Lecture 3: Teams

CS 5150, Spring 2025

Lecture goals

1. Select an appropriate development methodology for a project
    - … (Lecture 2)
    - Agile methods and eXtreme Programming
    - Mixed methodologies
2. Work effectively on a development team
    - Team size
    - Roles, personalities, and growth
    - Meetings
    - Retrospectives and peer evaluation
    - Collaboration tools

Upcoming Deadlines

- Fri, Jan 31: Submit assignment A1
- Mon, Feb 3: Team Formation. Submit Team Registration Survey.
- Last day to propose external projects and form teams; complete team formation survey
- Fri, Feb 7: **Project plan due**
- In-Class exams: Mar 27, May 1

Once you have formed a complete team, you will be assigned a client for internal project.

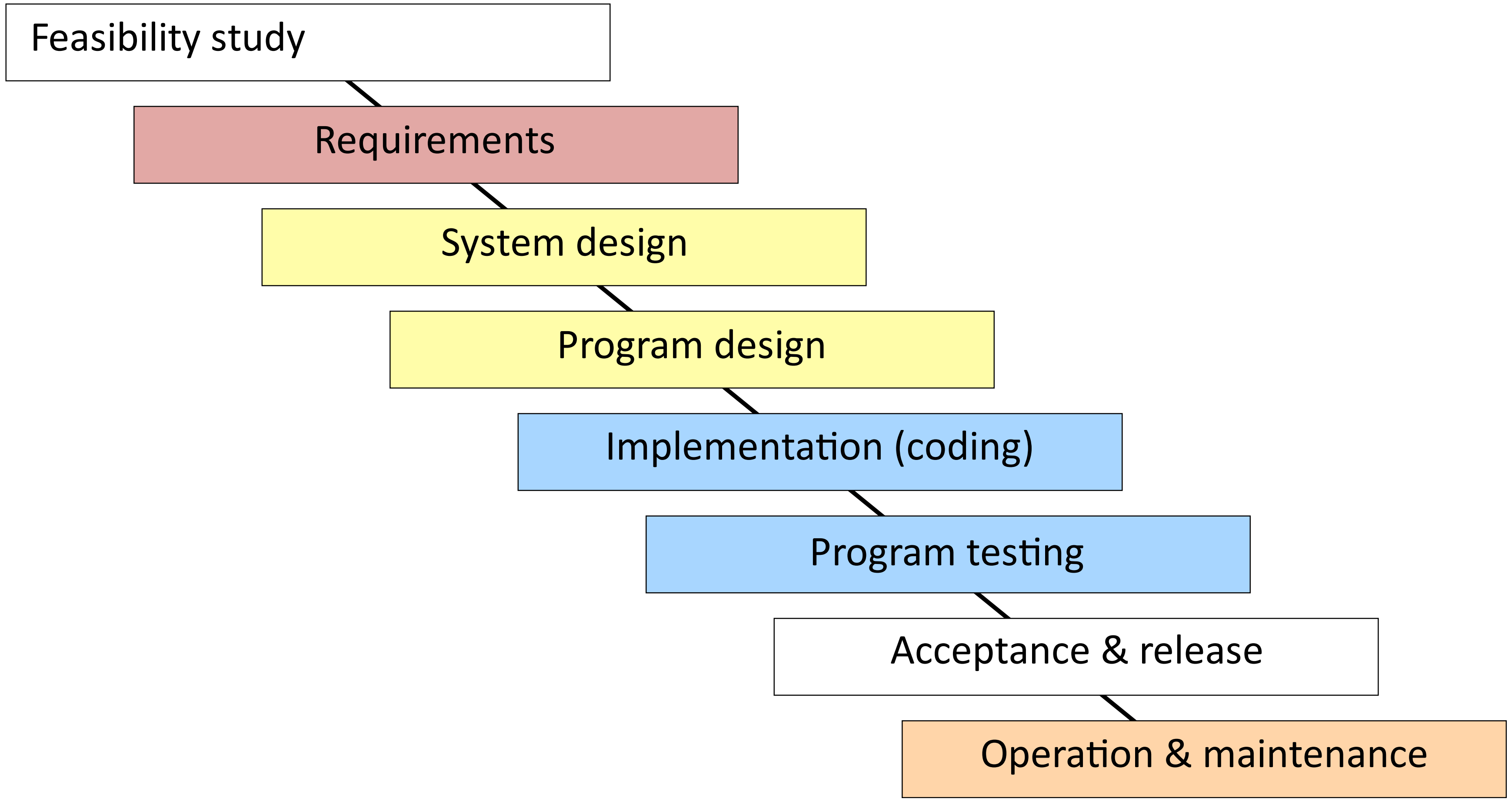
Development methodologies

… continued from Lecture 2

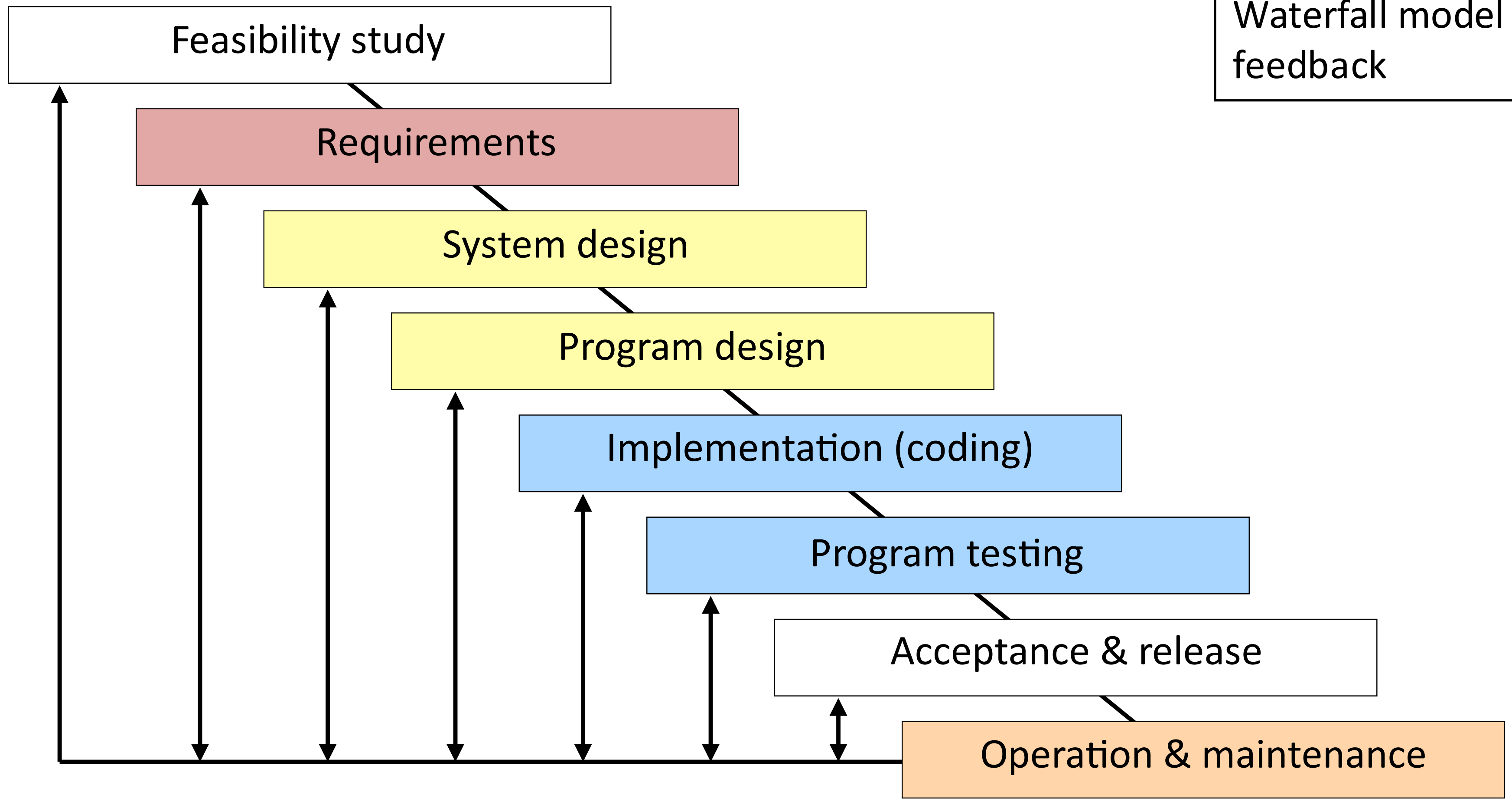
Review

- Waterfall & Modified waterfall
- Iterative refinement
  - Essential for user interfaces
- Incremental Delivery
  - Good for evolving working systems
  - Example: Microsoft Windows

The Waterfall model



Modified Waterfall model



Waterfall model with feedback

Agile Methods vs Plan-Driven Methods

Heavyweight / Plan-driven

- Processes and tools
- Specifications
- Following a plan
- Client negotiation/contracts

Lightweight / Agile

- Individuals and interactions
- Working software
- Responding to change
- Client collaboration

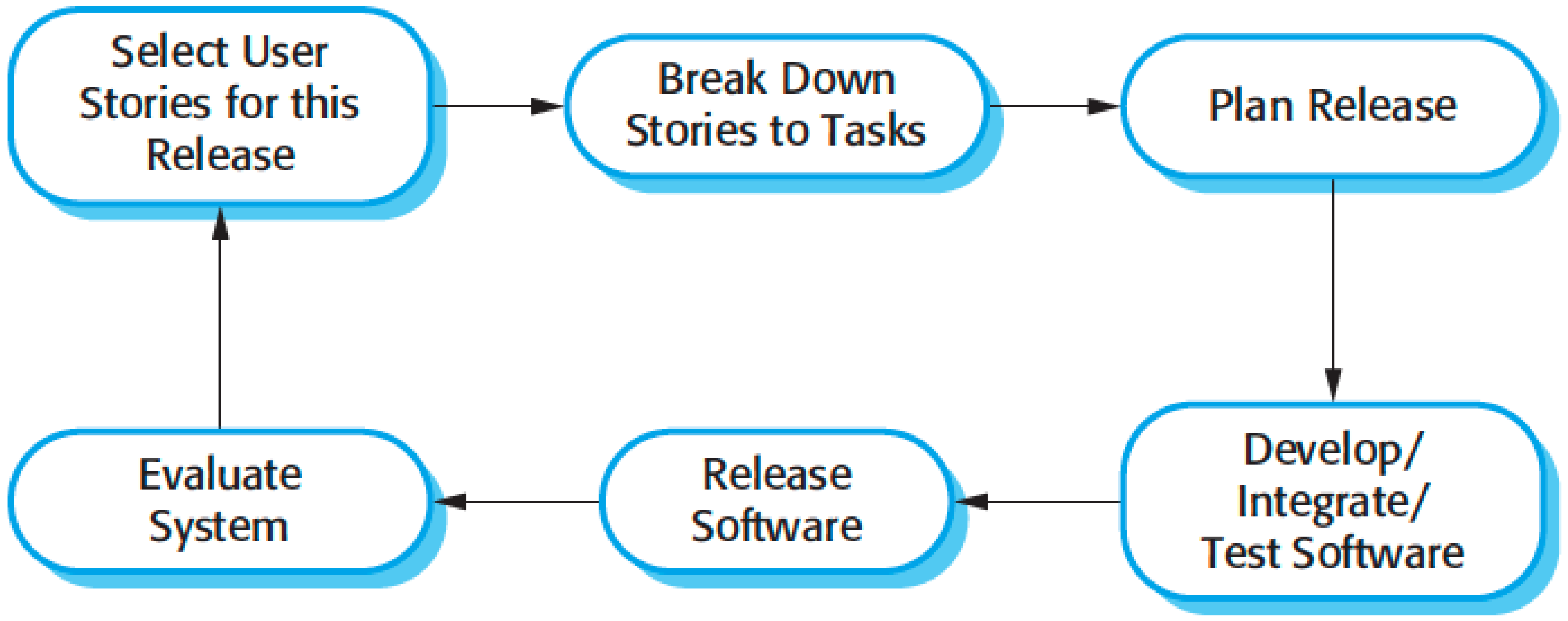
Based on the Manifesto for Agile Software Development
http://agilemanifesto.org/

eXtreme Programming (XP)

Focus on development step

- User stories for requirements
  - Improves communication
- Incremental planning
- Small releases
  - Improves visibility
- Simple design
- Test-first development
  - Shifts left
- Periodic refactoring
- Pair programming
  - Shifts left
- Collective ownership
- Continuous integration
  - Shifts left
- On-site customer
  - Improves communication

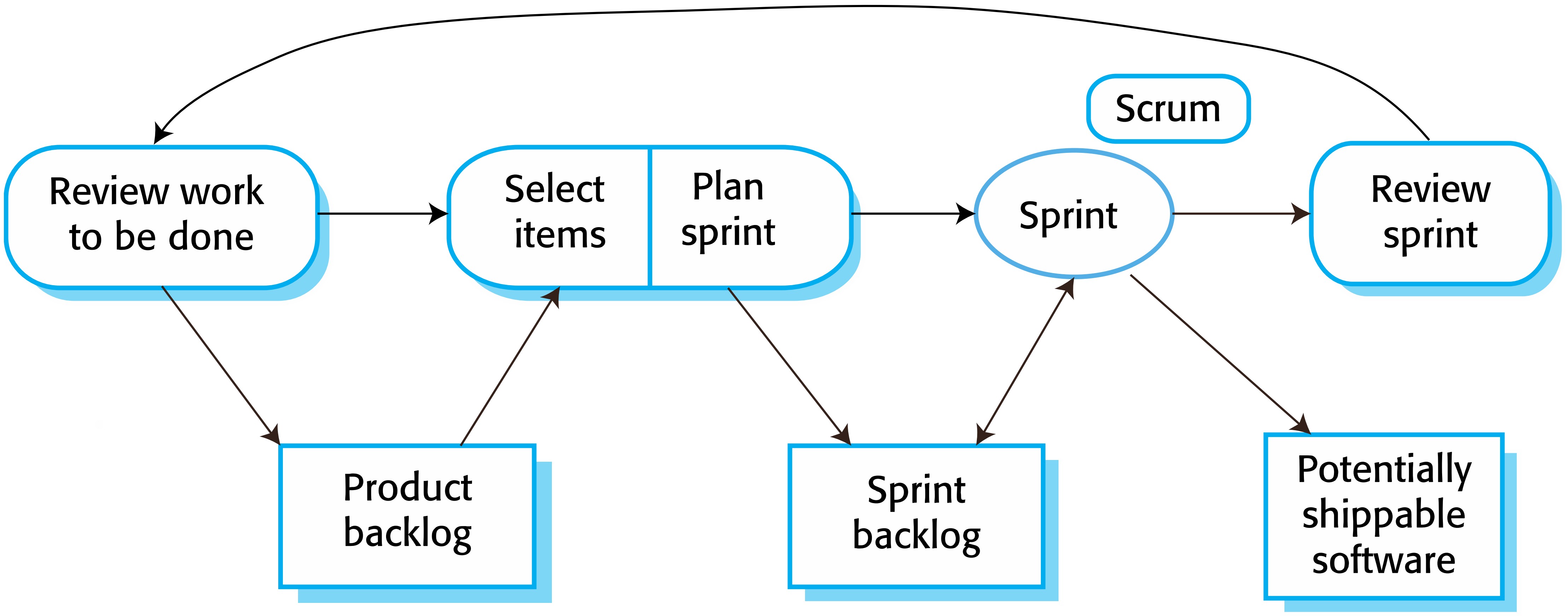
eXtreme Programming (XP)


Select User Stories for this Release
Break Down Stories to Tasks
Plan Release
Develop/ Integrate/ Test Software
Release Software
Evaluate System

Scrum Implementation of Agile

- Provides Agile management structure that accommodates XP
- Work scheduled as "time boxes" (sprints)
  - 2-4 weeks
- Tasks selected from backlog
  - Incomplete work is *not* automatically carried over
- Sprint product is released, production-quality code + docs
  - Sprint planning defines an MVP
- Daily team meetings

Scrum workflow



Agile methods

Benefits

- Good visibility and communication
- Accommodates change, fuzzy requirements
- Very popular today for small, dynamic projects

Challenges

- Tricky to scale to large projects, bureaucratic organizations
- Works best with highly-skilled, autonomous developers
- Hard to validate requirements for completeness
- Lack of formal docs impedes maintenance, handoff

Integration and configuration

- When system design is *standardized*, can better take advantage of **code reuse**
- Providers collect lots of configurable components into **commercial-off-the-shelf** (COTS) products
  - E.g. Enterprise Resource Planning (ERP) platforms
- Developers integrate, configure components based on client requirements
  - Effectively skip system design and program development steps

Pros
- Reduced cost and time

Cons
- Reduced function

Poll

What methodology was used for the FAA AAS?

Was this an appropriate choice?

PollEv.com/cs5150sp25

Mixed processes

Many projects mix elements of multiple methodologies

- If requirements are well-understood, might use **Waterfall** to define requirements & system design, then implement using Incremental Delivery performed in Scrum-like sprints
- If requirements are vague, might use **Iterative Refinement** to clarify requirements, followed by **Modified Waterfall** to build final version (prototype is discarded)
  - Might Integrate & Configure a COTS platform for prototype
- Might develop user interface with iterative refinement, but adopt another process for data store

Phased Development

- Decide at the outset to divide a project into multiple phases
  - First phase product is quickly brought into (limited) production
  - Subsequent phases based on experience from first phase
- Advantages
  - Early benefit from initial investment
  - Clarifies requirements for later phases
  - Costs can be spread out (or subsequent phases can be cancelled)

Summary

- Different development processes are appropriate for different projects
  - Processes can evolve during a project
  - Processes include common process steps
  - Processes must accommodate revision of prior steps
- Purpose of process is to minimize risk. Risk-reduction practices include:
  - Prototyping key components
  - Frequent releases, or decomposition into phases
  - Early and iterative testing with users/customers
  - Promoting visibility

Summary

- **Heavyweight**: Discourages change; more effort upfront to be confident in design choices
  - Beneficial if system has many inter-related components
  - Example use: Lockheed Martin
- **Lightweight**: Accommodates requirements uncertainty
  - Iteration can clarify requirements
  - Agility can respond to novel markets
  - Example use: Amazon

Teams

Why develop in teams?

- Large projects require many people
  - Need to deliver while still relevant
  - Diversity of ideas, peer review improve quality
  - Utilize many different skillsets
    - Managerial
    - Administrative
    - Writing
    - Domain expertise
- Multi-person development requires collaboration
  - Disseminate knowledge
  - Avoid duplicated effort, support
  - Respect dependency, priority order
  - Clarify interfaces
  - Encourage accountability
  - Satisfy esteem needs

Teams should be *small*

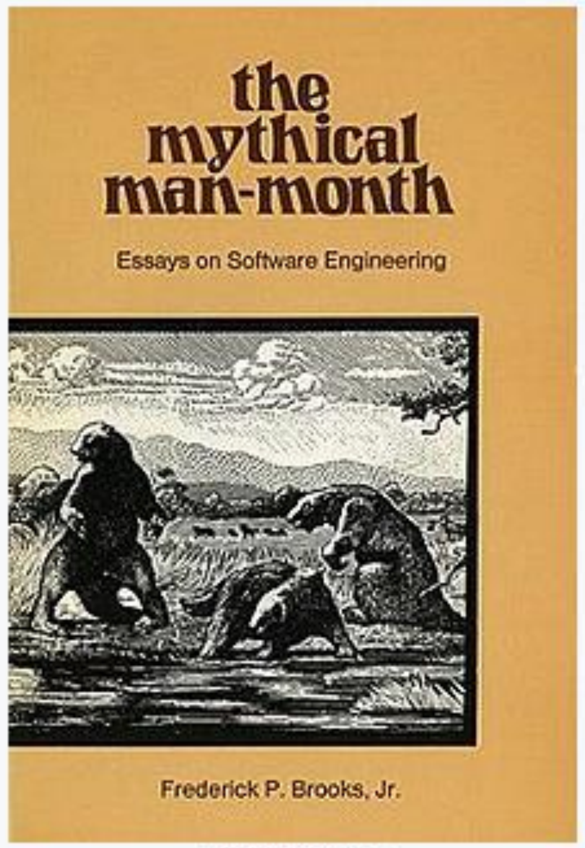
- Number of pairwise communication paths is *quadratic* in team size
- Cannot understand full system, keep pace with all updates
- Need efficient meetings, decisions
- Quality benefits from consistency

- UpShot: effective teams have 4-10 members; sweet spot is around 7
  - Consistent with CS 5150 team size requirements (4-5 students)

- Example: Microsoft
  - 3-5 dev sub-teams; Product: 20-30

The Mythical Man-Month

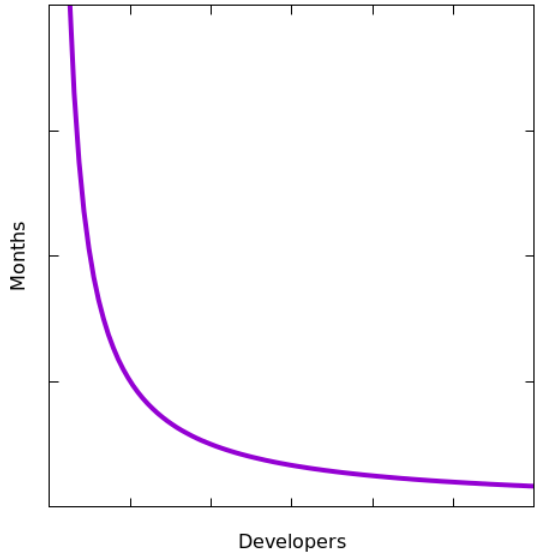
- Development effort often measured in person-months
  - How much does one developer accomplish in one month?
- Suggests that work could be sped up by adding more developers
- Brooks's law: "adding manpower to a late software project makes it later"
  - Onboarding time
  - Communication overhead
  - Indivisible tasks

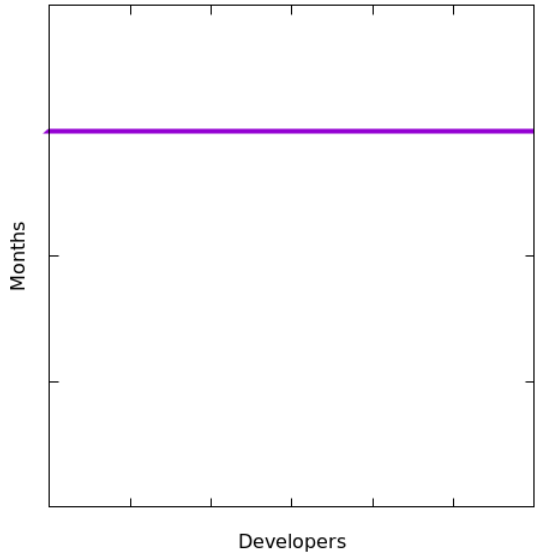


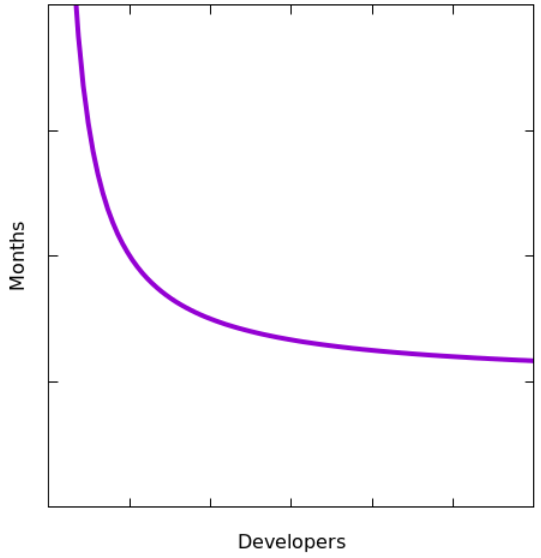
First edition

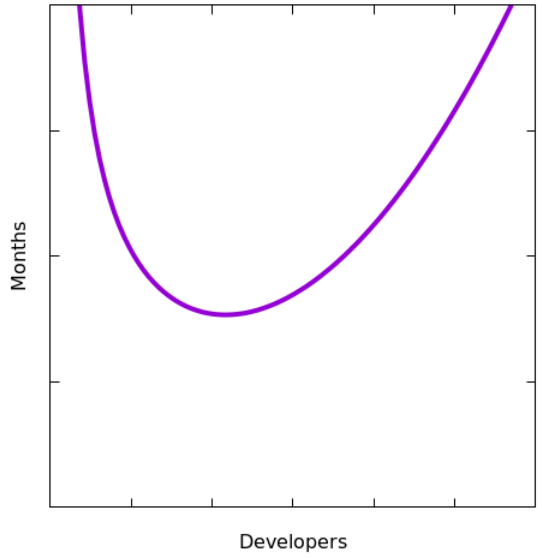
Poll: PollEv.com/cs5150sp25

Which plot shows how project duration varies with manpower for a partitionable task **without** complex interdependencies that requires onboarding?









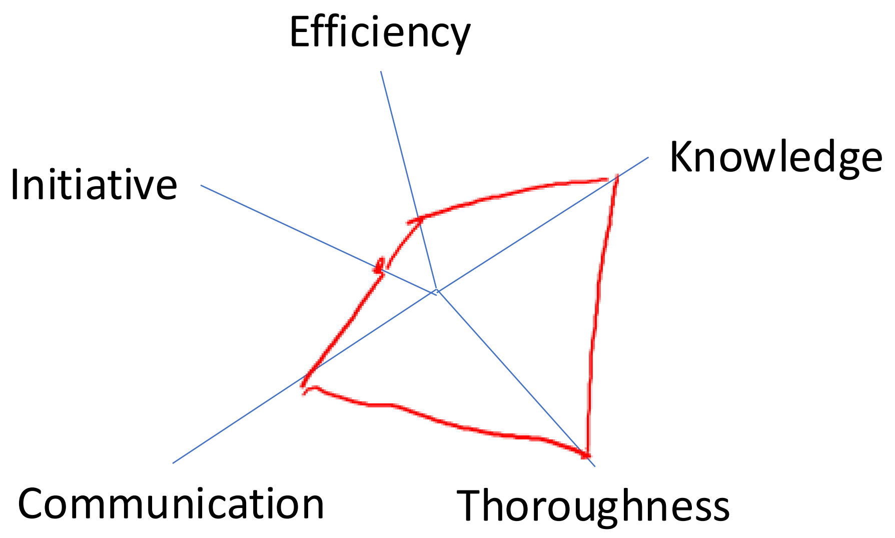
Roles (for CS 5150)

- No “team leader”
  - Project is a shared responsibility
  - Team decisions by consensus
- Team point-of-contact
  - Communicates with client (scheduling, reports)
  - Communicates with course staff
- Project manager
  - Maintains schedule
  - Monitors risks
  - Does *not* dictate tasking
  - May rotate
- Toolsmith
  - Infrastructure expert (DevOps)

Roles during meetings:

- Moderator
- Recorder

Developer strengths & motivations



Bass & Dunteman’s taxonomy:

- Task-oriented
- Self-oriented
- Interaction-oriented

Developer Growth

- Mentorship
- Training
- Self-learning
- Documentation
- Reviews
- Goals (w/ manager assistance)
- Osmosis (slow, shallow)

- At a point in time, some devs may be much more productive than others
- But all devs have the ***potential*** to become more productive
  - Maximize benefit by "leveling up" early
  - "Teach a man to fish" parable
- Will get most out of this course by leveling up your whole team

Discuss: Team Disfunctions

What problems do you anticipate in your project team?

1. Absence of trust
2. Fear of conflict
3. Lack of commitment
4. Avoidance of accountability
5. Inattention to results

Productivity Slang

Yak shaving

Spending lots of time on tangential tasks/reading

- Ask “would teammates agree this is really necessary?”
- Should bring real (not hypothetical) benefits to project
- Prioritize **simplicity**

Bike-shedding

Endlessly debating concrete but insignificant details

- Spinoff discussion for those with strong opinions
- Executive decision, then move on
- Prioritize addressing **largest technical risks**

Setting expectations

- Avoid misunderstandings, disappointment by setting clear expectations
  - Capture in writing
- Be consistent, firm when expectations are not met
  - Address acute issues ASAP
  - Address persistent, lingering issues at retrospectives
- Communicate often, ask for help
- Consider a "team charter"

Meetings

- High-bandwidth, high-visibility synchronous communication
- Expensive in terms of person-hours

- See John Cleese's *Meetings, Bloody Meetings*

Tips

- Have an agenda
- Circulate agenda + docs ahead of time
- Record action items
- Review previous action items
- Spin off detailed discussions if not relevant to all or not prepared for
- Avoid status reports
- Beware slides

Client Communication

- Respect client's time
  - Unless client requests chat-like communications, try to batch questions
  - Come prepared, stick to agenda, respect end time
- Prefer synchronous meetings
  - Higher bandwidth
  - Lower error rate
- Messages should go through point-of-contact
- Meetings should include most of development team
  - No one-on-one meetings
    - Need multiple viewpoints
    - Separate moderator from recorder

Retrospectives/postmortems

- Process improvement
- Retrospectives are regular
  - Like milestones, phased development – gates for evaluation, corrective action
- Postmortems come after an incident

Peer Review

Rubric will evaluate the following:

- Professionalism
    - Attendance/promptness
    - Preparation for meetings
    - Respectful interactions
- Quality of contributions
- Effective communication
    - Reports problems early
    - Communicates clearly
- Initiative
    - Contributes to brainstorming
    - Volunteers for tasks
- Healthy team dynamics
    - Raises and resolves conflicts
    - Trusts other members, helps them grow

Collaboration tools

- Version control
- Issue tracking
- Documentation (wiki?)
- Collaborative editing
- Iteration planning
- Project planning
- Code review
- Q&A
- Chat

Free tools

Cornell

- GitHub
  - Issue tracking & iteration planning
  - Wiki
  - Code review
- Google docs
  - Collaborative authoring & review
- Email
  - Communication

External

- Jira
  - Issue tracking & project planning
- Trello
  - Iteration planning
- Monday
  - Task tracking & project planning
- Slack
  - Chat

Assignment

- Form teams, select projects, fill out survey