Analysis vs. management

The requirements say what the users *want,* in their terms

- Functional, non-functional, resources, costs

The specification says what the users will *get*

- Addresses each identified requirements
- Does not address things like cost and timeliness

The results of analysis must be reified into software by an engineering team – and that takes planning

- Project teams
- Project management

Project teams

Large projects are necessarily performed by teams of engineers

- "The work" has to be split between team members
- Different tasks, different people, different responsibilities, different abilities and specialities
- The composition and dynamics of a team often make the difference between success and failure

This lecture is a basic overview of some common team structures used for software engineering

Why program in teams?

Most applications are much too big to tackle alone

- Too complex to analyse, too big to design, too much programming
- One person doesn't have the monopoly on good ideas – however talented they are

Project teams

- A collection of people bringing different talents to bear on a common overall goal
- ✓ More hands, more ideas, more diverse expertise
- × More communication, more dissent, more misunderstandings

The problem is to maximise the benefits whilst minimising the downside risk

Teamwork .. trade-offs

- ✓ Write more code, do more analysis
- ✓ Different expertise
- ✓ More ideas
- ✓ More parallelism
- ✓ Easier to pursue alternatives
- ✓ Can divide responsibility for different areas
- ✓ Can have a well-defined team structure – everyone knows their place
- ✓ Can have a sense of common vision and common purpose

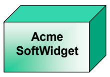
- × Diversity can get out of hand
- × Less focused
- × More scope for dissent
- × More communication
- × Harder to converge on a single plan of action
- × Have to divide authority too
- × Can leads to conflicts if people don't like their place
- × Not everyone may like where the team's going

The team's work

Requirements Specification
Analysts – good listeners, identify with customers' needs

Design
Good view of the big picture, good with abstractions

Maintenance
Respect for the users and the product



Coding

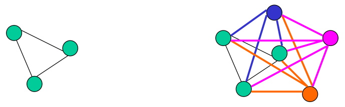
Deployment
Command of the vagaries of operating systems

Testing
Perverse delight in finding faults

Documentation
Good writing skills

Integration
Good sense of balance

What happens as teams grow?



3 agents
3 channels

6 agents
15 channels

Increasing scope for confusion

- Decisions aren't fully shared, people aren't notified of changes, …
- Not everyone understands the issues or ramifications of a decision
- Difficult to achieve unanimity of design or coding styles
- "Experts" will disagree on the "right" approach

The dilemma

Everyone agrees that small, sharp teams are better whenever possible

The problem is, that isn't very often

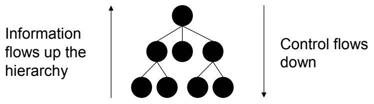
If we want to build large systems, we have to balance the pros and cons

- Add people to cope with the increased scale of work
- Add management to cope with the complexity and diversity

How to find a good, workable dynamic?

Management – a quick guide

Management is mostly about hierarchy



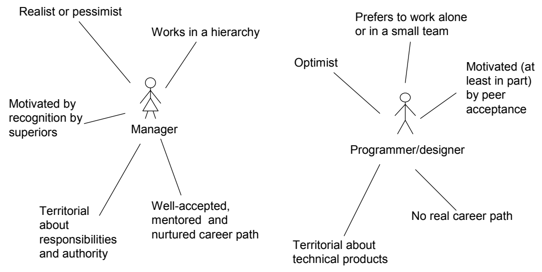
In general no employee should report to two managers

Responsibilities should be clearly divided and defined

Authority follows responsibility

- Only accept responsibility for something you've been given the authority to accomplish

The right stuff


Realist or pessimist
Works in a hierarchy
Motivated by recognition by superiors
Manager
Territorial about responsibilities and authority
Well-accepted, mentored and nurtured career path
Prefers to work alone or in a small team
Optimist
Motivated (at least in part) by peer acceptance
Programmer/designer
Territorial about technical products
No real career path

Approaches

"Democratic" teams

"Classical" chief programmer teams

"Surgical" teams

"Modern-variant" chief programmer teams

Scaling-up to larger projects

Most team studies focus on implementation, but the ideas apply equally to whole-cycle development

Democratic teams

Avoid territoriality by making programming *egoless*
- Encourage suggestions and debugging by others
- Foster a group identity and ethos

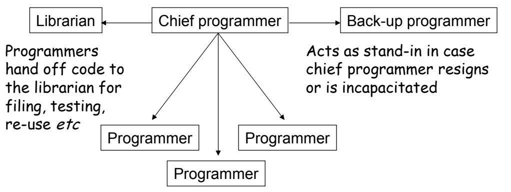
No managers, no hierarchy
- Makes for difficult promotions
- Not everyone is a crack programmer
- Lots of experts, no-one to decide in intractable situations

Can be excellent – or a recipe for chaos – depending on exactly the project and the people involved

Now also known as *extreme programming*
- http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap

Chief programmer teams – take 1

Place one experienced programmer in charge of a small team of less experienced programmers


Librarian
Chief programmer
Back-up programmer
Programmers hand off code to the librarian for filing, testing, re-use *etc*
Acts as stand-in in case chief programmer resigns or is incapacitated
Programmer
Programmer
Programmer

Jim Coplien's "truck number": how many catastrophes can befall a team before it folds?

Analysis

Good model for small projects, but has some major disadvantages

- Chief programmer is both programmer *and* manager – such people are rare and have conflicting motivations
- …but nowhere near as rare as people with the talent and small egos to be back-up programmers
- Can only directly manage so many programmers – say about five
- Direct management - the idea of personal success or failure - can result in damaging defensive attitudes (especially towards bugs)

Want to strike a balance

- The positive attitude to fault-finding of democratic teams
- The structure and leadership of chief programmer teams
- …and something less idealistic than both, perhaps

Brooks and Mills' "surgical team" - 1

Programmer productivities vary dramatically

- Empirical data suggests a factor of five to ten *at least*

Teams generally get less efficient as they grow

- Prefer small sharp teams, but they're not up to really big projects

Brooks and Mills hypothesise that the basic problem is that teams are pretty much undifferentiated

- Everyone's either a programmer or a manager
- Analogy to surgery – one surgeon, some nurses, other specialists
- Different areas of responsibility and expertise
- Differences in judgement settled by the surgeon when they're critical

Quoted in Brooks, *The mythical man-month*, Addison Wesley (1995).

Brooks and Mills' "surgical team" - 2

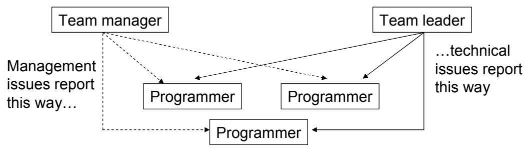
The surgeon	Chief programmer and dictator, with ultimate responsibility and authority
The co-pilot	Less experienced version of the surgeon, handles alternative design strategies
The administrator	Handles all personnel and other admin
The editor	Re-crafts documentation written by the surgeon and others
Two secretaries	
The program clerk	Handles all machine- and human-readable files – visible to all
The toolsmith	Services, maintains and creates coding and other tools
The tester	Stress-tests all code, checks against specs
The language lawyer	Responsible for detailed algorithm design

Note: ratio of "specialists" to "programmers"

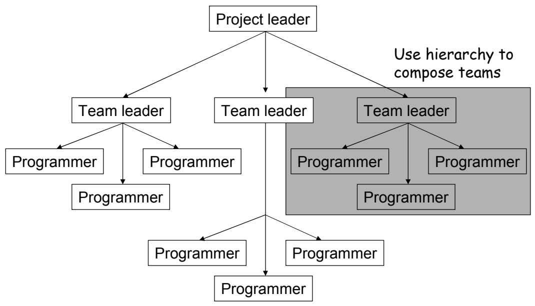
Chief programmer teams – take 2

Part of the problem seems to be that we want a manager *and* a programmer rolled into one

- Split the responsibility – team leader (technical details) and team manager (administrative details)
- Need to clearly delineate their responsibilities to avoid splitting each members' reporting hierarchy
- …but that still leaves the problem of which person has final control

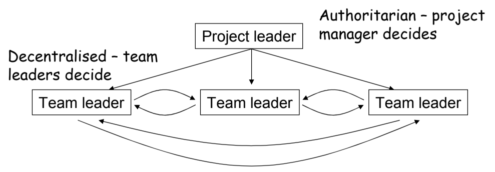

Team manager
Team leader
Management issues report this way…
…technical issues report this way
Programmer
Programmer
Programmer

Larger teams



Decision making

Adding more allegedly equal managers means having to decide how to make final decisions


Project leader
Authoritarian – project manager decides
Decentralised – team leaders decide
Team leader
Team leader
Team leader

Democracy has its problems even here – but with fewer people it's more practical

Remaining problems

…and there are no really good, large-scale solutions

Keeping everyone informed

- Project meetings – round-tables to share progress and problems
- Document repositories – document all decisions (a good idea anyway!) and make them available to the whole team

Keeping everyone involved

- Nurture a shared vision
- Foster a culture of openness – get complaints out into the open
- Make all benefits accrue to the team, not to the individual

Personality clashes

- Shift conflicting people into different areas, away from each other
- Appeal to their professionalism to see the job through
- Fire the less valuable one (or both)

Summary

Teams co-ordinate the actions of a group of people towards a single goal

Increasing project size implies increasing team size

- More work possible
- More confusion possible

Adding management layers helps manage the complexity

- Divide responsibility and authority

Managing a team in the broadest sense – project management – is what we'll turn to next