Lecture 4
The Mythical Man-Month
(Part 1)

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 1999

Today's Lecture

- Discuss first four chapters of The Mythical Man-Month
  - The Tar Pit
  - The Mythical Man-Month
  - The Surgical Team
  - Aristocracy, Democracy, and System Design

Background of the Book

- Fred Brooks
  - 1964 Became the manager for Operating System/360 for IBM
    - Previous experience was in hardware design
      - 1956-1963
  - OS/360 "was late, took more memory than was planned, costs were several times the estimate, and it did not perform very well until several releases after the first."

Background, continued

- The book is the result of analyzing the OS/360 experience:
  - What were the management and technical lessons to be learned?
  - Why was the process different from the 360 hardware development effort?
- Brooks is now a professor at the University of North Carolina, Chapel Hill

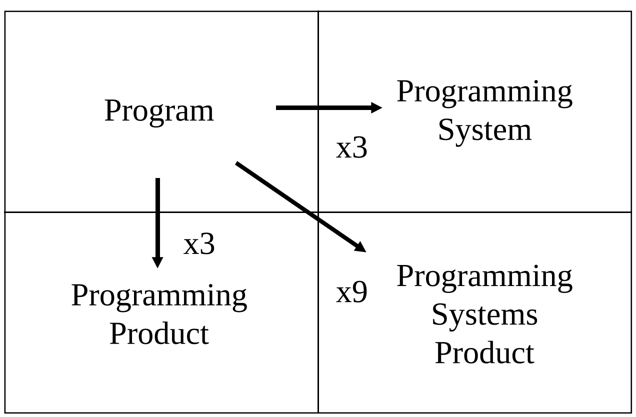
The Tar Pit

- Developing large systems is “sticky”
  - Projects emerge from the tar pit with running systems
    - But most missed goals, schedules, and budgets
    - “No one thing seems to cause the difficulty--any particular paw can be pulled away. But the accumulation of simultaneous and interacting factors brings slower and slower motion.”

The Tar Pit, continued

- The analogy is meant to convey that
  - It is hard to discern the nature of the problem(s) facing software development
- Brooks begins by examining the basis of software development
  - e.g. system programming

Evolution of a Program



What makes programming fun?

- Sheer joy of creation
- Pleasure of creating something useful to other people
- Creating (and solving) puzzles
- Life-Long Learning
- Working in a tractable medium
  - e.g. Software is malleable

What’s not so fun about programming?

- You have to be perfect!
- You are rarely in complete control of the project
- Design is fun; debugging is just work
- Testing takes too long!
- The program may be obsolete when finished!

Why are software project’s late?

- Estimating techniques are poorly developed
- Our techniques confuse effort with progress
  - The Mythical Man-Month
- Since we are uncertain of our estimates, we don’t stick to them!
- Progress is poorly monitored!
- When slippage is recognized, we add people
  - “Like adding gasoline to a fire!”

Optimism

- “All programmers are optimists!”
  - “All will go well” with the project
    - Thus we don’t plan for slippage!
  - However, with the sequential nature of our tasks, the chance is small that all will go well!
- One reason for optimism is the nature of creativity
  - idea, implementation, and interaction
  - The medium of creation constrains our ideas
    - In software, the medium is infinitely tractable, we thus expect few problems in implementation, leading to our optimism

The Mythical Man-Month

- Cost does indeed vary as the product of the number of men and the number of months
  - Progress does not!
  - The unit of the man-month implies that men and months are interchangeable
    - However, this is only true when a task can be partitioned among many workers with no communication among them!

The Man-Month, continued

- When a task is sequential, more effort has no effect on the schedule
  - “The bearing of a child takes nine months, no matter how many women are assigned!”
  - Many tasks in software engineering have sequential constraints!

The Man-Month, continued

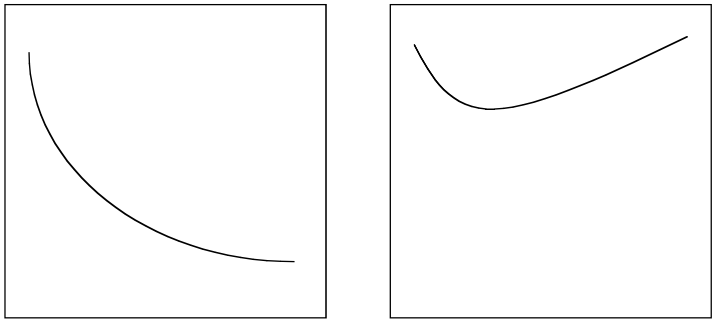
- Most tasks require communication among workers
- communication consists of
  - training
  - sharing information (intercommunication)
- Training affects effort at worst linearly
- Intercommunication adds n(n-1)/2 to effort
  - if each worker must communicate with every other worker

Intercommunication Effort

2 workers	1 path
3	3 paths
4	6 paths
5	10 paths
6	15 paths
7	24 paths

Comparison Graphs

“Adding more people then lengthens, not shortens, the schedule!”



Scheduling

- Brook's rule of thumb
  - 1/3 planning
  - 1/6 coding
  - 1/4 component test
  - 1/4 system test
- More time devoted to planning, half to testing!
- In looking at other projects, Brooks found that few planned for 50% testing, but most spent 50% of their time testing!
  - Many of these projects were on schedule until testing began!

The Surgical Team (Chapter 3)

- Or
  - How should the development team be arranged?
- The problem
  - Good programmers are much better than poor programmers
    - typically 10 times better in productivity
    - typically 5 times better in terms of program elegance

The dilemma of team size

- Consider the following example
  - 200-person project with 25 experienced managers
  - Previous slide argues for firing the 175 workers and use the 25 managers as the team!
    - However, this is still bigger than “the ideal” small team size of 10 people (general consensus)
  - However, the original team was too small to tackle large systems
    - OS/360 had over 1000 people working on it; consumed 5000 man-years of design, construction, and documentation!

Two needs to be reconciled

- For efficiency and conceptual integrity
  - a small team is preferred
- To tackle large systems
  - considerable resources are needed
- One solution
  - Harlan Mill's Surgical Team approach
    - One person performs the work
      - all others perform support tasks

The Proposed Team

- The surgeon
  - The chief programmer
- The co-pilot
  - Like the surgeon but less experienced
- The administrator
  - Relieves the surgeon of administrative tasks
- The editor
  - Proof-edits documentation
- Two secretaries
  - Support admin and editor
- The program clerk
  - Probably obsolete today
- The toolsmith
  - Supports the work of the surgeon
- The tester
- The language lawyer

How is this different?

- Normally, work is divided equally
  - Now only surgeon and copilot divide the work
- Normally, each person has equal say
  - The surgeon is the absolute authority
- Note communication paths are reduced
  - Normally 10 people => 45 paths
  - Surgical Team => at most 13 (See Fig. 3-1.)

How does this scale?

- Reconsider the 200 person team
  - Communication paths => 19,900!
- Create 20, ten-person surgical teams
- Now, only 20 surgeons must work together
  - 20 people => 190 paths
    - Two orders of magnitude less!
- Key problem is ensuring conceptual integrity of the design

Conceptual Integrity

- Brooks example => Cathedrals
  - Many cathedrals consist of contrasting design ideas
  - The Reims Cathedral was the result of eight generations of builders repressing their own ideas and desires to build a cathedral that embodies the key design elements of the original architect!
- With respect to software
  - Design by too many people results in conceptual disunity of a system which makes the program hard to understand and use.

Conceptual Integrity

- Brooks considers it the most important consideration in system design
  - Better to leave functionality out of a system if it causes the conceptual integrity of the design to break
- Questions
  - How is conceptual integrity achieved?
  - Are system architects raised to the level of aristocracy?
  - How does one keep architects' designs realistic?
  - How does one ensure that a design is correctly implemented?

Function vs. Complexity

- The key test to a system's design is the ratio of functionality to conceptual complexity
  - Ease-of-use is enhanced only if the functionality provides more power than it takes to learn (and remember) how to use it in the first place!
  - Neither function or simplicity alone is good enough
    - OS/360 had lots of functionality
    - PDP-10 has lots of simplicity
    - Both reached only half of the target!
- These can be achieved with conceptual integrity!

Architects as Aristocrats

- Conceptual Integrity requires that the design be the product of one mind
- The architect (or surgeon) has ultimate authority (and ultimate responsibility)!
  - Does this imply too much power for the architects?
    - In one sense, yes, but ease-of-use of a system comes from conceptual integrity!
    - In another sense, no, the architect sets the structure of the system, developers can then be creative in how the system is implemented!