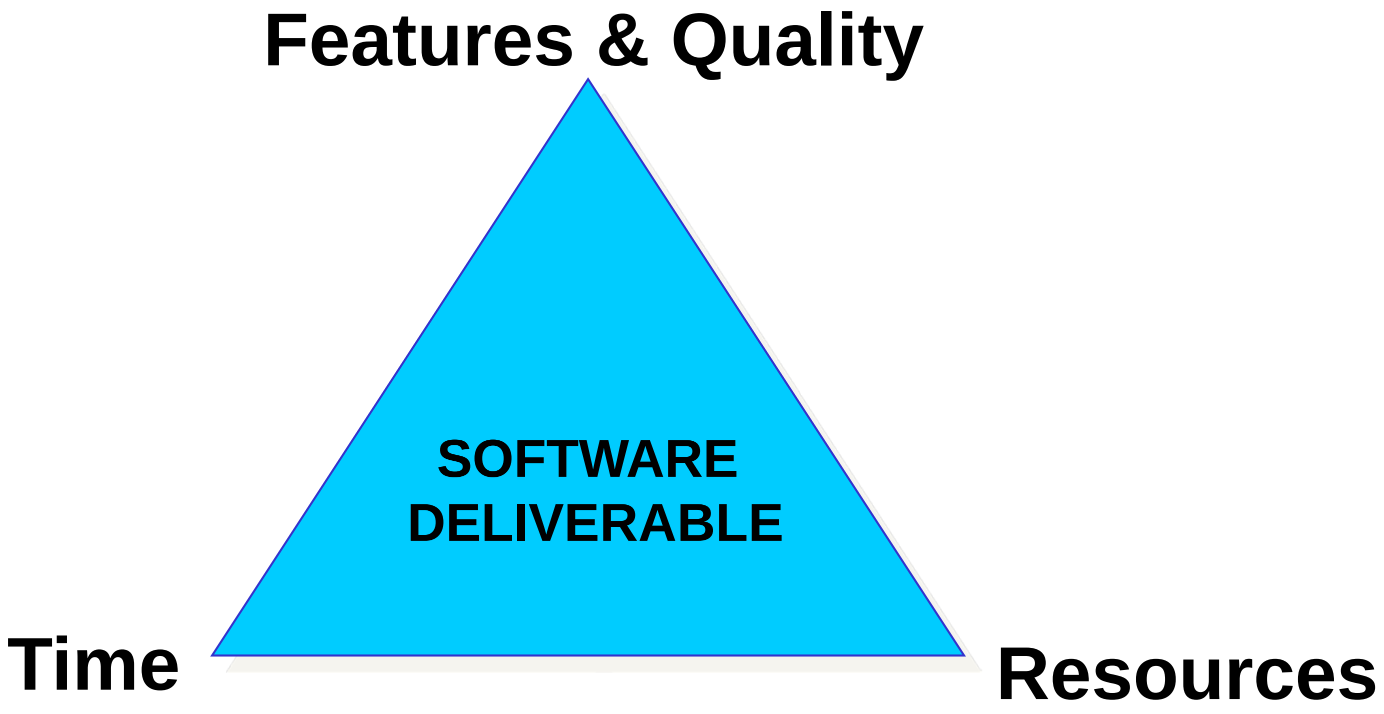
Group project

Gives you experience with the material

- You'll meet *technical challenges* given the larger project
- You'll meet *social challenges* given the team effort
  - Frequent meetings (at *minimum*, each Tuesday)

What is a software project?

Projects are a balance of three dimensions, with the goal of producing a successful deliverable



“Good, fast, cheap … choose two”

The Project

- You make product proposals
  - And then vote on which products to "fund"
- You're divided into project teams
  - Larger teams, larger projects, like industry
- You develop your deliverable in stages
  - See next slide
- Another team will act as your customer
  - A project is successful only if it satisfies its customer

Project development stages

Project development in stages

- Proposal
- Requirements
- Design
- Implementation
- Testing, validation, verification
- Documentation
- Customer exposure
- Final deliverable

- Reflects modern methodologies for effective development
- Regular feedback from customers and your own team

Assignment 1 - Pitches

- Your chance to turn a great idea into a product!
- Prepare a pitch
  - Vision
  - Software architecture
  - Novelty
  - Challenges and risks
- Present in class
- Vote
  - Rank your choices
  - Self-select groups (or the staff will…)

Project culture

- This is a real project
  - We expect you to work to build a real system
  - To be used by real people

- Take responsibility
  - Take initiative
  - Find and solve problems yourselves
  - Coding is only part of the job
  - Good planning and design, hitting your market, and working well with your team, are all needed for success

Communication

- Foundation of the success of our team was communication
- Team communication and cooperation are all-important
- Working together (physically) was good
- Well-run and consistently scheduled meetings help a project a lot

Lessons from past students

Scheduling

- We often underestimated tasks. If we had spent more time analyzing each task and breaking it down into more manageable chunks our estimated completion times would have been more accurate.
- Get things done early; don't cram at the end
- Remember you can cut features (triple constraint)
- Don't underestimate the difficulty of learning new programming languages, frameworks, and tools

Lessons from past students

Testing and coordination

- Thoroughly testing your code and ensuring that your code passes all current tests before submitting is very helpful
- We needed a better upfront testing design

- We learned (through some pain) to ensure to do small, frequent updates and commits. Failing to do this results in merges that can be a nightmare.

Goals of 403 (What's in it for you?)

- see how software is produced, from idea to ship to maintenance
- get exposure to software development practices in use today
- get experience collaborating in a team toward a common goal
- be able to articulate and understand ideas
- understand issues and tradeoffs in decisions as a manager

Unique aspects of CSE 403

- cross-disciplinary nature of the subject
- larger teams
- propose and work on your own ideas
- course staff in the "coach" role
- mistakes along the way are encouraged, not penalized
  - have a rationale; don’t make the same mistake twice
- few clearly right/wrong answers
- plans always change
- content: software design, testing, project management, etc.

Could you learn just as much at an internship?

Probably not:

- Focused on one role in the team (often dev. or test)
- Requirements, arch, high-level design may be set
- Less opportunity for reflection
- Less generalization (such as from reading and discussing papers)
- Mentor may be more focused on results than process and developing you as an engineer

Internships are complementary to CSE 403

People who have had internships learn *different* things in CSE 403, but no *less*

Is software engineering different?

Are the problems faced in software any different than those faced in other engineering fields?

- Arguments in favor:
  - testing software quality is hard (example: the halting problem)
  - lower barrier to entry
  - immaturity of the discipline
  - customer expectations: quality, delivery timeline, etc.
  - fast pace of technological change
  - software is easier to copy
- Arguments against:
  - software isn't always "soft"
    - change is not easy, yet requirements do change
    - change often forces a rewriting of major parts of the software
  - developers still need to plan, execute, test, and sell
  - the discipline is still in its infancy