Project management

In the widest sense, the techniques needed to follow a lifecycle model successfully

- Contracts
- Estimation
- Planning
- Risk assessment and management
- Monitoring and control

In the next lectures we will look at each of these aspects

What is a project?

“A project is a temporary endeavour undertaken to create a unique product or service”

Project Examples

- Building a Bridge
- Developing a new or modified information processing system
- Delivering an information processing service, for example as a Web Service
- Eradicating the millennium bug

Software contracts

The business-level agreement with the customer

The process

- Customer issues a *tender* document detailing roughly what’s wanted and under what funding regime
- Companies *bid* for the tender, usually making a presentation of their past successes and strengths
- Customer picks the one judged to be most likely to succeed

Items mandated by a contract

- Specification, schedule, maintenance
- Payments and ownership - is the software leased or bought? Who owns the ideas (IPR) after completion?
- Practices – software methodology, ISO9000 certification, …

Types of contract

What they contract for

- Product – supply some (software) product at some cost on some schedule
- Service-level agreement (SLA) – provide some service according to some cost or quality constraints

How they pay for it

- Fixed price – an agreed price for the goods/service
- Time and materials – an agreed price per person/month *etc*
- Costs-plus – what it costs to build plus a possible bonus
- Fixed price per unit – an agreed price for each part of a multi-part delivery

Penalty clauses

- Supplier forgoes some payment if late
- Supplier indemnifies customer against incorrect operation

Product

"Provide an accounting system for us
- Must meet all current accounting practices
- Must cost less than €1,000,000

Payment models
- Fixed price – "thanks, here is €1,000,000"
- Time and materials – "send us your time sheets and hardware invoices at the end of the year, and we will pay them"
- Cost-plus – "send us your bills and we will add 10% as long as you deliver on time"
- Fixed price per unit – "thanks, here is €100,000 for the first sub-system"

Service-level agreements

"Maintain a database available over the web"
- No more than 1% unscheduled down-time during working hours
- All queries answered within 5s

Payment models
- Fixed price – "here is €1,000,000 for this year – get on with it"
- Time and materials – "send us your time sheets and hardware invoices at the end of the year, and we will pay them"
- Cost-plus – "send us your bills and we will add 10% as long as you exceed the quality requirements"
- Fixed price per unit – "charge us €0.01 for each query"

Can also encounter "cost-minus": "send us your bills and next year we will expect to pay 10% less"

Which to aim for? - 1

Fixed-price
- Both sides know what's coming
- Supplier is motivated to deliver, and can add contingency funding (which turns into extra profit if all goes well)
- Can be hard to change plans once things start, so quality can suffer

Time and materials
- Customer absorbs all the risks to the project's scheduling, can get hit with an unexpectedly large bill, and may have to bail out early
- Less motivation for efficiency, which can increase quality
- Relatively easy to change plans

Most contracts are now fixed price or fixed price per unit

Which to aim for? - 2

Cost-plus
- Largely similar disadvantages as time and materials
- Customer can use bonus structure for emphasis

Fixed price per unit
- Supplier absorbs risks that the product does not ship well
- Revenue stream over product lifetime
- Easily modify the product to improve its attractiveness in the light of feedback

Current moves towards a software component market will be based on fixed price per *use* of a unit
- Also Application Service Provider Model and Web Services Model

In any case…

Getting into any kind of contract is a two -part game

- Customer – know what you are getting when, and for how much
- Supplier – know what you are building to what deadlines and with what budget

For the supplier, there are four critical tasks

- Estimating how long things will take
- Planning and scheduling those things within contractual constraints
- Identifying critical paths, points and risks to be watched carefully
- Monitoring how things progress, detecting and correcting problems

Estimation

How long is a particular task going to take?

What resources will be needed to complete it?

- Planning – how much resources needed when?
- Contractual obligations – can it be delivered on time?

This is hard! – it is basically educated guesswork

- The codified experiences of other projects
- Can be more or less horribly wrong – so keep some "slack" time and resources to cover all contingencies

Many projects – and not just software – have failed through confusing estimates with truth

Why is this so important?

Parkinson's law

"Effort expands to fill the time available"

Make sure you keep time (and money) in reserve

Brook's law

"Adding manpower to a late project makes it later"

It's extremely hard to recover from an overrun

Taken from Brooks, *The mythical man-month*, Addison Wesley (1995).

Hofstadter's law

"It always takes longer than you expect, even if you take Hofstadter's law into account"

Never trust an estimate

Taken from Hofstadter, *Gödel, Escher, Bach*, Penguin (1979)

Approaches to estimation

Algorithmic – model of the requirements plus "effort factors" of how hard they are to meet

Expert judgement – ask someone who might know

Analogy – what it took last time

Top-down – estimate the whole project, break down to component tasks

Bottom-up – identify and estimate the component tasks, and sum them for the full estimate

Price to win – the highest price that will win the tender

Parkinson's law – how much time have we got?

Eliminate the unsuitable…

Parkinson gets bitten by Brooks

- Unless you have a schedule, it's impossible to know whether you're on time
- Once you fall behind, it's hard to recover

Price-to-win is great – as long as you can collect your winnings

- Works if the customer over-estimated the difficulty (or believed your over-estimate) – doesn't happen very often
- Can leave the supplier with a large loss – they like to avoid that, so quality suffers and their reputation nose-dives accordingly

Analogy often doesn't work well with software

- Similarities can be superficial, differences have a non-linear impact
- Only use this if you're building an identical system again – re-use!

…and what remains might work

Bottom-up

- Break down the task into component sub-tasks, until you get something small enough to be accurately guessed
- Sum all the estimates (and add a percentage for error)

Top-down

- Estimate the overall size of the project and apply some kind of productivity factor
- Can use bottom-up to estimate the rough size of the project

Expert judgement

- If there's an expert around, use her!
- …but she'll only informally apply these same techniques – just with a better "feel" for the problem areas

How "big" is a software project?

A function of many things

- Size of the code – large projects take longer, and their complexity grows non-linearly
- Novelty of the project – takes longer if you have to understand the problem space from scratch first
- Importance of the project – harder to build an aeroplane control system than a word processor of similar size
- Experience of the team – programmers (and analysts, and designers, …) vary radically in their capabilities

Estimating size – *software metrics*

- Common approach – try to move from requirements to a guess about the size of code needed to meet them
- Function points – requirements to computational elements
- COCOMO – account for non-functional requirements and context

Function points

1. Count the number of inputs, outputs, internal structures *etc* from the analysis

2. Classify them in terms of their average complexities

User type	**Complexity** Low	Medium	High
External input types	3	4	6
External output types	4	5	7
Logical internal file types	7	10	15
External file types	5	7	10
External enquiry types	3	4	6

3. Multiply by the factors and sum to get the total number of FPs

4. Multiply by a programming language specific number to get KLOC

91 for COBOL, 128 for C, …

COnstructive **CO**st **MO**del

The current state of the art for large projects

- Data derived from analysis of real projects
- Classify projects as *organic*, *embedded* and *semi-detached* with productivities varying for each

Estimate source size KDSI — Thousands of delivered source-level instructions (statements)

In person-months

Nominal effort for project $NE = 3.2 \times KDSI^{1.05}$

Productivity for an "organic" (small and straightforward) project

Exponent accounts for the non-linear complexity growth of large projects

Multiply by the applicable development effort multipliers $E = NE \times e_1 \times e_2 \times \ldots$

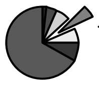
Suspension of disbelief

The numbers that come out of these metrics look dreadful

- KDSI = 12, NE = 43 person-months – about 300 lines a month
- Most people can write that code volume in a night…

But that's just code – and, as we saw, that's a miniscule fraction of the overall system development time

- Design, documentation, maintainability, re-use, …



300 lines a month is about 15% of the average overall activity per month – or about 35% if we treat maintenance separately

COCOMO effort multipliers

Development factors

- Required reliability, analyst capabilities, …
- The state of each for the project is rated from "very low" through "nominal" to "extra high"

	Rating					
Cost driver	V low	Low	Nominal	High	V high	X high
Analyst's capabilities	1.46	1.19	1.00	0.86	0.71	

Boehm, *Software engineering economics*, IEEE Trans on Software Engineering **10** (Jan 1984)

NE = 43 person-months

Inexperienced analyst E = 63

E = 31 Experienced analyst

Does estimation work?

Well, sort of…the basic problem is the need to derive numbers for qualities like "difficulty"

- Getting it wrong (accidentally or deliberately) completely changes the estimate – real numbers are flexible things…
- Only source of data is past performance, so very questionable for novel projects

Big, complex, unreliable and unwieldy – but only quantitative game in town

- Expert qualitative estimates probably work best for small projects
- …but for large projects, these approaches are as good as it gets

More mature organisations are generally better at estimation than younger ones

Capability Maturity Models

Initial – haphazard procedures, dependent on individuals' skills

Repeatable – basic procedures in place, but still quite individual in application

Defined – agreed and enforced definition of how tasks are performed

Managed – processes are measured and controlled

Optimised – measurements feed-back to improve the process at each run

Organisation becomes increasingly able to improve its approach to its business

What's the point?

We're striving to stop software being a *craft* and make it *engineering*

- Don't buy a bridge from someone who can't tell you how much steel it'll need…

A mature organisation *records* how it performs its tasks and *reflects* on how to improve next time

- Know what worked and what didn't – and reject the latter next time
- Know how the tasks were performed – and see whether they were performed as well as possible
- Know how long things took – and use this knowledge in the next tender

Sometimes called *institutional knowledge*

- Things that the organisation has taken to its heart

Planning

Complex processes require planning; improving them requires quite formal planning and monitoring

Deciding what gets done when

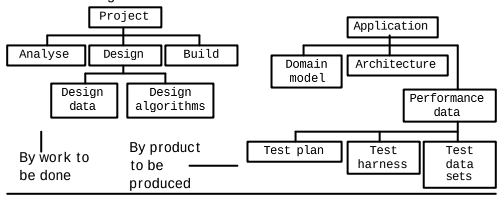
- Feasibility – is it possible to do this?
- Resource allocation – what needs to be made available when?
- Costing – what costs occur when?
- Motivation – does everyone know what they should be doing?
- Co-ordination – managing the activities of several projects concurrently

More generally, gives insight into how well an organisation conducts its business

Activities

Each project is composed of a number of activities

- Activities are related – some must be done before others
- Aim for clearly-defined purpose – specify the deliverables
- Aim for clearly-defined resources – decide what's needed and for how long



Scheduling activities

Having identified the activities, we need to map them onto a schedule for the project

- Assign staff effort to each activity
- Maintain flexibility within any hard completion deadlines

Two main approaches

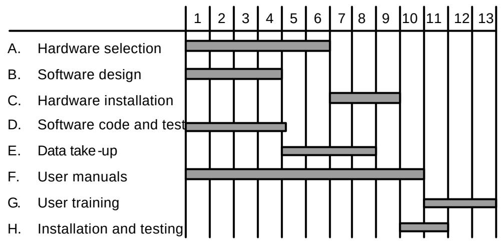
- Critical Path Method (CPM) networks
- GANTT charts

Example

Activity	Duration	Precedents
A. Hardware selection	6	
B. Software design	4	
C. Hardware installation	3	A
D. Software code and test	4	B
E. Data take-up	3	B
F. User manuals	10	
G. User training	3	E,F
H. Installation and testing	2	C,D

Adapted from Hughes and Cotterell, *Software project management*, McGraw-Hill (1999)

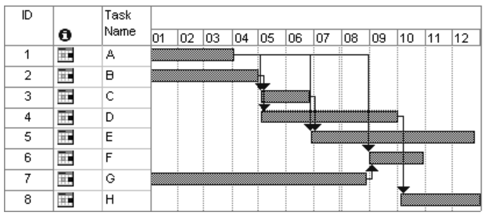
GANTT charts



Not very detail, but can be understood easily "at a glance"

Sometimes put the number of person-months each month along the bottom

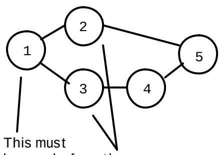
An actual example



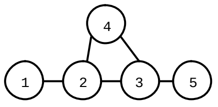
Critical path method

A network model of activities indicating their scheduling constraints and durations

- One start node, one end node, time moves from left to right
- Nodes indicate events; links indicate sequencing and duration
- No loops or dangles

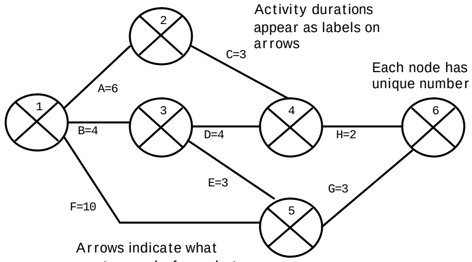


This must happen before these



Although loops happen in practice they should be avoided in planning as they indicate potentially infinite delay

Basic example network



Activity durations appear as labels on arrows

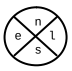
Each node has a unique number

Arrows indicate what must occur before what

Adding time

From a basic network we add scheduling estimates

- Earliest possible completion times
- Latest possible start time
- Amount of “slack” in the schedule



Node number

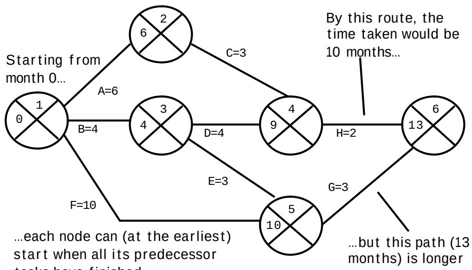
Earliest start time

Latest start time

Slack time

Times are usually measured in “wall clock” months, starting from zero – so elapsed times rather than effort times

Example – forward pass



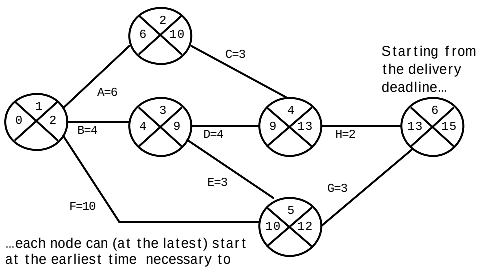
Starting from month 0…

By this route, the time taken would be 10 months…

…each node can (at the earliest) start when all its predecessor tasks have finished

…but this path (13 months) is longer

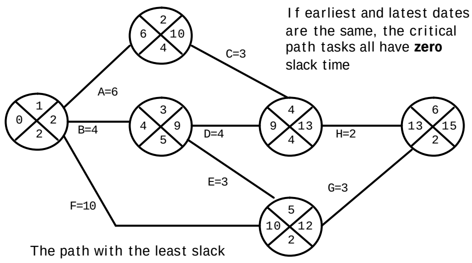
Example – backward pass



Starting from the delivery deadline…

…each node can (at the latest) start at the earliest time necessary to keep its successors on time

Example – the critical path



If earliest and latest dates are the same, the critical path tasks all have **zero** slack time

The path with the least slack time is the **critical** path

Significance

Delays on the critical path delay the completion of the project

- The risks associated with critical path tasks are the really important ones to identify and manage

To shorten the project we have to shorten the critical path

- Shortening anything else may gain flexibility but not time

Slack times show the amount a single event may move without affecting the project end time *assuming* nothing else slips

- Actually a property of paths, not events…

Planning summary

A mature organisation is characterised by the way it plans and conducts its business

- Monitoring, control, reliability, repeatability, …

Planning concerns the scheduling and timing of activities

- Critical paths focus the mind on the most important areas
- Summaries for higher management

Managing the plan requires both monitoring its progress and assessing (and reacting to) its risks

Project management

Contracts
- Specification, schedule, maintenance, payments, ownership

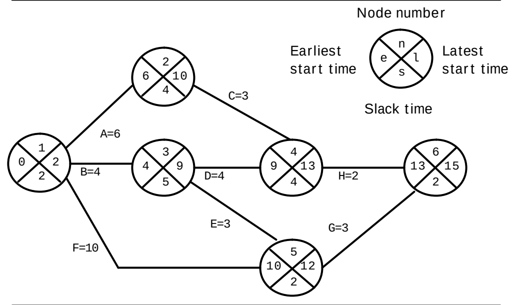
Estimation
- Identifying activities, their duration, and the recourses needed
- Algorithmic, expert judgement, analogy, top-down, bottom-up
- Functions points – size, COCOMO - effort

Planning
- Mapping activities to a schedule
- Critical path method, Gantt charts

Monitoring and control
- Project progress

Risk assessment and management

Critical path method



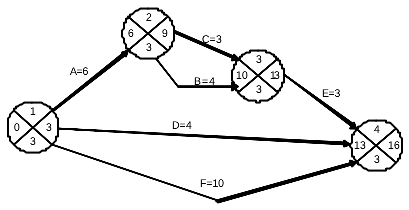
Example

Activity	Duration	Precedents
A. Hardware selection	6	
B. Software design	4	A
C. Hardware installation	3	A
D. Software code and test	4	
E. Data take-up	3	B,C
F. User manuals	10	

Slack time = 3 months

Adapted from Hughes and Cotterell, *Software project management*, McGraw-Hill (1999)

The diagram



Where is the critical path?

Monitoring and control

The important thing about the plan is that it gives you a framework within which to gauge a project's progress

- Not necessarily important to stick rigidly to the plan – in fact that can be downright dangerous
- …but it *is* important to *know* when you've deviated
- …or, put another way, dump the plan deliberately, not by chance

Knowing when to deviate means knowing "where you are" along the project plan

Deviation from the plan

Several dimensions

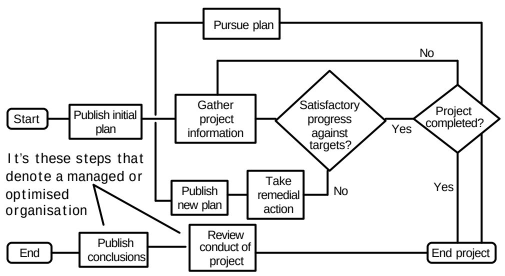
- Time – slipping behind schedule, especially along the critical path or where all the slack time is used up (*I.e.* late in the project)
- Functionality/quality – the product doesn't meet specifications or expectations
- Resources – the product needs more work expended on it

In all these cases (especially the last) Brooks' law will make things hard to correct

"Adding manpower to a late project makes it later"

- …although the emphasis here is "hard", not "impossible"
- It *is* possible to add effort to a project: it's just that the new staff need time to get up to speed on the project before they can usefully contribute to it

Project control cycle


Pursue plan
No
Start
Publish initial plan
Gather project information
Satisfactory progress against targets?
Yes
Project completed?
It's these steps that denote a managed or optimised organisation
Publish new plan
Take remedial action
No
Yes
End
Publish conclusions
Review conduct of project
End project

Adapted from Hughes and Cotterell, *Software project management*, McGraw-Hill (1999)

Management information

What sorts of things to collect? Ideally want

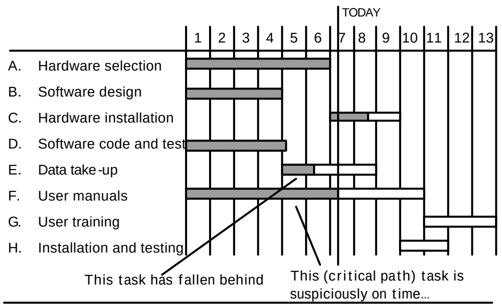
- Objective, tangible data – date of delivery, results of performance or reliability experiments, …
- Timely – information turning up in good time to make decisions

How to collect?

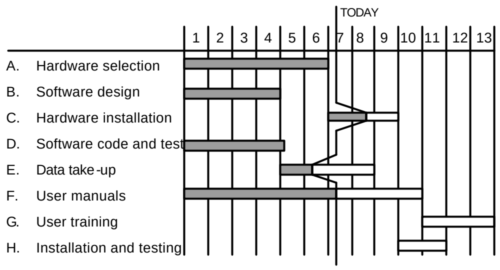
- Regular meetings – spoken reports with written minutes
- Time sheets or other formal progress reports
- …and discussions in coffee lounges can provide early warning of problems – people will say more when it's not being recorded

Written minutes prevent later denials

Presentation – GANTT charts again



Same thing, more impact



Priorities

All tasks are not equal

- Critical path tasks – problems here have the largest impact
- "No slack" tasks – tasks which have slipped as far as you can allow
- Sub-critical tasks – important but fairly robust to delays
- Peripheral tasks – can live without them being on time

If resources are scarce – and they always are – we need to establish priorities for dealing with issues

Things can get very difficult very quickly

- Problems tend to "snowball" – a failure in one place leads to cascade failures elsewhere
- …and to a general loss of confidence, which can be worst of all

What can we do if things slip?

No plan survives its first contact with reality

First rule: don't panic (too soon)

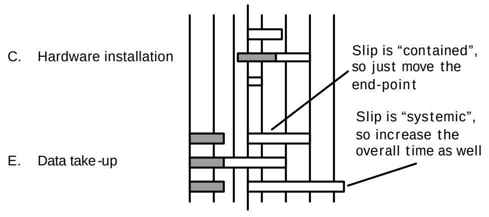
- *All* projects suffer delays: it's perfectly natural
- Precipitate action can cause more problems than it solves

Second rule: the plan did not come from God

- It's a guide to be followed for exactly as long as it's helpful, relevant and cost-effective to do so
- …and should be dumped as soon as this isn't the case

Re-scheduling

Same tricks work (in a more positive way) for ahead-of-schedule activities


C. Hardware installation
E. Data take-up
Slip is "contained", so just move the end-point
Slip is "systemic", so increase the overall time as well

The "infinitely malleable plan" plan

The plan is followed at all times, as closely as possible

EXCEPT THAT

the team can change the plan whenever they like

AS LONG AS

everyone in the team is in the room when the plan is revised

AND AS LONG AS

the new plan is agreed by and propagated to everyone

AND AS LONG AS

no other changes to or deviations from the plan are allowed

Changes to the plan are allowed, but only as long as everyone knows that the change has been made and agrees with it **as a group**

Options

Shorten the critical path – if possible, as it's usually been pared down as far as it'll go

Exhort the staff work harder – …and hope they aren't working as hard as they can already

Add staff – as long as there's enough slack time available to overcome Brooks' law

New staff need time to get up to speed

Change staff – if the problem is caused by personality conflicts or inexperience

Remove staff – radical, but it might work…

Re-consider the priorities – drop less essential work to focus on the most important bits

Monitoring summary

It's better to have honest bad news than well-intentioned lies

Keep track of what's going on

- Talk to the team, non-confrontationally
- Mixture of formal and informal monitoring works well
- Regularly assess progress against estimates
- De-brief a completed project - become a optimised organisation

Don't panic, but don't delay indefinitely if things start to go wrong

- Make sure changes are properly understood, agreed and publicised
- Consider dumping the plan and starting from scratch – but this will involve *major* amounts of time and effort

Risk management

"A chance of possibility of danger, loss, injury or adverse consequences"
Oxford Reference English dictionary (1998)

In software engineering, a risk is something which, if it happens, will cause the project a problem

Quick test: identify risks in software engineering in general and in your 3BA7 project in particular…

Risks

In software engineering, a risk is something which, if it happens, will cause the project a problem

- Computer crashes or thefts
- Staff illnesses or resignations
- Unexpected complexities
- Unplanned changes

A lot of risks are inherent; most can be planned for

Many management scientists would argue that risk is *the* single factor that managers deal with

Risk aversion

"I have seen a room get suddenly quiet when someone brings up a 'concern' … but pretending that problems will not occur will not prevent them."

Dwayne Phillips, *The project manager's handbook*, IEEE Computer Society (1998). Quoted in Hughes and Cotterell, *Software project management*, McGraw-Hill (1999)

Risk engineering approach

Risk identification – list all the risks which would affect the outcome of the project
Risk estimation – their probability, and their impact if they happen
Risk evaluation – rank risks and formulate strategies to deal with them
Risk planning – draw up contingency plans
Risk control – react to problems and minimise their impact on the project
Risk monitoring – re-assess everything as the project progresses – risk impacts sometimes vary with time

Barry Boehm, *A tutorial on software risk management*, IEEE Computer Society (1989)

Risk identification

Risks come from several different sources

- The application may be inherently difficult, for example a hard real-time system with on-the-limit processing requirements
- …or just very ill-specified, or very changeable
- Some staff are inherently risky – less experienced, less able to estimate, too defensive, less reflective, wanting to resign, …
- Suppliers default (or go bust), external agencies interfere, …
- Tools don't always work as they should, computers crash (or get stolen), …

Risks hunt in packs

- Compound risks and cascade failures – one thing going wrong can cause another to happen (or become more likely)

If the chief programmer resigns in disgust, the junior programmers may bail out too

Risk estimation - exposure

In financial services, *exposure* means what percentage of a company's assets are tied to a single source

- A market crashing isn't too important if the portfolio is balanced across several largely independent markets

The situation is the same for risks: risk exposure is the impact of the risk *versus* its probability

- Ideally make the impact a cash value, so multiplying the impact by the percentage allows exposures to be compared quantitatively
- Doesn't always work – what's the cost of a personal disagreement within the team? or the probability of a requirements change?
- Can use a less formal scheme – "likely", "unlikely", or "very unlikely" against "bad", "very bad" and "catastrophic"

Numbers look, good but can generate a false sense of certainty

Reducing the risks

Prevent, avoid, duck

- Avoid those circumstances which make the risk more likely
- Take active steps to reduce the likelihood of the risk
- Plan a project to stay away from unacceptably risky areas

Pass the buck

- Insurance companies are basically a form of risk management out-sourcing
- Get the customer to share some of the risks

Contingency planning

- Plans within plans within plans
- …and spare money, and slack time in the schedule

…and learn from this project ready for the next one

Achieving quality

The aim of estimation, planning, monitoring, risk management and so on is to achieve *quality*

A mature organisation might be characterised as one more likely to deliver a quality product, on time and within budget

- A set of techniques is only useful if placed within the right culture of responsibility and planned activity
- …and there's no silver bullet

Tutorial

MS Project and GANTT charts

See course page

Focus on lesson 1 of the tutorial

MS Project is available in the East End
public access computer rooms (Panoz Institute)

Tutorial

O-O Analysis and Design using CRC Cards

http://www.csc.calpoly.edu/%7Edbutler/tutorials/winter96/crc_b

Can be done as a group, for example using the course assignment groups…