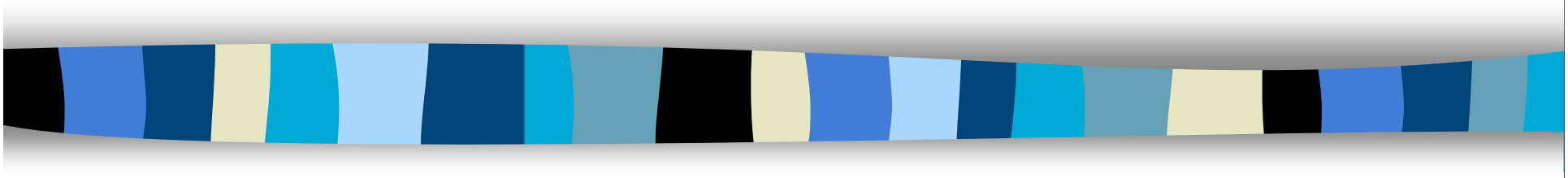
The Mythical Man Month

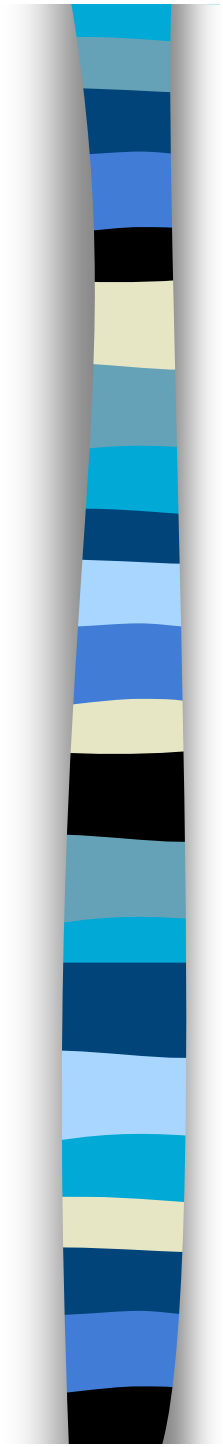


COMP340

Dr. Paul Watters
<pwatters@ics.mq.edu.au>

This Week

- Discussion of productivity in software engineering
  - With a focus on team structure and composition
  - Issues first raised by Fred Brooks in 1975
    - Do his assertions still hold true?
    - How are they related to agile and XP?
- Readings
    - No Silver Bullet: Essence and Accidents of Software Engineering (IEEE Computer, 1987)
    - F.P. Brooks (1975). *The Mythical Man Month*. Addison Wesley Longman.

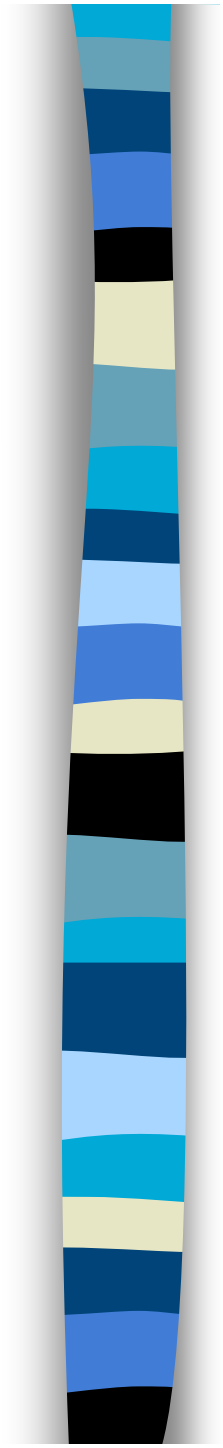


The Tar Pit

- Basic questions
  - Why is software engineering so hard/expensive/complex?
    - Why do large teams only produce comparatively small amounts of code compared to “garage” duos?
- Large systems programming…
  - …is like a tar pit because “great and powerful beasts have thrashed violently in it”
    - Few projects meet schedules, budgets, goals - why?

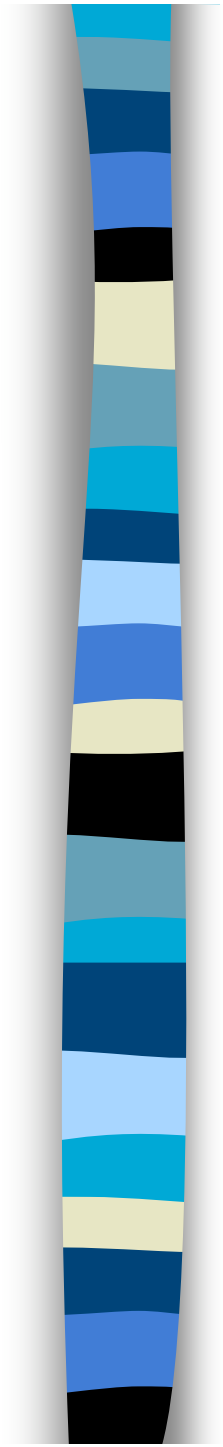
Programming, Systems, Products

- Accumulation of simultaneous and interacting factors reduces progress
  - Hardware x operating system x software environment x programming language x applications (1..*n*)
    - Writing individual programs is easy
    - Not a Programming Systems Product
    - Programming System -> Interfaces + Integration
    - Programmig Product -> Generationalisation + testing + documentation + maintenance



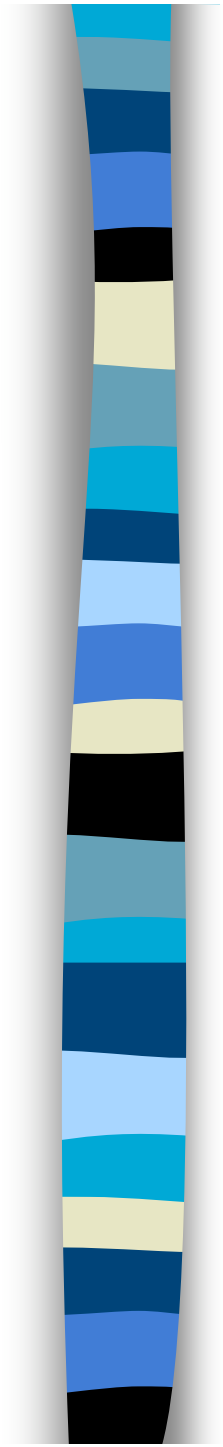
Programming Product

- Can be run, tested, extended by anybody
  - Written in a generalised fashion
    - Can be used with many different types of data
  - Range and form of inputs catered for
  - Thoroughly tested across inputs ranges
    - Develop bank of test cases with all boundary conditions
  - Documentation - use it, fix it, extend it
    - Programming product - at least 3x the cost of a debugged program



Programming System

- A program is just one *component* in a larger system
  - Interacting components, orchestrated and disciplined for large tasks
    - Input/output syntax and semantics conforms to well-defined interfaces
    - Requirements for memory, CPU, I/O are well-known
    - Must be tested with other system components
      - Exponential growth in test cases
      - Subtle bugs arise from low-level interactions of debugged components
      - At least 3x the cost of a debugged program

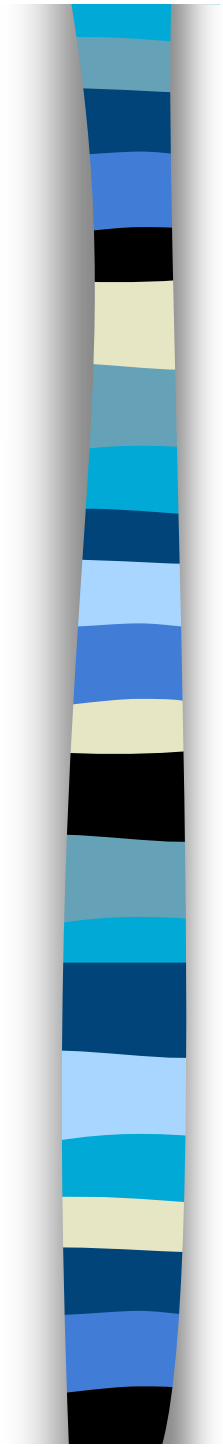


Programming Systems Product

- Has all requirements of Programming Product and Programming System
  - But takes at least 9 times the effort of a simple debugged program
  - But is infinitely more useful than a simple debugged program
    - Enjoyable to make, useful, complex, tractable, non-repetitive
    - But programming systems products is hard - requires perfection, others set objectives, dependence on others, tedious debugging, so quickly obsolete...

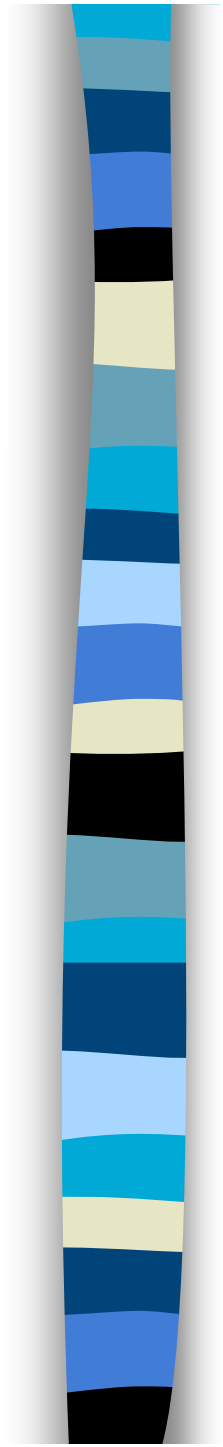
Why is so much effort required?

- Estimating techniques are poorly developed
  - Often assume nothing will go wrong
  - Effort is confounded with progress
    - Assumption is that humans and months are interchangeable
    - Uncertainty in time estimation is not accepted by management
    - Schedule progress is poorly monitored
    - When the schedule slips, more people are added
    - Conjecture: Adding people makes things worse



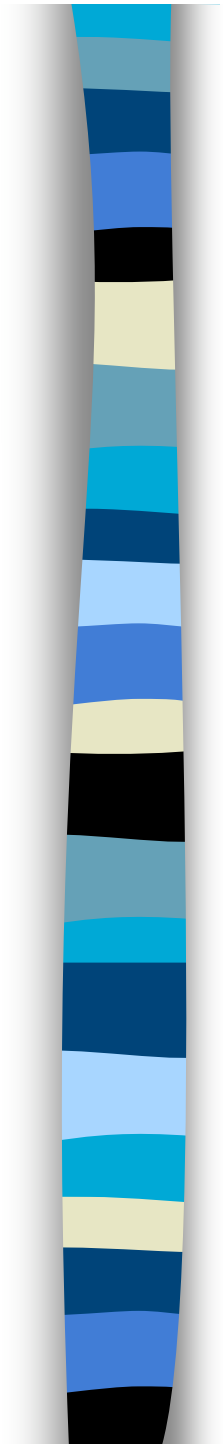
Fallacies

- Optimism
  - All programmers are optimists
    - “This time it will surely run”
    - “I just found the last bug”
    - Reinforced by previous success? Short memories? Folly of youth?
    - Assumption that all will go well
    - Incompleteness and inconsistency of ideas only becomes apparent during implementation!
    - “Working out” and experimentation are indispensible tools for creative disciplines



Optimism

- Assumption
  - Each task will only take as long as it "ought" to take
    - Physical limitations of medium may prevent execution - but thoughts are very tractable - hence our optimism
    - Inadequacy of ideas in the first place - may need revision
      - Hence appropriateness of XP-style methodologies in constant revision/iteration
      - Waterfall model still dominant in SE textbooks!
    - Given the number of complex and interacting tasks, probability that nothing will fail is nil



The Man Month

- Basic unit of time estimation is the “man month”
  - Cost varies as a product of number of workers and the number of months
    - Progress does not!
    - Thus, you can’t interchange the size of the workforce and the length of the project
      - Exception: Workers and months can be interchanged when a task is perfectly partitionable, can be performed with no communication between workers, and requires no learning to complete
      - If a task cannot be partitioned because of sequential constraints, adding more people does not reduce time

Examples

- Cotton picking
  - One worker picks 10 bales per day
  - Ten workers can pick 100 bales per day
  - Adding 9 workers has a tenfold increase in productivity as a function of time
- Childbirth
  - One woman takes 9 months to deliver
  - Ten women task 9 months to deliver
  - Adding 9 women has no increase in productivity as a function of time

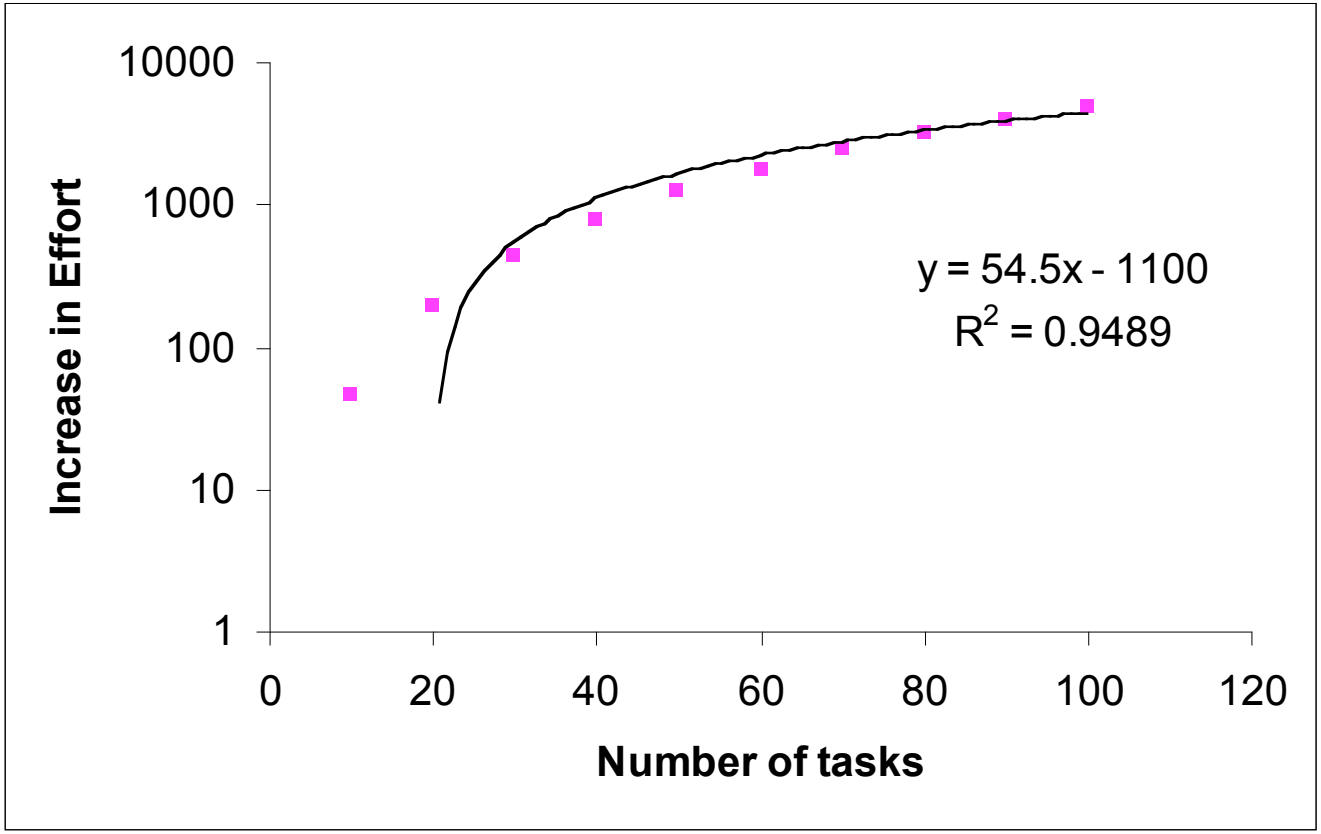
Cotton picking or childbirth?

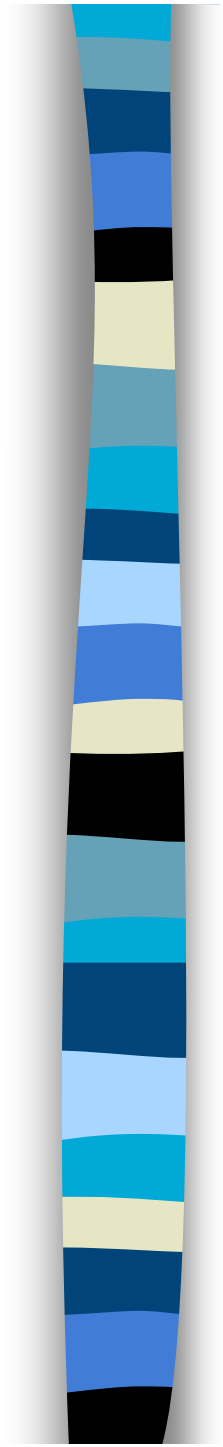
- Is programming more like cotton picking or childbirth?
  - In natural state, more like childbirth
    - Sequential nature of debugging
    - Even if task is divided into subtasks, communication between workers is required
    - Effort of communication delays progress
    - Effort of non-partitionable training delays progress
      - Training in general technologies
      - Training in goals, strategy, plan, current system and development state
      - Added effort is linear

Intercommunication

- If each subtask is separately co-ordinated with each other subtask, effort increase is n(n-1)/2
  - 2 subtasks: effort increase ~ 1
    - 3 subtasks: effort increase ~ 3
    - 10 subtasks: effort increase ~ 45
  - Hence, reported productivity of large commercial SE teams of only 1,000 statements per year

Large Projects





Possible Solutions

- Don’t partition tasks that require lots of intercommunication
  - Partition tasks so that they are as independent as possible
    - Hire people with the requisite skills in the technology concerned
    - Can’t escape goals, strategy, plan, current system and development state
    - But using components, interfaces and layers, can drastically reduce amount of interaction required between developers
      - Maximise independence of workers/tasks
      - What agile/XP techniques can achieve this?

Main Conjecture

- Adding people to projects increases the time to complete the project
  - Taking everything into account, the schedule will almost certainly have to lengthen
  - Is this still true with modern systems? Architectures? The focus on design?
    - E.g., CORBA systems - agree on interface up-front, then client/server code developed independently
    - Regular reviews and revisions?
    - Effect of interface changes?

Systems Testing

- Greatest contributions to schedule
  - Component debugging
  - Systems testing
    - Time required is hard to predict
    - Depends on subtlety and number of bugs
    - Number of actual bugs always greater than predicted
  - Rule of thumb:
    - 1/3 planning
    - 1/6 coding
    - 1/4 component test
    - 1/4 system test

Scheduling

- Coding is usually over-emphasised relative to testing and planning
  - A detailed and solid specification is critical
    - Assist with the development of independent subtasks
    - Planning time does not include investigation of new techniques or research
    - Debugging time is under-estimated in most schedules
      - Should test regularly - test-driven development?
    - Coding time is easiest to predict

Effect of Delays

- Projects that allocate proper testing time usually on budget and on time
  - Including only coding time in estimates will result in delays
    - Delays can have budgetary consequences
    - Psychological issues - demotivation
    - Cascading of issues throughout the business that relied on the software being available on a specific date/time
    - “Gutless estimation” - task urgency only influences scheduled **not** actual completion
      - Be realistic and defend actual estimates for testing

Regenerative Schedule Disasters

- When behind, add more people…
  - Typical response - may help or make situation worse!
  - If deadlines missed
    - Add more people to make up the time
    - Extend the taskdeadline by increasing estimates
    - Reschedule
    - Trim the task (formally or informally)
  - Problems with adding more people
    - Training, partitioning effects, slipping, add more people again...

Scenario

- 4-month project, 3 people, milestones at each month A B C D (12 mm)
  - What happens if you haven't reached A by the end of two months?
    - Solution 1: assume task still completed on time, assume A mis-estimated. 9 mm remain. Add 2 extra people ~ 10 mm.
    - Solution 2: assume task still completed on time, assume all mis-estimated. 18 mm remain. Add 6 extra people ~ 18 mm.
    - Solution 3: Reschedule (if allowed).
    - Solution 4: Trim the task when you slip.

Analysis: Solution 1

- Disastrous!
  - Assume 2 new recruits are competent in general technology
    - Training not accounted for in new schedule
      - 3 mm then wasted in 1 months training (2 new, 1 experienced)
      - Repartitioning required - now 3 people not 5
      - Result: at the end of the month, now only 2 mm devoted to project, not 5. **@3 months~7mm remain**
    - If only 1 month required for training, then completion is pushed out by only 3 mm.
      - If longer, then 3*m* mm required, where *m* is months of training
      - What if more than 1 is required for training?

Analysis: Solution 2

- Worse!
  - Training, repartitioning and system testing effects now for 6 new staff - not just 2!
    - All of these effects unaccounted for - how many staff required to train 6 new ones? Repartition for 9 staff not 3? Probably zero progress by the end of the third month!
  - Brooke's Law
    - *Adding people to a late software project makes it later*
    - Project length depends on sequential constraints; Staffing depends on number of independent subtasks / good design

Analysis: Solutions 3&4

- Golden rules
  - If you have to reschedule, try to do it once
  - Allow enough time to get the job done thoroughly with the present team
  - Use time estimates derived from actual work
  - Best to use less people and get more months
    - Getting more months is not always possible
    - Lack of calendar time is a significant problem in programming
- Try to get compromise on requirements as the product evolves
  - And the results of experimentation and non-functional tests are known…
  - If you get enough time allocated to planning, then coming up with a good set of needs/wants should be possible with stakeholders

The Big Questions

- The big questions are…
  - How do we get the most out of the teams we have?
  - How do we do really big projects with large staff numbers?
    - Can't have the ideal of small, sharp team doing everything
  - How do we deal with ~10x variability in coding performance? (Sackman et al)
    - No correlation between experience and performance
  - How to reduce intercommunications and increase task independence?
    - Do agile methods and XP methodologies help?

What's a large system?

- IBM OS/360
  - Had 1,000 employees concurrently on the project over 4 calendar years
  - Programmers, writers, operators, clerks, managers, secretaries etc.
  - 5,000 man years went into design, construction and documentation
  - With a 200-person team, only 25 years to complete…!
  - How to improve?

Large Project Teams

- Assume
  - 10 person team
  - 7x as productive as OS/360 team
  - 7x productivity improvement from reduced communications (small team size)
  - Same team on the job
  - Then 5000/(10x7x7)=10 man-years
  - Sounds great – but product will be obsolete before it is complete! Hence, most people stick to brute force…

Harlan Mills Proposal

- Each segment of a large project be undertaken by a team
  - Team organised like a surgical team with specific roles
  - One “surgeon” doing the butchery; everyone else in support
  - Few minds involved in design
  - Highly segmented/independent tasks
  - Minimal communication during operations; only ever 1-1

The Roles

- Surgeon
  - Chief programmer
  - Defines all functional and non-functional specifications
  - Designs, codes and tests program, writes doco
  - Very experienced, very talented person
  - Considerable applications and systems knowledge in relevant field
  - Uses computer system to implement and test the code directly

The Roles

- Copilot
  - Backup for the surgeon
  - Can do everything the surgeon does
  - But has less experience
  - Shares in design as a thinker, discussant and evaluator
  - Surgeon tries out ideas on her, but doesn't have to accept advice
  - Copilot communicates with team; relays advice to surgeon
  - Knows all code intimately; researches alternative designs
  - Bus accident insurance
  - May write code but not responsible for it

The Roles

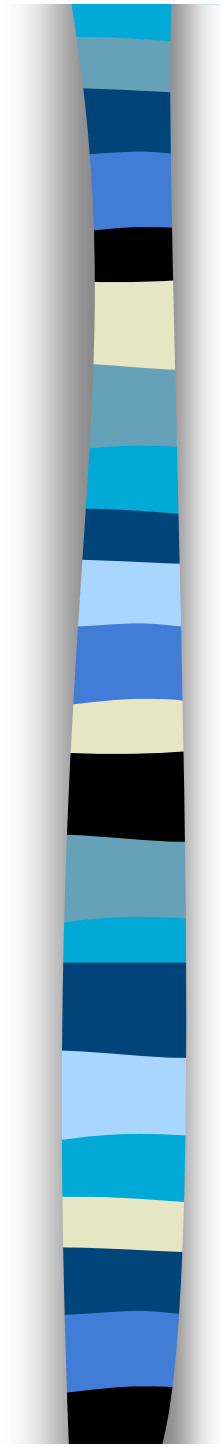
- Administrator
  - Surgeon is the boss…BUT must spend no time dealing with personnel, space, money, machines, bureaucracy
  - This is the role of the administrator
  - Liaises with stakeholders
  - May be full-time if significant legal, contractual, reporting requirements
  - One administrator may serve multiple teams

The Roles

- Editor
  - Responsible for writing doco
  - Generates external and internal descriptions
  - Takes draft provided by surgeon and refines, clarifies, amplifies
  - Criticises and reworks
  - Manages versioning
  - Inserts references and bibliographies
  - Responsible for production and publication

The Roles

- Two Secretaries
  - Surgeon and administrator both have a secretary
  - Deal with correspondence and non-product files
- Program Clerk
  - Responsible for maintaining technical records of team in programming-product library
  - Trained as secretary
  - Logs all input/output for filing and indexing
  - Archiving and source management
  - Makes code available for the team to review
  - Configuration management?



The Roles

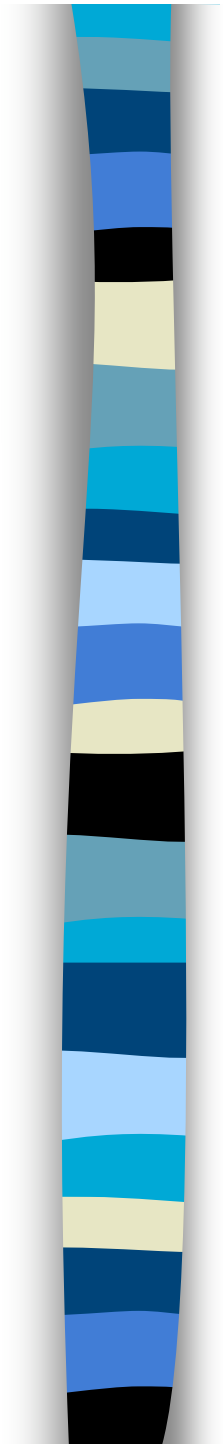
- Toolsmith
  - Available to build customised utilities on request of surgeon
  - Can be called on to write procedures, libraries, macros etc
  - Any special tools for this specific project (not COTS)
- Tester
  - Devises test cases and data from functional specification
  - Tests releases in an adversarial fashion
  - Responsible for building test harnesses
  - Plans test sequences for unit and regression tests
- Language Lawyer
  - Experts in syntax and semantics of languages used in the project
  - Knows neat and efficient ways of doing things
  - Encyclopaedic knowledge of APIs and libraries

Team Processes

- One creative genius drives team production
  - But supporting roles are no less important
  - Surgeon focuses on designing and developing the system
  - Freed up from supporting but essential activities
  - Radically different from projects where each person does their own design development and testing
  - Role specialisation is critical to gaining the increases in productivity over the “average” programmer
  - Surgeon has unilateral control over decisions
    - Conforms to social psychology studies on team productivity showing authoritarian models are most productive if less satisfying than democratic or laissez-faire models
    - Model supported by Baker’s 1972 study
    - How would you structure a team based on agile/XP methodology?

Remaining Problem

- A highly performing team of 10 would still take 10 years
  - So how do we scale up this sort of structure?
  - Must divide larger project into sub-projects
  - With 200 people, only have 20 creative minds to co-ordinate
  - Requires an overall system architect who has authority over surgeons
  - Architectural responsibility is separate from implementation responsibility
  - How to maintain conceptual integrity and maintain unity?
  - How to prevent surgeons getting too creative and veering off course?
  - How to keep team members happy if they are not the boss?



Summary

- Structure of teams is a contentious issue
  - How to ensure maximum productivity?
  - How to ensure that software is delivered on time?
  - How to avoid pitfalls when restructuring teams to meet deadlines?
  - How can we use agile/XP to reduce communications between team members?