Software Development Process

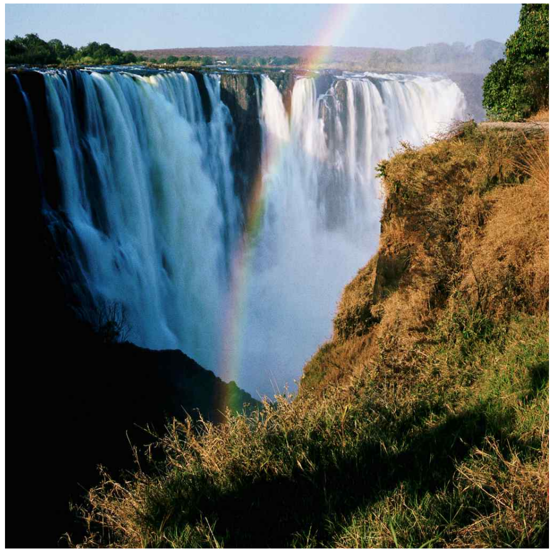
Chung-Kil Hur

(Credit: Byung-Gon Chun & Many Slides from UCB CS169 taught by Armando Fox and David Patterson)

Outline

- SW Development Processes: Plan & Document
- SW Development Processes: Agile Manifesto
- It Takes a Team: Two-Pizza and Scrum
- Pair Programming

Plan-and-Document processes: Waterfall vs. Spiral



How to Avoid SW Infamy?

- Can’t we make building software as predictable in schedule, cost and quality as building a bridge?
- If so, what kind of development process to make it predictable?

Software Engineering

- Bring engineering discipline to SW
  - Term coined ~ 20 years after 1st computer
  - Find SW development methods as predictable in quality, cost, and time as civil engineering
- "Plan-and-Document"
  - Before coding, project manager makes plan
  - Write detailed documentation all phases of plan
  - Progress measured against the plan
  - Changes to project must be reflected in documentation and possibly to plan

1st Development Process: Waterfall (1970)

- 5 phases of Waterfall "lifecycle"

1. Requirements analysis & specification
2. Architectural design
3. Implementation & Integration
4. Verification
5. Operation & Maintenance

- Complete one phase before start next one
  - Why? Earlier catch bug, cheaper it is
  - Extensive documentation/phase for new people

How well does Waterfall work?

- *And the users exclaimed with a laugh and a taunt: "It's just what we asked for, but not what we want."*
  - *Anonymous*
- Often when customer sees it, wants changes to SW product

How well does Waterfall work?

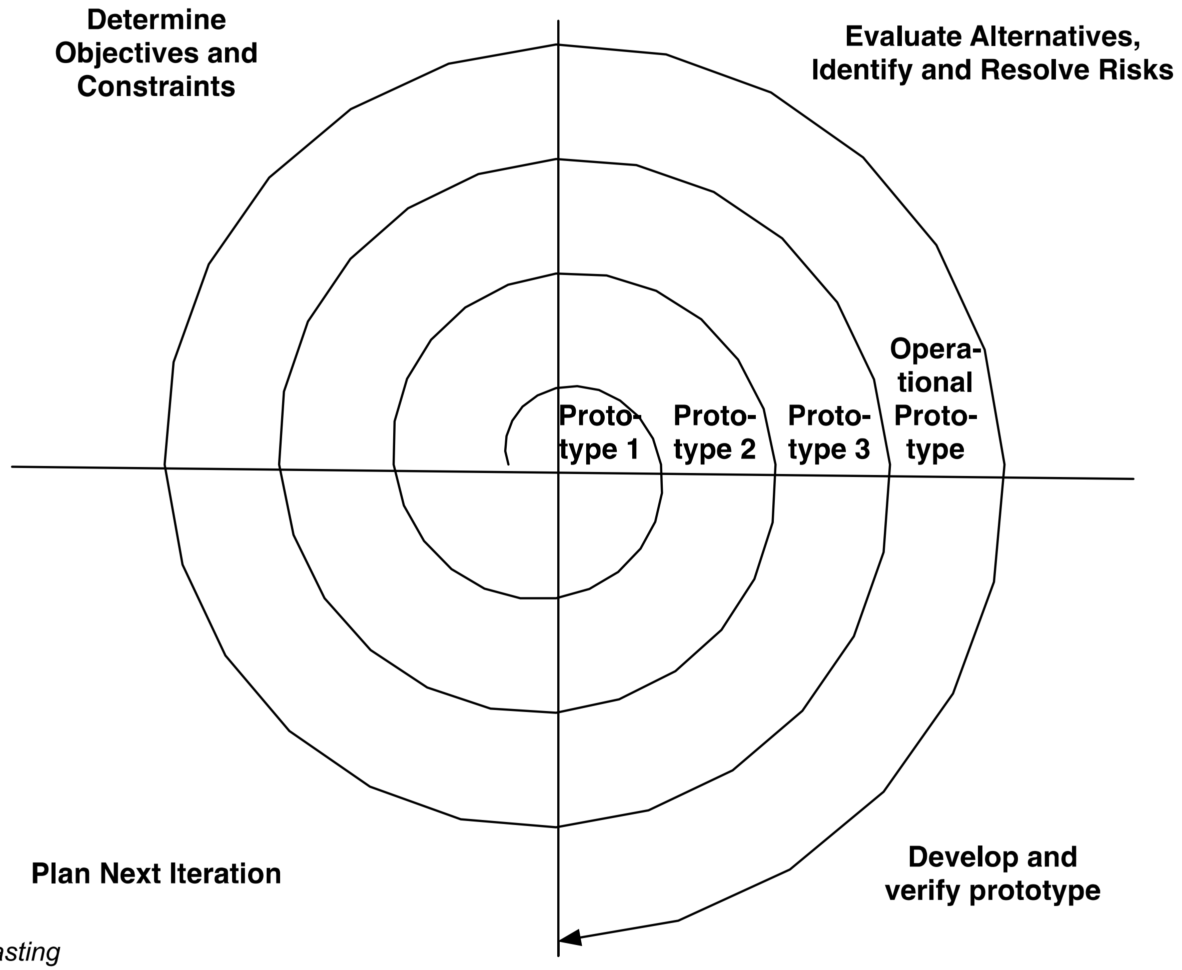
- *“Plan to throw one [implementation] away; you will, anyhow.”*
  - Fred Brooks, Jr.
    (1999 Turing Award winner)
- Often after build first one, developers learn right way they should have built it

(Photo by Carola Lauber of SD&M www.sdm.de. Used by permission under CC-BY-SA-3.0.)

Spiral Lifecycle (1986)

- Combine Plan-and-Document with prototypes
- Rather than plan & document all requirements 1st, develop plan & requirement documents across each iteration of prototype as needed and evolve with the project

Spiral Lifecycle


Determine Objectives and Constraints
Evaluate Alternatives, Identify and Resolve Risks
Proto-type 1
Proto-type 2
Proto-type 3
Opera-tional Proto-type
Plan Next Iteration
Develop and verify prototype

(Figure 1.4, *Engineering Long Lasting Software* by Armando Fox and David Patterson, 2nd Beta edition, 2013.)

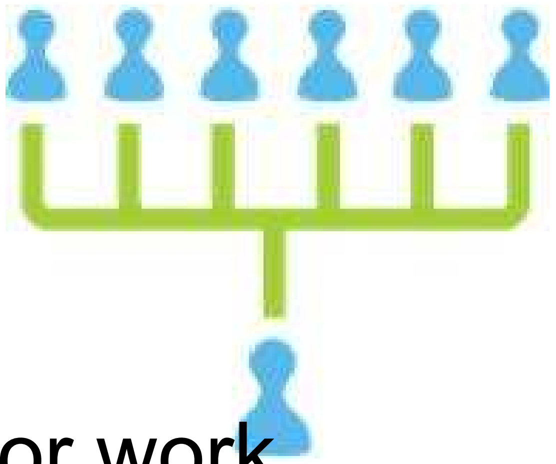
P&D Project Manager

- P&D depends on Project Managers
  - Write contract to win the project
  - Recruit development team
  - Evaluate software engineers performance, which sets salary
  - Estimate costs, maintain schedule, evaluate risks & overcomes them
  - Document project management plan
  - Gets credit for success or blamed if projects are late or over budget

P&D Team Size

"Adding manpower to a late software project makes it later."

Fred Brooks, Jr.,
The Mythical Man-Month



- It takes time for new people to learn project
- Communication time grows with size, leaving less time for work
- Groups 4 to 9 people, but hierarchically composed for larger projects

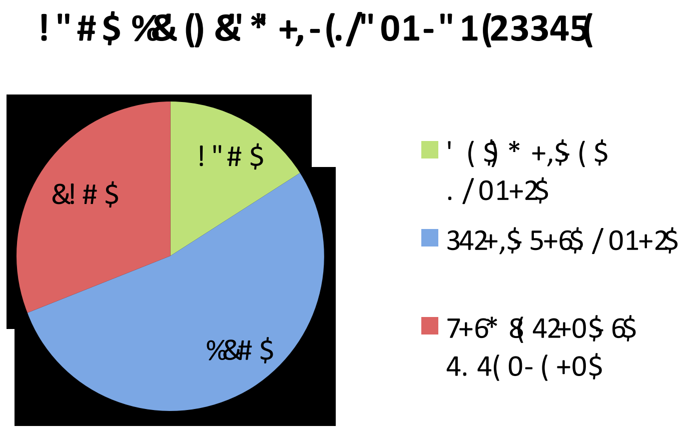
Agile Development Process

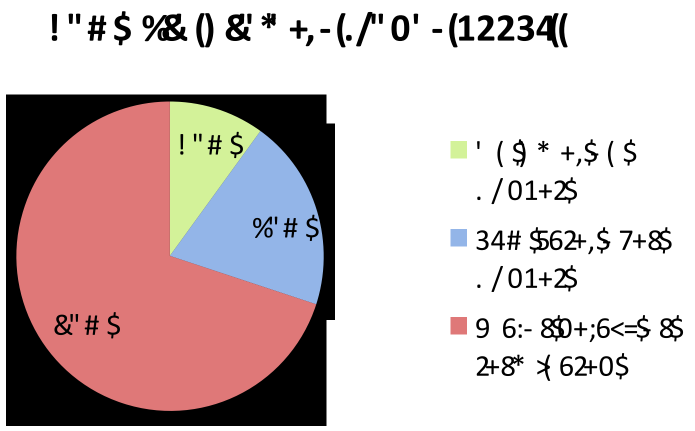
Alternative Process?

- How well can Plan-and-document hit the cost, schedule & quality target?
- P&D requires extensive documentation and planning and depends on an experienced manager
  - Can we build software effectively without careful planning and documentation?
  - How to avoid “just hacking”?

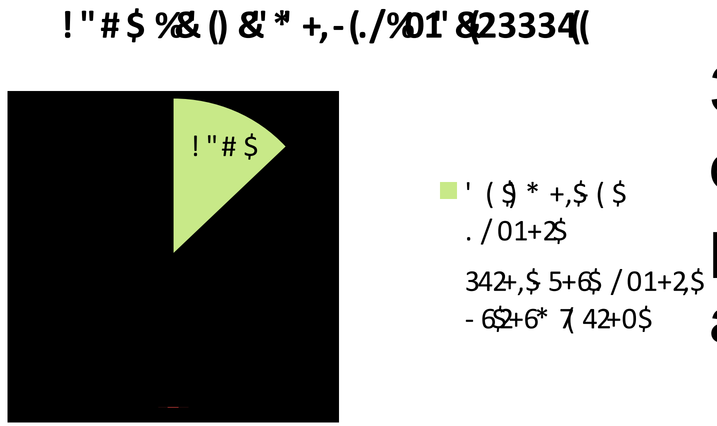
How Well do Plan-and-Document Processes Work?

- IEEE Spectrum "Software Wall of Shame"
  - 31 projects (4 from last lecture + 27): lost $17B





3X



3/~500 new development projects on time and budget

(Figure 1.6, *Engineering Long Lasting Software* by Armando Fox and David Patterson, 2nd Beta edition, 2013.)

Agile Manifesto, 2001

“We are uncovering better ways of developing SW by doing it and helping others do it. Through this work we have come to value

- Individuals and interactions over processes & tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

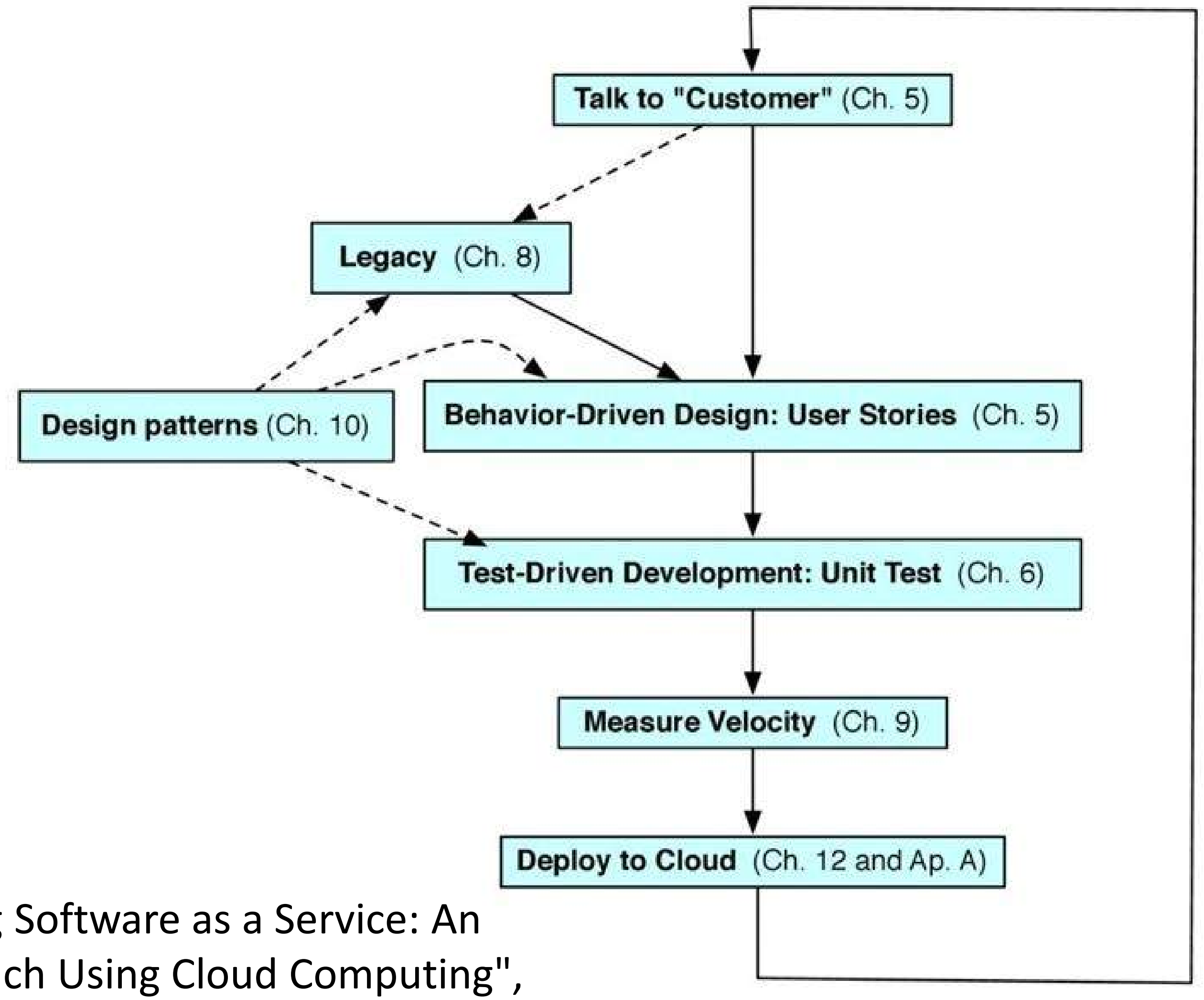
“Extreme Programming” (XP) version of Agile lifecycle

- If short iterations are good, make them as short as possible (weeks vs. years)
- If simplicity is good, always do the simplest thing that could possibly work
- If testing is good, test all the time. Write the test code before you write the code to test.
- If code reviews are good, review code continuously, by programming in pairs, taking turns looking over each other’s shoulders.

Agile lifecycle

- Embraces change as a fact of life: continuous improvement vs. phases
- Developers continuously refine working but incomplete prototype until customers happy, with customer feedback on each Iteration (every ~1 to 2 weeks)
- Agile emphasizes Test-Driven Development (TDD) to reduce mistakes, written down User Stories to validate customer requirements, Velocity to measure progress

Agile lifecycle



Talk to "Customer" (Ch. 5)
Legacy (Ch. 8)
Design patterns (Ch. 10)
Behavior-Driven Design: User Stories (Ch. 5)
Test-Driven Development: Unit Test (Ch. 6)
Measure Velocity (Ch. 9)
Deploy to Cloud (Ch. 12 and Ap. A)

"Engineering Software as a Service: An Agile Approach Using Cloud Computing", by Armando Fox and David Patterson

Agile Then and Now

- Controversial in 2001
  - "… yet another attempt to undermine the discipline of software engineering… nothing more than an attempt to legitimize hacker behavior."
    - Steven Ratkin, "Manifesto Elicits Cynicism," *IEEE Computer*, 2001
- Accepted in 2013
  - 2012 study of 66 projects found majority using Agile, even for distributed teams

Yes: Plan-and-Document No: Agile (Sommerville, 2010)

1. Is specification required?
2. Are customers unavailable?
3. Is the system to be built large?
4. Is the system to be built complex (e.g., real time)?
5. Will it have a long product lifetime?
6. Are you using poor software tools?
7. Is the project team geographically distributed?
8. Is team part of a documentation-oriented culture?
9. Does the team have poor programming skills?
10. Is the system to be built subject to regulation?

Fallacies and Pitfalls

- Fallacy: The Agile lifecycle is best for software development
  - Good match for some SW, especially SaaS
  - But not for NASA, code subject to regulations

- Note: you will see new lifecycles in response to new opportunities in your career, so expect to learn new ones

Question: Which statement is FALSE?

- ☐ A big difference between Agile and P&D is that Agile does not use requirements
- ☐ A big difference between Agile and P&D is measuring progress against a plan
- ☐ You can build SaaS apps using Agile, but not with Plan-and-Document
- ☐ A big difference between Agile and P&D is building prototypes and interacting with customers during the process

It Takes a Team:
Size & Scrum

SW Eng now a Team Sport

- Now in Post-Superhero-Programmer Era
- Rising bar of functionality/quality => cannot do SW breakthrough alone
- Successful SW career => programming chops AND plays well with others AND can help make *team* win
- *"There are no winners on a losing team, and no losers on a winning team."*

– Fred Brooks, Jr.

Alternative Team Organization?

- Plan-and-document require extensive documentation and planning and depends on an experienced manager
- How should we organize an Agile team?
- Is there an alternative to a hierarchical organization with one manager who runs the project?

Scrum: Team Organization

- "2 Pizza" team size (4 to 9 people)
- "Scrum" inspired by frequent short meetings
  - 15 minutes every day at same place and time
  - To learn more: *Agile Software Development with Scrum* by Schwaber & Beedle

Daily Scrum Agenda

- Answers 3 questions at “daily scrums”:
  1. What have you done since yesterday?
  2. What are you planning to do today?
  3. Are there any impediments or stumbling blocks?
- Help individuals by identify what they need

Scrum roles

- Team: 2-pizza size team that delivers SW
- ScrumMaster: team member who
  - Acts as buffer between the Team and external distractions
  - Keeps team focused on task at hand
  - Enforces team rules (coding standard)
  - Removes impediments that prevent team from making progress

Scrum roles (cont’d)

- Product Owner: A team member (not the ScrumMaster) who represents the voice of the customer and prioritizes user stories

Resolving Conflicts

- eg. Different view on right technical direction

1. 1st list all items on which the sides agree
   - vs. starting with list of disagreements
   - Discover closer together than they realize?
2. Each side articulates the other's arguments, even if don't agree with some
   - Avoids confusion about terms or assumptions, which may be real cause of conflict

Resolving Conflicts

3. Constructive confrontation (Intel)
   - If you have a strong opinion that a person is proposing the wrong thing technically, you are obligated to bring it up, even to your bosses
4. Disagree and commit (Intel)
   - Once decision made, need to embrace it and move ahead
   - “I disagree, but I am going to help even if I don’t agree.”

- Conflict resolution useful in personal life!

Scrum Summary

- Basically, self-organizing small team with daily short standup meetings
- Work in “sprints” of 2-4 weeks
- Suggest members rotate through roles (especially Product Owner) each iteration

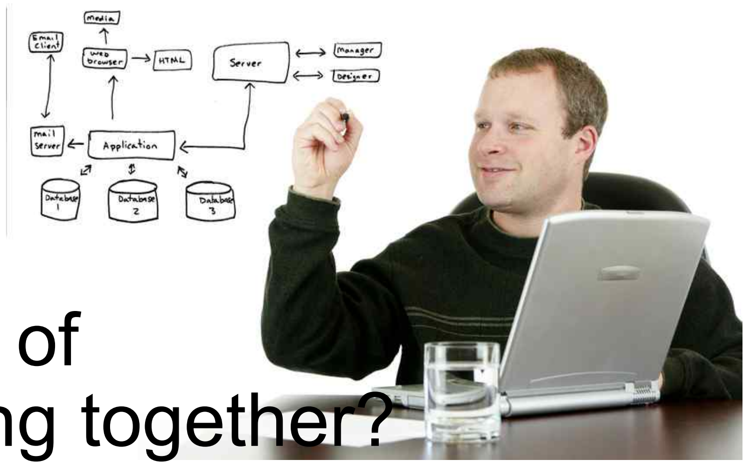
Which statement regarding teams is TRUE?

- ☐ As compared to Scrum, P&D project managers act as both Scrum Master and Product Owner
- ☐ Teams should try to avoid conflicts at all costs
- ☐ P&D can have much larger teams than does Scrum, which report directly to the project manager
- ☐ As studies show 84%-90% projects are on-time and on-budget, P&D managers can promise customers a set of features for an agreed upon cost by an agreed upon date

Pair Programming

Are 2 minds better than 1?

- Stereotypical programmer is lone wolf working all night alone drinking Red Bull
- Is there a more social way to program?
  - What would be the benefits of multiple people programming together?
- How would you prevent one person from doing all the work while the other gets coffee and checks Facebook?



Pair Programming

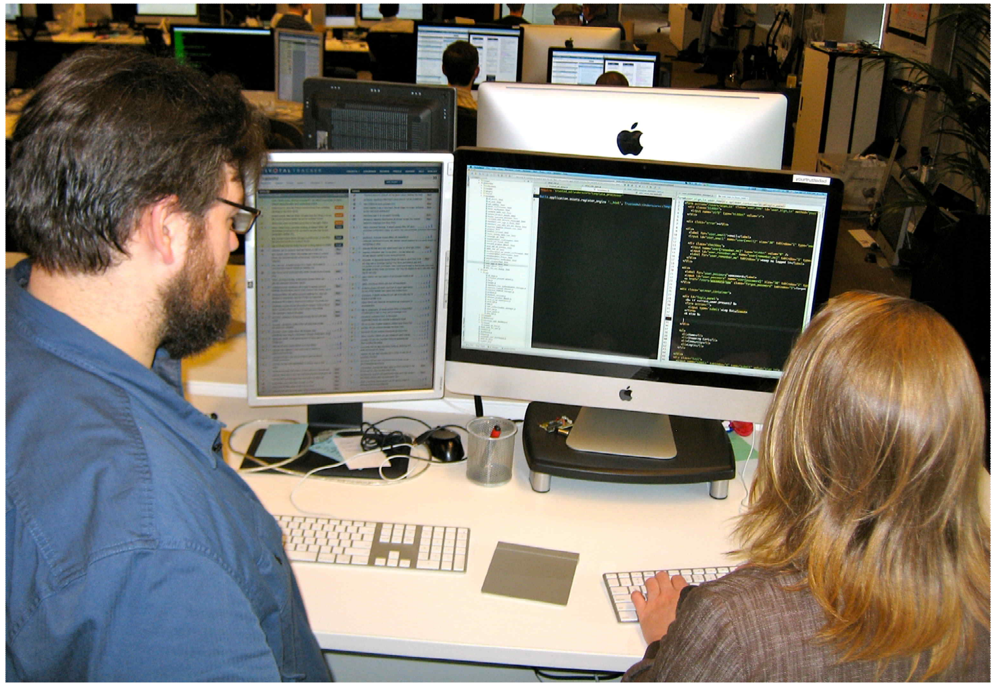
- Goal: improve software quality, reduce time to completion by having 2 people develop the same code
  - Some people (and companies) love it
  - Try in practice sections to see if you do
  - Some students in the past have loved it and used for projects

Pair Programming

Q. At Google, you share an office, and you even code together.
Sanjay: We usually sit, and one of us is typing and the other is looking on, and we're chatting all the time about ideas, going back and forth.

- Interview with Jeff Dean and Sanjay Ghemawat, creators of MapReduce

Pair Programming



- Sit side-by-side facing screens together
  - Not personal computer; many for any pair to use
  - To avoid distractions, no email reader, browser

Pair Programming

- Driver enters code and thinks tactically about how to complete the current task, explaining thoughts while typing
- Observer reviews each line of code as typed in, and acts as safety net for the driver
- Observer thinking strategically about future problems, makes suggestions to driver
- Should be lots of talking and concentration
- **Pair alternate roles**

Pair Programming Evaluation

- PP quicker when task complexity is low
- PP yields higher quality when high
  - Anecdotally, sometimes *more readable* code too
- But more effort than solo programmers
- Also transfers knowledge between pair
  - programming idioms, tool tricks, company processes, latest technologies, …
  - Some teams purposely swap *partners* per task => eventually everyone is paired ("promiscuous pairing")

Pair Programming Do’s & Don’ts

- Don’t fiddle with your smartphone when you are the observer
- Do consider pairing with someone of different experience level—you’ll both learn!
  - Explaining is a great way to better understand
- Do swap frequently—learning goes both ways, and roles exercise different skills
  - Observer gains skill of explaining his/her thought process to Driver

Which expression statement regarding Pair Programming is TRUE?

- ☐ Pair programming is quicker, better quality, cheaper, and less effort than solo programming
- ☐ The Driver works on tasks at hand, the Observer thinks strategically about future tasks
- ☐ A pair will eventually figure out who is the best driver and who is the best observer, and then stick primarily to those roles
- ☐ Promiscuous pairing is one long-term solution to the problem of a shortage of programmers

Summary

- Lifecycles: Plan&Document (careful planning & documentation) v. Agile (embrace change)
- Teams: Scrum as self organizing team + roles of ScrumMaster and Product Owner
- Pair programming: increase quality, reduce time, educate colleagues