

Software Engineering
CSE 335, Spring 2006

Software Engineering Lessons

Stephen Wagner

Michigan State University

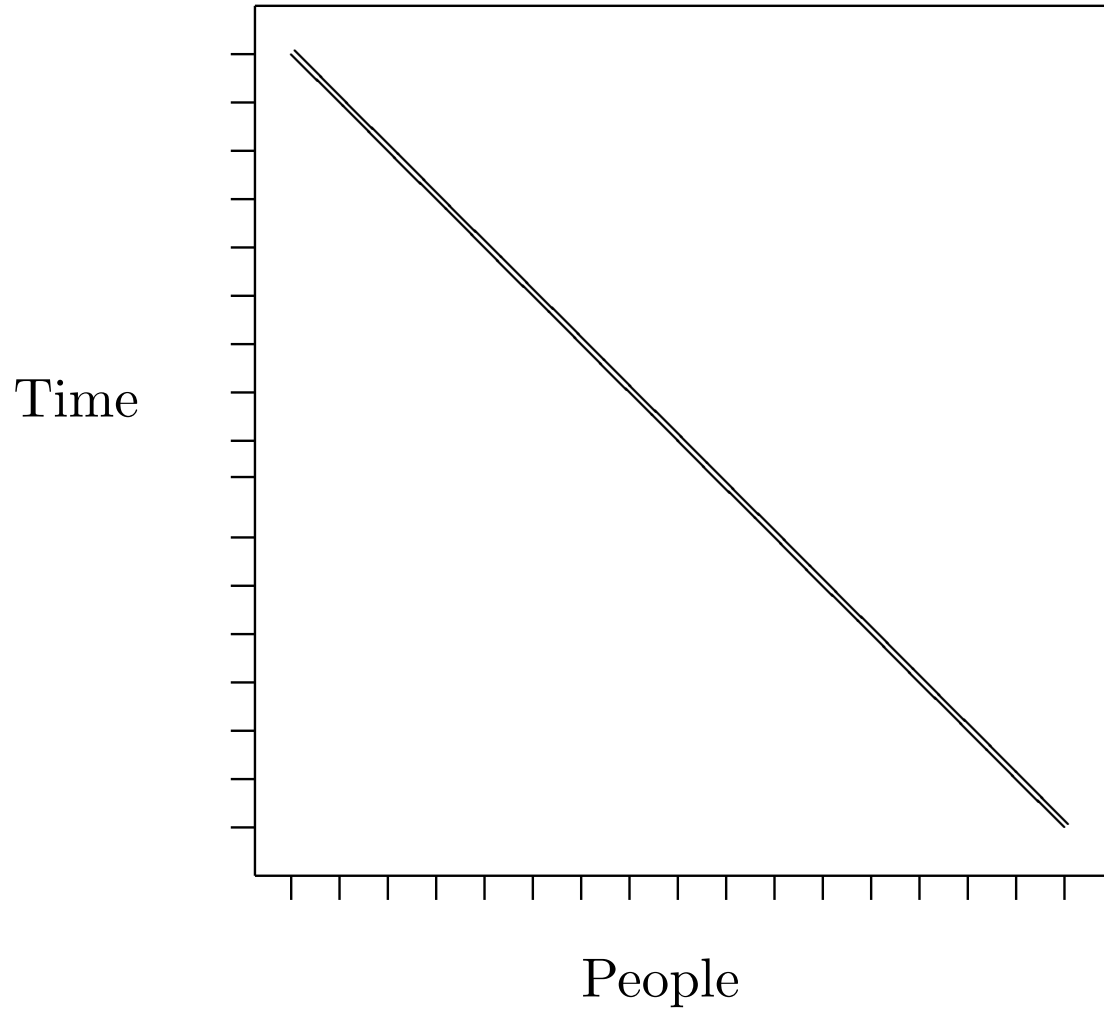
Software: The Process

- Major Theme: The quality of a software product is a function of the *process* use to develop the project.
- Software projects fail with alarming frequency
- Usually the culprit is lack of calendar time

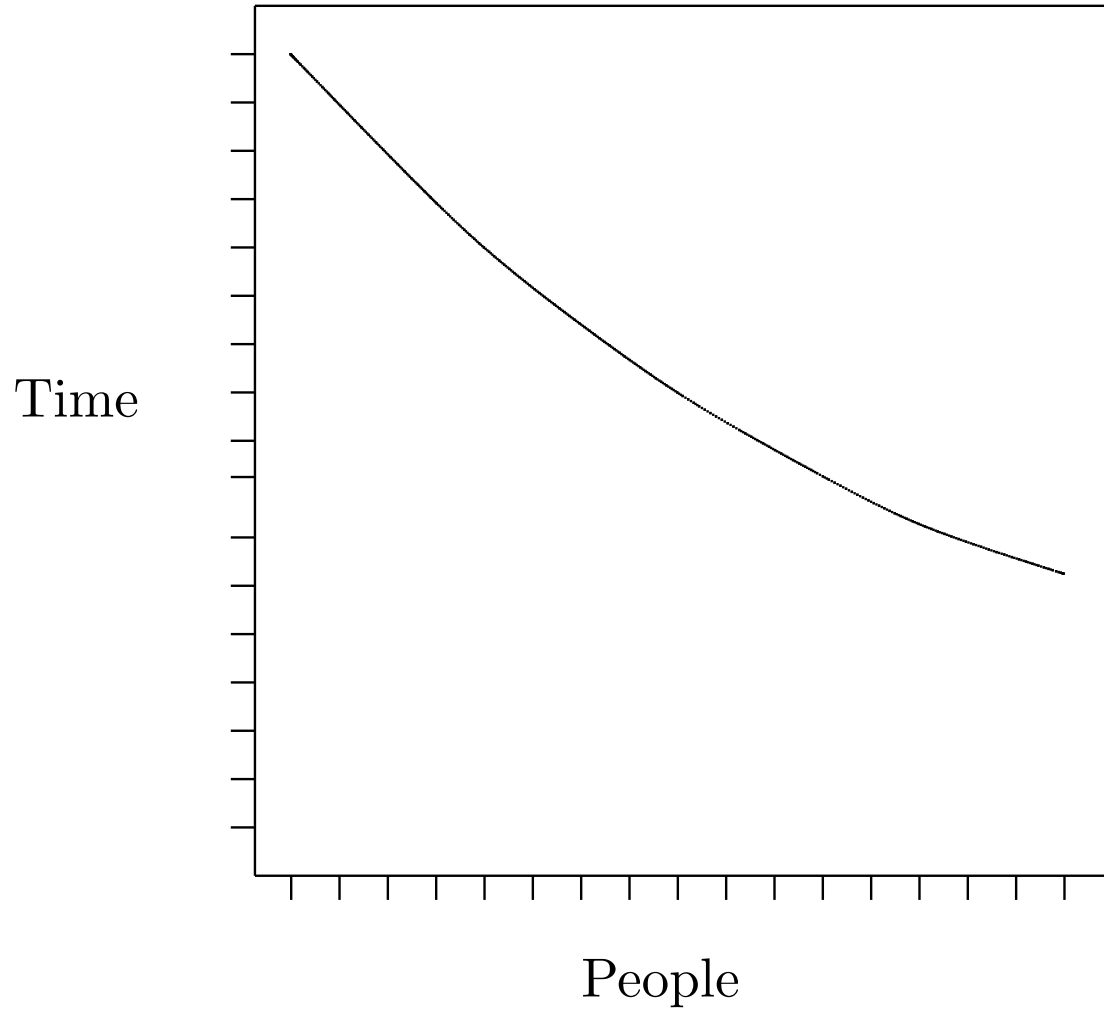
Mythical Man Month

- “More software projects have gone awry for lack of calendar time than for all other causes combined.”
- Brooks' Law: *Adding manpower to a late software project makes it later.*

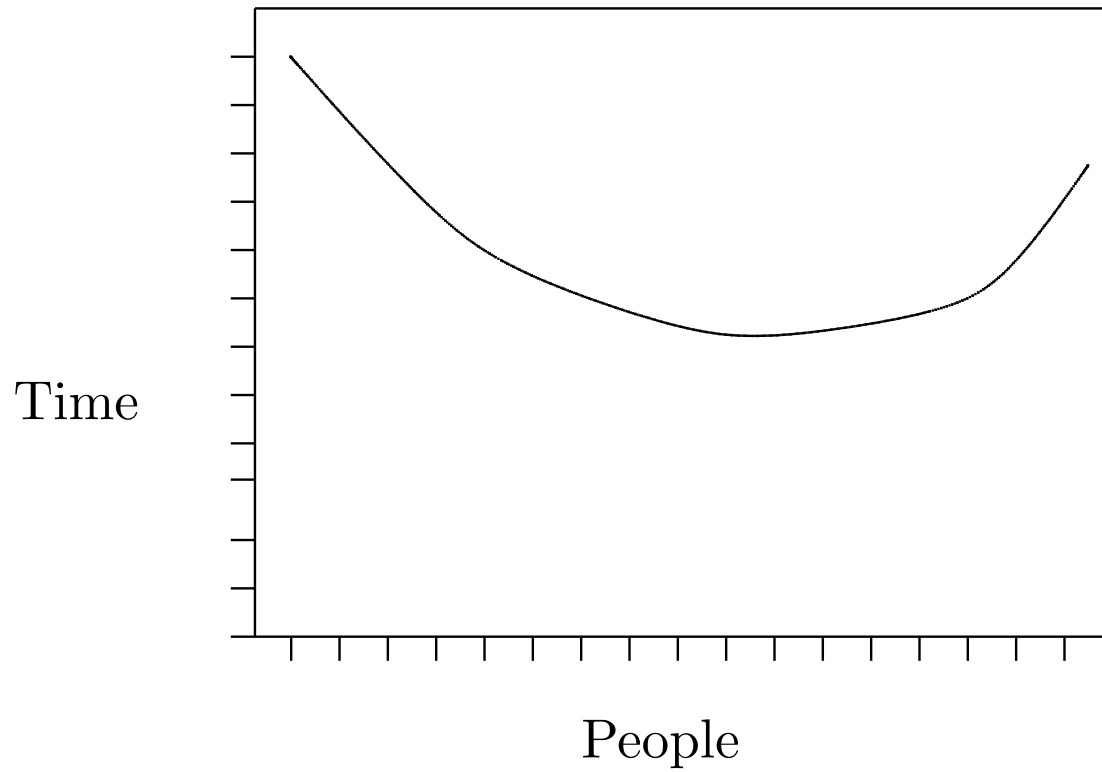
Ideal



Realistic



Depressingly Realistic



- Why is software problematic?

Why is time the enemy?

1. Estimation techniques poorly developed and based on assumption that all will go well
2. Estimation techniques confuse effort with progress
3. Owing to uncertainty about estimates, managers tend to “give in” to customer pressures
4. Schedule progress is poorly monitored
5. When schedules begin to slip, natural response is to add people, but this only makes matters worse

How should we estimate?

- Brook's rule of thumb for scheduling
 - 1/3 planning
 - 1/6 coding
 - 1/4 component test and early system test
 - 1/4 system test, all components
- Half of the time is budgeted to testing

Programmer productivity

- Lots of variance
 - Best programmers are 10x more productive than the worst
 - Code differently. 5:1 in speed and space usage
- Not necessarily correlated with experience
 - Some people “born programmers”?
- Teams: Smaller is better
 - Easier to coordinate a small number of minds
 - Teams should not exceed 10 people

Problem of Scale

- Logical conclusion: Use nothing but small teams
 - If a 200-person project has 25 managers who are the most competent programmers
 - Fire the 175 mediocre programmers and put the managers back to programming
- Problem: Consider a really large project
 - E.g., OS 360 took 5000 man years
 - 200-person team could do it in 25 years
 - Even small, talented team, would take 10 years
 - * 7x improvement for talent, 7x improvement for communication

Mill's Proposal

- Break a project into segments, each of which is tackled by a small *surgical team*
 - Surgeon (a.k.a. *chief programmer* writes all of the specs, all of the code, and all of the docs)
 - Other specialists support chief in various ways
- Benefits:
 - Few minds involved in design/construction
 - Many hands brought to bear

The Surgical Team

- The surgeon (chief programmer): responsible for specification, design, coding and documentation.
- The copilot: alter ego and understudy of the surgeon. Able to perform all of surgeon's tasks.
- The administrator: responsible for administrative tasks such as money
- The editor: responsible for final version of documentation. Surgeon writes the documentation, but editor reworks it.
- Several others including “the language lawyer” responsible for knowing the tricks of programming languages
- Reasonable for a medium size project, because surgeon is responsible for all the coding
- What about bigger projects?

Bigger Projects, Again

- For really large projects, you need lots of programmers
 - One person can only type so much in a lifetime
- How to use a large number of programmers effectively?

Big Projects, Again

- Architecture:
 - Defn: Complete and detailed specification of every observable aspect of the system
 - Identifies major components of a system and their protocols of interaction
- Principle: Large design projects require separating architectural from implementation effort:
 - Architecture defined by one mind (or small number of minds)
 - Leads to *conceptual integrity*
 - Each major component then implemented by a team
 - Programming cannot begin before architecture defined

Conceptual Integrity

- Most systems exhibit conceptual disunity
- Arises from (poor) separation of design task
- Better to omit unusual design “features” but instead reflect unified set of design ideas
 - Bad: good, but independent, uncoordinated ideas
 - Good: single, tied, coordinated set of design ideas
- Usually means splitting design from implementation

The Architect

- Democracy vs. Aristocracy
- “Too many cooks spoil the brew”
- Limit the number of people responsible for creative input
- The Architect(s) is (are) responsible for the design
- The architects are responsible for what happens, not how it happens
- To goal is to separate design and implementation

Complaints about Architects

- The specifications will be too rich in function, and will not reflect practical cost considerations
- The architects will get all the creative fun and shut out the inventiveness of the implementers
- The many implements will have to sit idly by while the architects work on the specifications

Architectural “self discipline”

- Problem separating design and implementation
 - What if architect becomes too extravagant?
 - i.e., it specifies something that cannot be built on time and under budget?
- Solution:
 - Architects must negotiate with “contractor”
 - Here contractors are chief programmers
 - Architect may need to scale down design or suggest cheaper way to implement features

The second-system effect

- Natural phenomenon:
 - A successful project followed by a failure
 - Usually results from over-confidence or lack of self-discipline on part of architect
- Qualities of “second systems”
 - Bulky, lots of frills
 - Lack of conceptual integrity
 - Powerful features with complex interactions

Passing the Word

- How to communicate information clearly and easily to a large team?
- Specifications
- Formal Definitions
- Meetings

Project workbook

- Structure imposed on the documents produced by a project
 - Objectives, external specifications, technical standards, etc.
 - Also administrative memoranda
- Technical prose enables the tracing of ideas through the lifetime of a project
- Problems
 - Timely updates
 - Size of workbook

Calling the shot

- Estimating how long a project takes is important
- The efficiency of individual programmers is a function of how often they interact with other programmers
 - Few Interactions 10000 instructions per year
 - Some Interactions 5000 instructions per year
 - Many Interactions 1500 instructions per year