Prof.Tirup Parmar

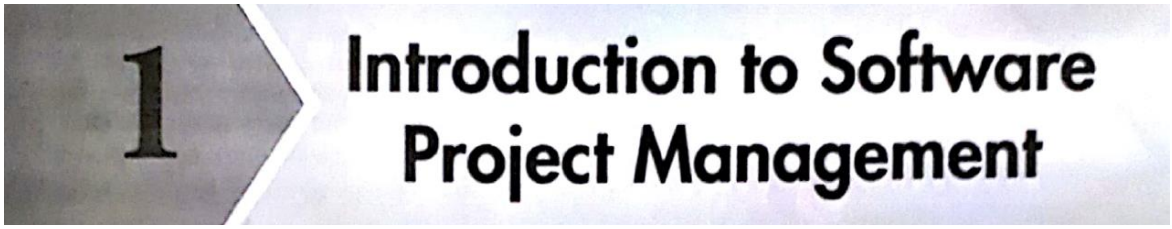
[Document subtitle]

Mumbai University

Unit	Details	Page No.
I	**Introduction to Software Project Management**: Introduction, Why is Software Project Management Important? What is a Project? Software Projects versus Other Types of Project, Contract Management and Technical Project Management, Activities Covered by Software Project Management, Plans, Methods and Methodologies, Some Ways of Categorizing Software Projects, Project Charter, Stakeholders, Setting Objectives, The Business Case, Project Success and Failure, What is Management? Management Control, Project Management Life Cycle, Traditional versus Modern Project Management Practices. **Project Evaluation and Programme Management**: Introduction, Business Case, Project Portfolio Management, Evaluation of Individual Projects, Cost–benefit Evaluation Techniques, Risk Evaluation, Programme Management, Managing the Allocation of Resources within Programmes, Strategic Programme Management, Creating a Programme, Aids to Programme Management, Some Reservations about Programme Management, Benefits Management. **An Overview of Project Planning**: Introduction to Step Wise Project Planning, Step 0: Select Project, Step 1: Identify Project Scope and Objectives, Step 2: Identify Project Infrastructure, Step 3: Analyse Project Characteristics, Step 4: Identify Project Products and Activities, Step 5: Estimate Effort for Each Activity, Step 6: Identify Activity Risks, Step 7: Allocate Resources, Step 8: Review/Publicize Plan, Steps 9 and 10: Execute Plan/Lower Levels of Planning	
II	**Selection of an Appropriate Project Approach:** Introduction, Build or Buy? Choosing Methodologies and Technologies, Software Processes and Process Models, Choice of Process Models, Structure versus Speed of Delivery, The Waterfall Model, The Spiral Model, Software Prototyping, Other Ways of Categorizing Prototypes, Incremental Delivery, Atern/Dynamic Systems Development Method, Rapid Application Development, Agile Methods, Extreme Programming (XP), Scrum, Lean Software Development, Managing Iterative Processes, Selecting the Most Appropriate Process Model. **Software Effort Estimation:** Introduction, Where are the Estimates Done? Problems with Over- and Under-Estimates, The Basis for Software Estimating, Software Effort Estimation Techniques, Bottom-up Estimating, The Top-down Approach and Parametric Models, Expert Judgement, Estimating by Analogy, Albrecht Function Point Analysis, Function Points Mark II, COSMIC Full Function Points, COCOMO II: A Parametric Productivity Model, Cost Estimation,	

	Staffing Pattern, Effect of Schedule Compression, Capers Jones Estimating Rules of Thumb.	
III	**Activity Planning:** Introduction, Objectives of Activity Planning, When to Plan, Project Schedules, Projects and Activities, Sequencing and Scheduling Activities, Network Planning Models, Formulating a Network Model, Adding the Time Dimension, The Forward Pass, Backward Pass, Identifying the Critical Path, Activity Float, Shortening the Project Duration, Identifying Critical Activities, Activity-on-Arrow Networks. **Risk Management**: Introduction, Risk, Categories of Risk, Risk Management Approaches, A Framework for Dealing with Risk, Risk Identification, Risk Assessment, Risk Planning, Risk Management, Evaluating Risks to the Schedule, Boehm's Top 10 Risks and Counter Measures, Applying the PERT Technique, Monte Carlo Simulation, Critical Chain Concepts. **Resource Allocation**: Introduction, Nature of Resources, Identifying Resource Requirements, Scheduling Resources, Creating Critical Paths, Counting the Cost, Being Specific, Publishing the Resource Schedule, Cost Schedules, Scheduling Sequence.	
IV	**Monitoring and Control**: Introduction, Creating the Framework, Collecting the Data, Review, Visualizing Progress, Cost Monitoring, Earned Value Analysis, Prioritizing Monitoring, Getting the Project Back to Target, Change Control, Software Configuration Management (SCM). **Managing Contracts**: Introduction, Types of Contract, Stages in Contract Placement, Typical Terms of a Contract, Contract Management, Acceptance. **Managing People in Software Environments**: Introduction, Understanding Behaviour, Organizational Behaviour: A Background, Selecting the Right Person for the Job, Instruction in the Best Methods, Motivation, The Oldham–Hackman Job Characteristics Model, Stress, Stress Management, Health and Safety, Some Ethical and Professional Concerns.	
V	**Working in Teams:** Introduction, becoming a Team, Decision Making, Organization and Team Structures, Coordination Dependencies, Dispersed and Virtual Teams, Communication Genres, Communication Plans, Leadership. **Software Quality:** Introduction, The Place of Software Quality in Project Planning, Importance of Software Quality, Defining Software Quality, Software Quality Models, ISO 9126, Product and Process Metrics, Product versus Process Quality Management, Quality Management Systems, Process Capability Models, Techniques to Help Enhance Software Quality, Testing, Software Reliability, Quality Plans.	

Unit I



In this introduction the main questions to be addressed will be:

- What is software project management? Is it really different from 'ordinary' project management?
- How do you know when a project has been successful? For example, do the expectations of the customer/client match those of the developers?

The two key questions are:

1. *What exactly is software project management?*

This is going to be tackled by looking firstly at what is meant by 'project'. We are then going to examine whether 'software project management' is really different from 'normal' project management. Is there anything special about software as opposed to other engineered artefacts?

2. *How do we define whether a project is a success or not?*

The point about studying project management is to be able to have successful projects. So how do we know if we have been successful?

Why is project management important?

- Large amounts of money are spent on ICT e.g. UK government in 2003-4 spent £2.3 billions on contracts for ICT and only £1.4 billions on road building
- Project often fail – Standish Group claim only a third of ICT projects are successful. 82% were late and 43% exceeded their budget.
- Poor project management a major factor in these failures

The methodology used by the Standish Group to arrive at their findings has been criticized, but the general perception of the prevalence of ICT project failure is still clear.

What is a project?

Some dictionary definitions:

"A specific plan or design"

"A planned undertaking"

"A large undertaking e.g. a public works scheme"

Longmans dictionary

Key points above are *planning* and *size* of task

- An endeavor with specific objectives:
  - Usually consists of multiple tasks
  - With defined precedence relationships
  - With a specific time period for completion
- Non-Software Examples:
  - A wedding
  - An MBA degree
  - A house construction project
  - A political election campaign

Here are some definitions of 'project'. No doubt there are other ones: for example

'Unique process, consisting of a set of coordinated and controlled activities with start and finish dates, undertaken to achieve an objective conforming to specific requirements, including constraints of time, cost and resources'

BSO ISO 10006: 1997

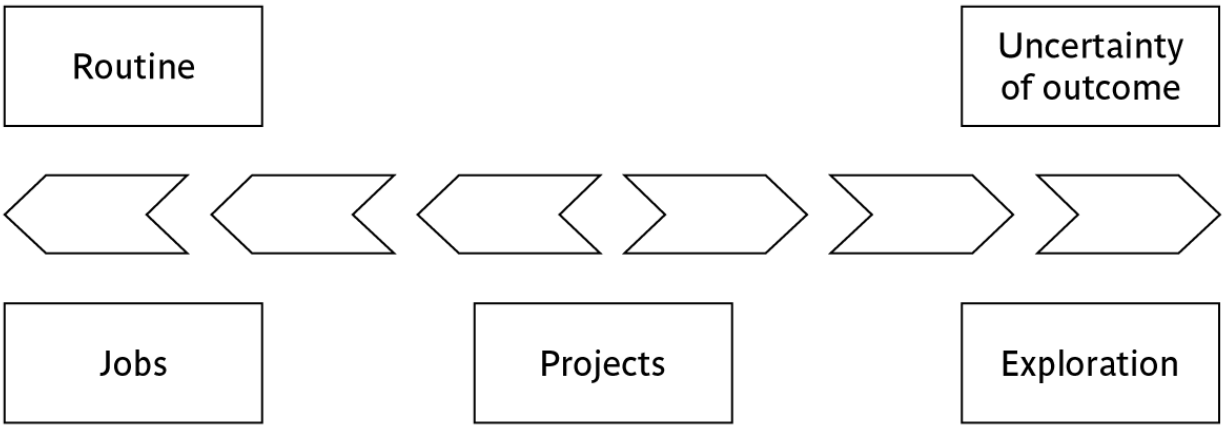
What is a Task?

- A small piece of work:
  - Meant to accomplish a straightforward goal
  - Effort of no longer than a few person-hours

- Involves only a few people
- May or may not be a part of some project
- Usually repetition of a previously accomplished task
- Process management may be relevant!

- Non-software Examples:
  - Attend a lecture class
  - Buy a chocolate from the market
  - Book a railway ticket

Jobs versus projects



'Jobs' – repetition of very well-defined and well understood tasks with very little uncertainty

'Exploration' – e.g. finding a cure for cancer: the outcome is very uncertain

Projects – in the middle!

On the one hand there are repetitive jobs a similar task is carried out repeatedly, for example Kwikfit replacing a tyre on a car or a lecturer giving an introductory talk on project management. The task is well-defined and there is very little uncertainty. In some organizations, software development might tend to be like this – in these environments software process management might be more important than software project management

On the other hand some exploratory activities are very uncertain. Some research projects can be like this – we may not be sure what the outcome will be, but we hope that we will learn some things of

importance. It may be very difficult to come up with precise plans, although we would probably have some idea of a general approach.

Projects seem to come somewhere between these two extremes. There are usually well-defined hoped-for outcomes but there are risks and uncertainties about achieving those outcomes.

Characteristics of projects

A task is more 'project-like' if it is:

- Non-routine
- Planned
- Aiming at a specific target
- Carried out for a customer
- Carried out by a temporary work group
- Involving several specialisms
- Made up of several different phases
- Constrained by time and resources
- Large and/or complex

Exercise 1.1 in the Software Project Management text is a good way of introducing this material if you have time. I have found this exercise to be a good 'ice-breaker'. Get each student to list the example activities in an order which matches the degree to which they merit the description of 'project'. You can create a grid on a whiteboard with the projects on the vertical axis and the positions 1st, 2nd, 3rd etc on the horizontal axis. You then go through asking how many put 'producing a newspaper' first, second, etc. (Avoid making jokes about this being like the Eurovision song contest). This is time-consuming but it does mean that every student participates in building up a general picture of people's perceptions, and you can discuss disagreements in perceptions as you go along.

Are *software* projects really different from other projects?

Not really …but... The factors

- Invisibility

- Complexity
- Conformity
- Flexibility

make software more problematic to build than other engineered artefacts.

This is based on Fred Brooks' paper No Silver Bullet: Essence and Accidents of Software Engineering which appeared in IEEE Computer 20(4) pp10-19 April 1987

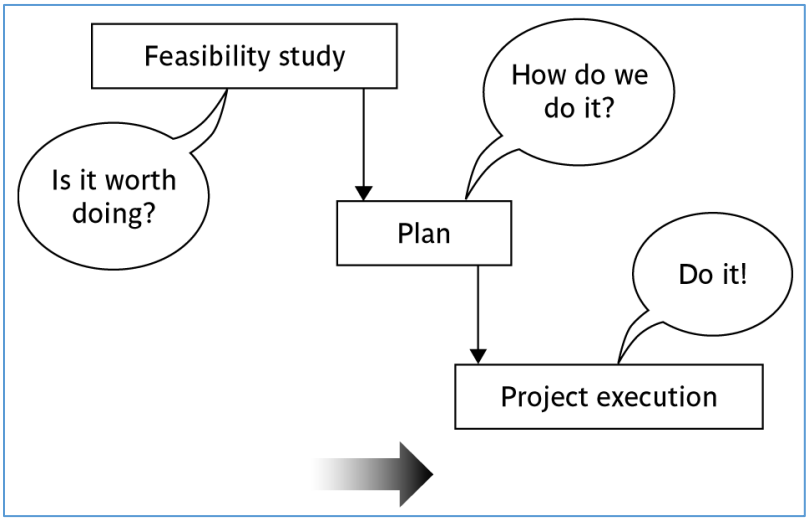
Contract management versus technical project management

Projects can be:

- **In-house**: clients and developers are employed by the same organization
- **Out-sourced:** clients and developers employed by different organizations
- 'Project manager' could be:
  - a 'contract manager' in the client organization
  - a technical project manager in the supplier/services organization

In general the book looks at things from the point of view of the technical software project manager.

Activities covered by project management



Feasibility study:

Is project technically feasible and worthwhile from a business point of view?

Planning:

Only done if project is feasible

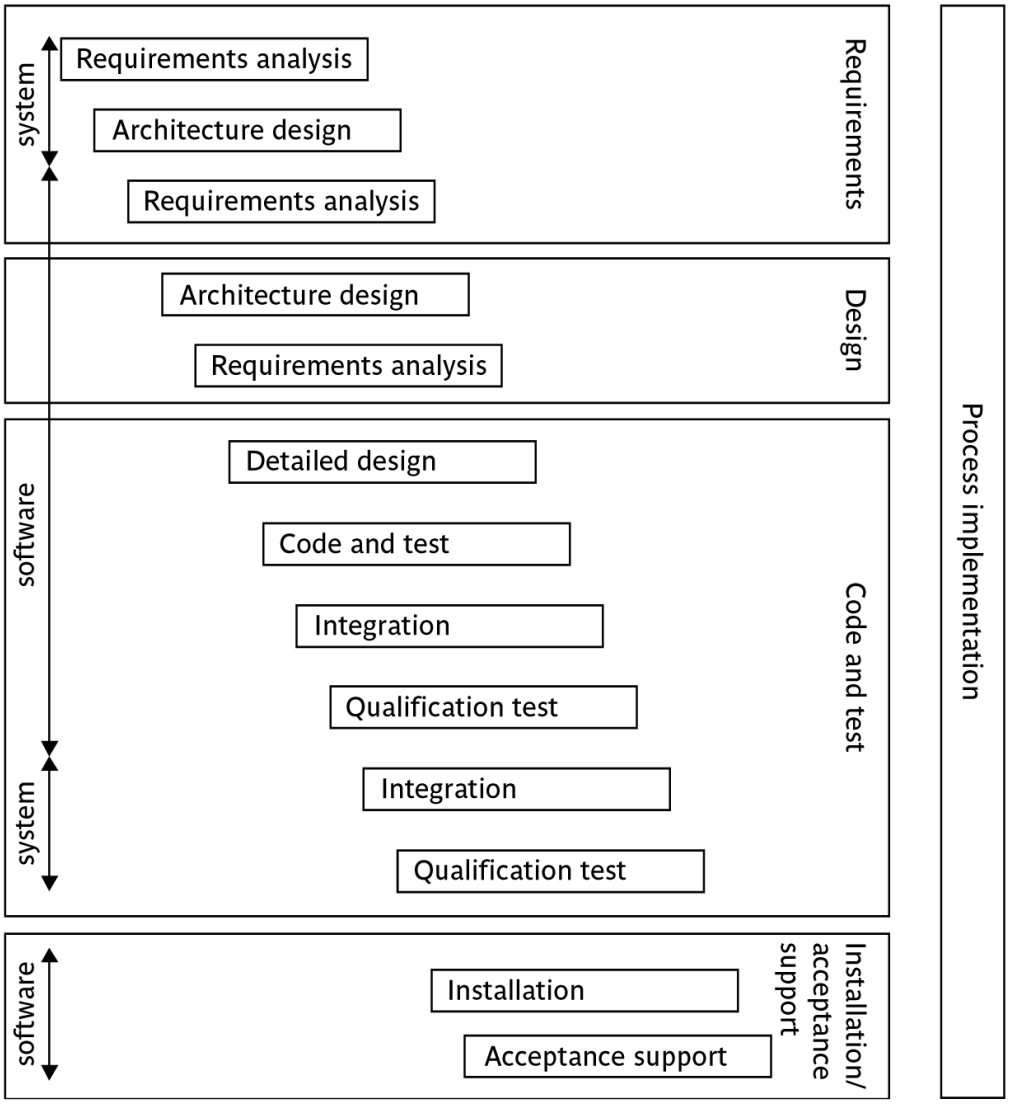
Execution:

Implement plan, but plan may be changed as we go along

There are two key points here.

1. *Often you see something like 'feasibility study' being put as the first stage of development life cycle, and indeed it might be. However, the recommendation of the feasibility study might be not to carry out the proposed project. Planning of the project should therefore take place after the feasibility study (or as a part of the feasibility study perhaps). Clearly the feasibility study itself might need a plan.*
2. *All plans are to some extent provisional and subject to change. The key point is that the evolving plan allows us to control the project.*

The software development life-cycle (ISO 12207)



Note that this is a technical model. It identifies the technical constraints on the order activities are done. This does NOT imply that a 'waterfall' approach is the only way to organize projects. The technical model could be implemented as increments or in an evolutionary manner.

ISO 12207 life-cycle

- Requirements analysis
  - Requirements elicitation: what does the client need?
  - Analysis: converting 'customer-facing' requirements into equivalents that developers can understand
  - Requirements will cover
    - Functions
    - Quality
    - Resource constraints i.e. costs

The key point here is that requirement analysis has to face in (at least) two different directions. It needs to communicate and elicit the requirements of the users, speaking in their language. It needs to organize and translate those requirements into a form that developers can understand and relate to.

- Architecture design
  - Based on *system requirements*
  - Defines components of system: hardware, software, organizational
  - *Software requirements* will come out of this
- Code and test
  - Of individual components
- Integration
  - Putting the components together

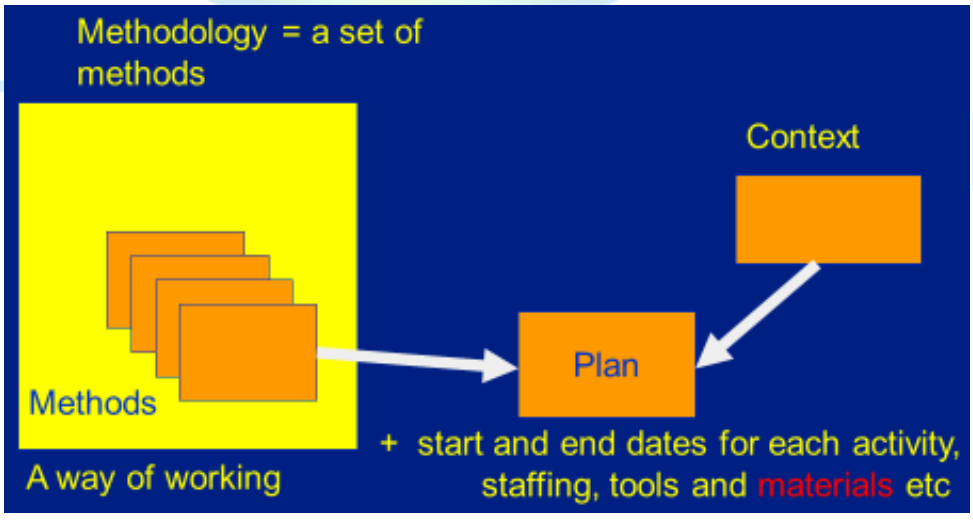
- *The software project will almost certainly be part of a larger project which has non-software elements. In a software engineering environment it could be the software will be embedded in hardware product of some kind. Thus there are system requirements for the product as a whole and software requirements for the software element.*

- *In a business information systems environment, the software development could be a relatively minor part of a much larger organizational change project.*

- Qualification testing
  - Testing the *system* (not just the *software*)
- Installation
  - The process of making the system operational
  - Includes setting up standing data, setting system parameters, installing on operational hardware platforms, user training etc
- Acceptance support
  - Including maintenance and enhancement

The confusion about what 'aimplementation' really means could be mentioned. Does it mean implementing the design (that is, coding) or implementing the complete system in its user environment? It is best to use 'installation' to describe the latter in order to avoid confusion.

Plans, methods and methodologies



A plan of an activity must be based on some idea of a method of work. While a method relates to a type of activity in general, a plan takes one or more methods and converts them into real activities by identifying:

- *Start and end dates*

- *Who will carry it out*
- *What tools and materials would be needed.*

A methodology is a set of related methods. Strictly speaking 'methodology' ought to mean the study of methods!

Some ways of categorizing projects

Distinguishing different types of project is important as different types of task need different project approaches e.g.

- Voluntary systems (such as computer games) versus compulsory systems e.g. the order processing system in an organization
- Information systems versus embedded systems
- Objective-based versus product-based
- Product-development versus outsourced

- *With objective-based projects, a general objective or problem is defined, and there are several different ways in which that objective could be reached. The project team have freedom to select what appears to be the most appropriate approach.*
- *With product-based projects, the product is already very strictly defined and the development team's job is to implement the specification with which they have been presented.*
- *Arguably, information systems projects are more likely to be objective-based than is the case with software engineering.*
- *In many cases, an objective-based project could consider a problem and recommend a solution that is then implemented by a product-based project.*
- *Exercise 1.5 in the text is relevant here.*

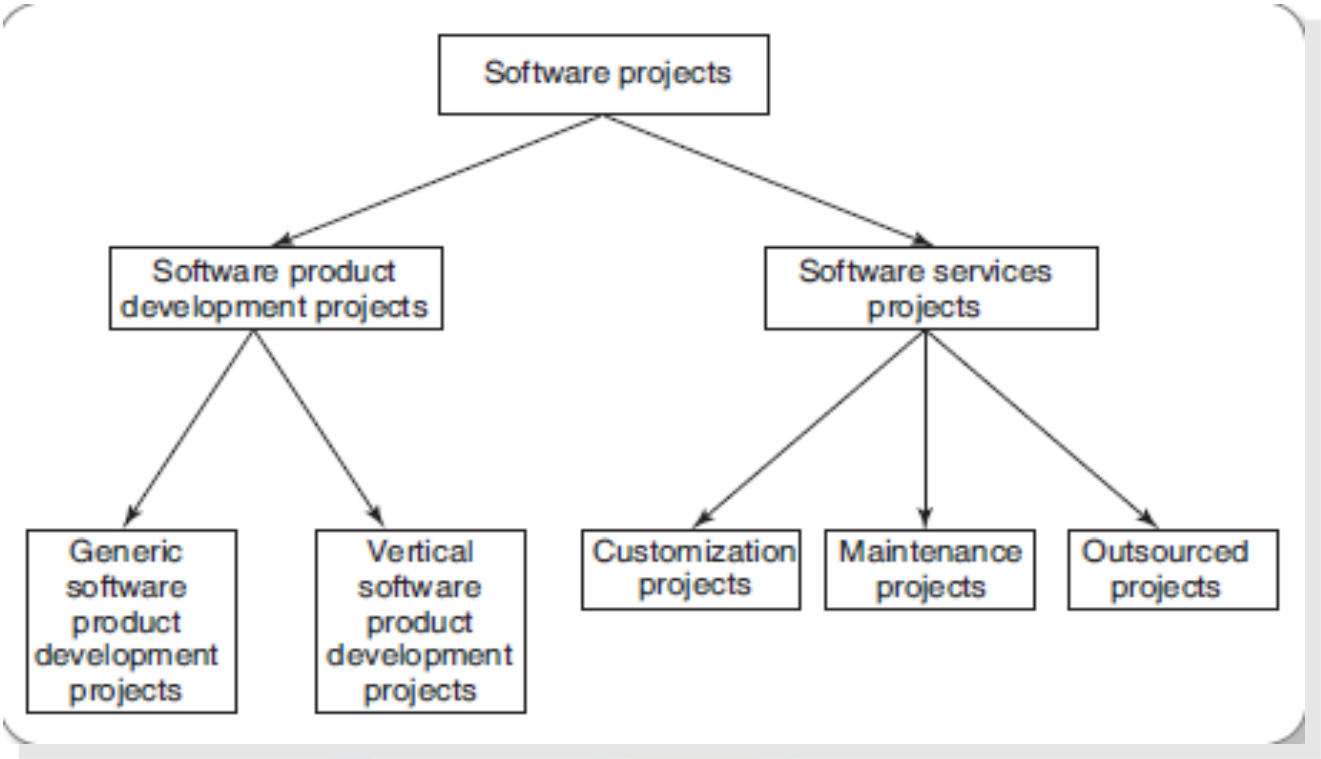


Fig. A Categorization of Software Projects

Two Types of Software Projects

- Software product development projects
- Software services projects

Software Services

- Software service is an umbrella term, includes:
  - Software customization
  - Software maintenance
  - Software testing
  - Also contract programmers who carry out coding or any other assigned activities.

Stakeholders

These are people who have a stake or interest in the project

In general, they could be *users/clients* or *developers/implementers*

They could be:

- Within the project team
- Outside the project team, but within the same organization
- Outside both the project team and the organization

Different stakeholders may have different objectives – need to define common project objectives

Each stakeholder will have their own goals and concerns in relation to the project which may be different from those of the project as a whole. For example, a software developer might work to make a living, pay the mortgage, learn new things, solve interesting problems. The main stakeholders need, however, to understand and accept overall project objectives that everyone can agree to.

See Exercise 1.6 in the text.

Setting objectives

- Answering the question '*What do we have to do to have a success?*'
- Need for a *project authority*
  - Sets the project scope
  - Allocates/approves costs
- Could be one person - or a group
  - Project Board
  - Project Management Board
  - Steering committee

- *Different people who are involved in a project (Stakeholders) will have different interests in the project and are likely to see different outcomes as being important.*
- *For example, end-users would want a system that is 'user-friendly', that is, easy to learn and to use, and a system that helps rather than hinders them from doing their jobs. Their managers may be more interested in whether the new system would allow them to reduce staffing levels.*

- *It is important therefore that a set of clearly defined objectives are identified and published for the project. Some individual or group needs to be pinpointed who acts as the main client for the project.*
- *See Exercise 1.7 in the text.*

Objectives

Informally, the objective of a project can be defined by completing the statement:

The project will be regarded as a success

if…….

……

Rather like *post-conditions* for the project

Focus on *what* will be put in place, rather than *how* activities will be carried out

The focus here needs to be on what the situation will be when the project is completed. In what ways will the world be different? The objectives should avoid describing activities:

e.g. 'a new payroll application will be operational by 4th April' not 'design and code a new payroll application'

Objectives should be SMART

S – specific, that is, concrete and well-defined

M – measurable, that is, satisfaction of the objective can be objectively judged

A – achievable, that is, it is within the power of the individual or group concerned to meet the target

R – relevant, the objective must relevant to the true purpose of the project

T – time constrained: there is defined point in time by which the objective should be achieved

I have seen some places where the R is said to stand for 'resource-constrained', that is that there is a target cost associated with the achievement of the objective.

Goals/sub-objectives

These are steps along the way to achieving the objective

Informally, these can be defined by completing the sentence

To reach objective X, the following must be in place

A……

B……

C…… etc

Scoring a goal in football is a 'goal' or sub-objective on the way to achieving the overall objective of winning the match. Sub-objectives and objectives can be nested in a hierarchy, so that the objective of winning the match could itself be a goal or sub-objective on the way to winning the league etc.

Often a goal can be allocated to an individual

Individual might have the capability of achieving goal on their own, but not the overall objective e.g.

Overall objective – user satisfaction with software product

Analyst goal – accurate requirements

Developer goal – reliable software

Goals can be formulated in such a way that they represent what an individual or group need to do to contribute to the success of the project's objectives.

In the example above, the analyst or developer, by themselves, cannot guarantee user satisfaction. However, the analyst can contribute to the achievement of the objective by making sure the users' requirements are accurately recorded and the developer by making sure that the software is reliable.

See Exercise 1.7 in the text.

Measures of effectiveness

How do we know that the goal or objective has been achieved?

By a practical test, that can be objectively assessed.

e.g. for user satisfaction with software product:

- Repeat business – they buy further products from us
- Number of complaints – if low etc etc

See Exercise 1.8 in the text.

The business case

Benefits of delivered project must outweigh costs

Costs include:

- Development
- Operation

Benefits

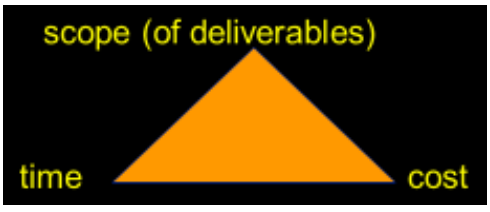
Quantifiable

Non-quantifiable

It is not always possible to put a precise financial on the benefits of a project. The client's willingness to pay up to a certain price to get a project implemented implies that they have informally identified a value to them of getting that project implemented.

Project success/failure

- Degree to which objectives are met



In general if, for example, project is running out of time, this can be recovered for by reducing scope or increasing costs. Similarly costs and scope can be protected by adjusting other corners of the 'project triangle'.

Other success criteria

These can relate to longer term, less directly tangible assets

- Improved skill and knowledge

- Creation of assets that can be used on future projects e.g. software libraries
- Improved customer relationships that lead to repeat business

What is management?

This involves the following activities:

- Planning – deciding what is to be done
- Organizing – making arrangements
- Staffing – selecting the right people for the job
- Directing – giving instructions
- Monitoring – checking on progress
- Controlling – taking action to remedy hold-ups
- Innovating – coming up with solutions when problems emerge
- Representing – liaising with clients, users, developers and other stakeholders

Exercise 1.6 (a day in the life of a project manager) is of relevance here.

Project Planning

- Carried out before development starts.
- Important activities:
  - Estimation
  - Scheduling
  - Staffing
  - Risk management
  - Miscellaneous plans

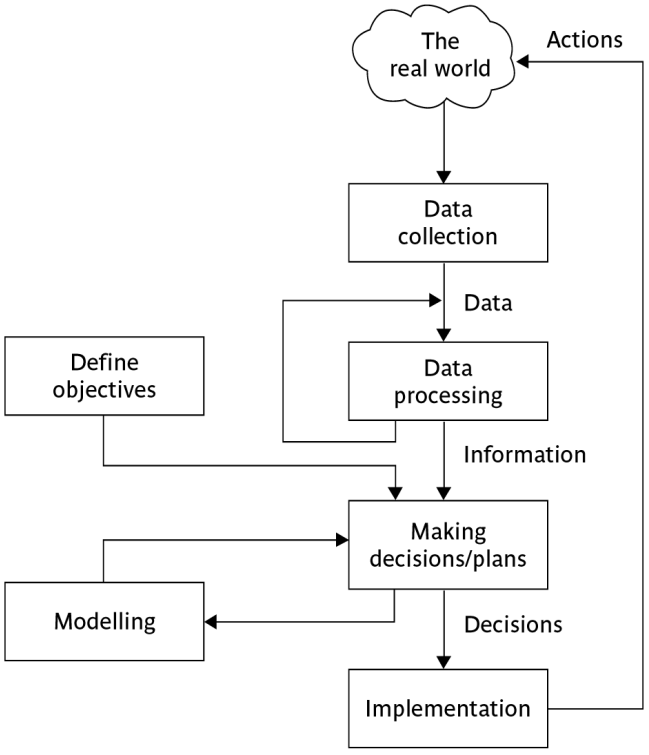
In the project initiation stage, an initial plan is made. As the project start, the project is monitored and controlled to proceed as per the plan. But, the initial plan is refined from time to time to factor in additional details and constraints about the project become available.

Traditional versus Modern Project Management

- Projects are increasingly being based on either tailoring some existing product or reusing certain pre-built libraries.
- Facilitating and accommodating client feedbacks
- Facilitating customer participation in project development work
- Incremental delivery of the product with evolving functionalities.

Rather than making a long term project completion plan, the project manager now plans all incremental deliveries with evolving functionalities. This type of project management is often called Extreme project management.

Management control



Data – the raw details

e.g. *'6,000 documents processed at location X'*

Information – the data is processed to produce something that is meaningful and useful

e.g. 'productivity is 100 documents a day'

Comparison with objectives/goals

e.g. we will not meet target of processing all documents by 31st March

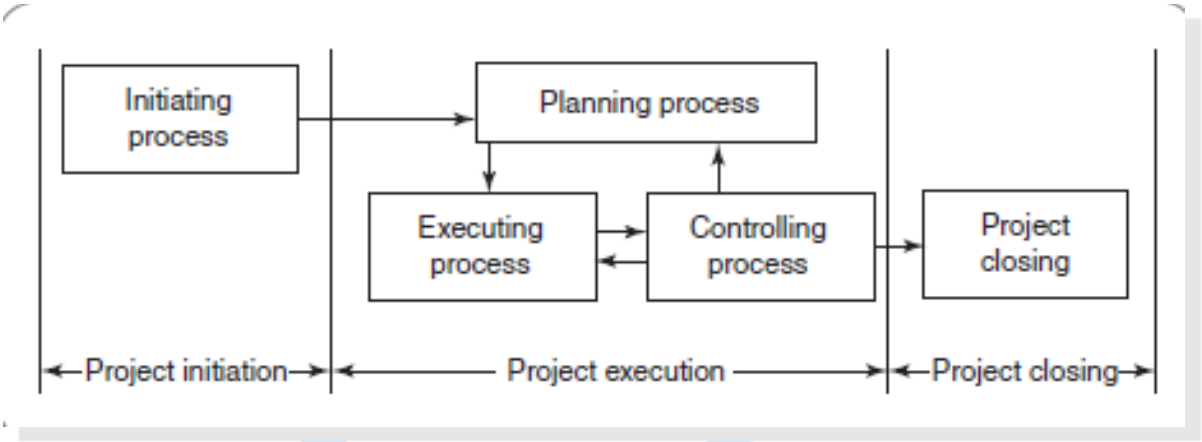
Modelling – working out the probable outcomes of various decisions

e.g. if we employ two more staff at location X how quickly can we get the documents processed?

Implementation – carrying out the remedial actions that have been decided upon

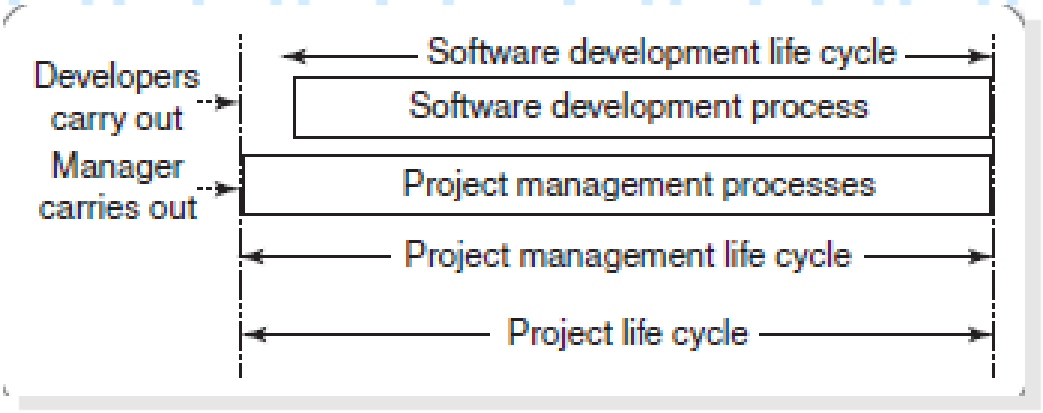
> *The authors' view is that an initially defective plan can often be remedied by good project control and management.*

Project Management Processes



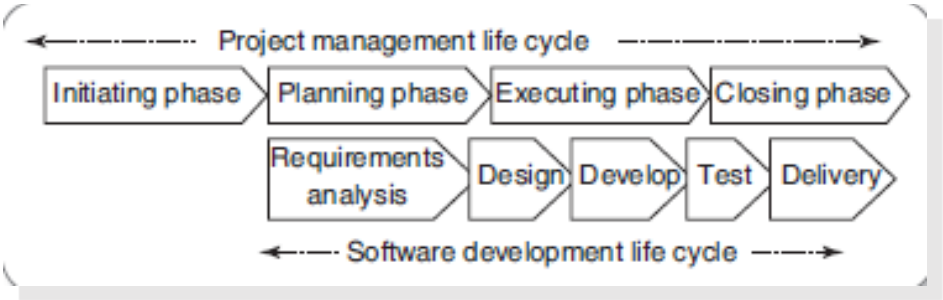
In the project initiation stage, an initial plan is made. As the project starts, the project is executed and controlled to proceed as planned. Finally, the project is closed.

Project Management Life Cycle Versus Product Development Life Cycle



During the software development life cycle, the software developers carry out several types of development processes. On the other hand, during the software project management life cycle, the software project manager carries out several project management processes

Phases of Project Management Life Cycle



Project Initiation

- During the project initiation phase it is crucial for the champions of the project to develop a thorough understanding of the important characteristics of the project.
- In his W5HH principle, Barry Boehm summarized the questions that need to be asked and answered in order to have an understanding of these project characteristics.

W5HH Principle

- A series of questions that lead to a definition of key project characteristics:
  - Why is the software being built?
  - What will be done?
  - When will it be done?
  - Who is responsible for a function?
  - Where are they organizationally located?
  - How will the job be done technically and managerially?
  - How much of each resource is needed?

Project Planning

- Various plans are made:
  - *Project plan: A*ssign project resources and time frames to the tasks.
  - *Resource plan: List the resources, manpower and equipment that required to execute the* project.

- *Financial plan: plan for manpower, equipment and other costs.*
- *Quality plan: Plan of quality targets and control.*
- *Risk plan: Identification of the potential risks, their prioritization and a plan* for the actions that would be taken to contain the different risks.

Project Execution

- Tasks are executed as per the project plan
- Monitoring and control processes are executed to ensure that the tasks are executed as per plan
- Corrective actions are initiated whenever any deviations from the plan are noticed.

Project Closure

- Involves completing the release of all the required deliverables to the customer along with the
- necessary documentation.
- Subsequently, all the project resources are released and supply agreements with the vendors are terminated and all the pending payments are completed.
- Finally, a post-implementation review is undertaken to analyze the project performance and to list the lessons learnt for use in future projects.

Key points in lecture

- Projects are non-routine - thus uncertain
- The particular problems of projects e.g. lack of visibility
- Clear objectives which can be objectively assessed are essential
- Stuff happens. Not usually possible to keep precisely plan – need for control
- Communicate, communicate, communicate!

Main topics to be covered

- The business case for a project
- Project portfolios
- Project evaluation
  - Cost benefit analysis
  - Cash flow forecasting
- Programme management
- Benefits management

The business case

- **Feasibility studies** can also act as a 'business case'
- Provides a justification for starting the project
- Should show that the benefits of the project will exceed development, implementation and operational costs
- Needs to take account of business risks

See page 22

It is worth recalling the difference between project success (the project is successfully completed) and business risks (the new product or system successfully established by the project goes on to generate benefits for the organization).

Contents of a business case

1. Introduction/ background

2. The proposed project
3. The market
4. Organizational and operational infrastructure
5. The benefits
6. Outline implementation plan
7. Costs
8. The financial case
9. Risks
10. Management plan

- **Introduction/background:** describes a problem to be solved or an opportunity to be exploited
- **The proposed project:** a brief outline of the project scope
- **The market:** the project could be to develop a new product (e.g. a new computer game). The likely demand for the product would need to be assessed.
- **Organizational and operational infrastructure:** How the organization would need to change. This would be important where a new information system application was being introduced.
- **Benefits** These should be express in financial terms where possible. In the end it is up to the client to assess these – as they are going to pay for the project.
- **Outline implementation plan:** how the project is going to be implemented. This should consider the disruption to an organization that a project might cause.
- **Costs:** the implementation plan will supply information to establish these
- **Financial analysis:** combines costs and benefit data to establish value of project

Financial analysis – we will see later that the comparative value of costs and benefits may not be obvious as the timing of costs and benefits need to be taken into consideration.

Project portfolio management

The concerns of project portfolio management include:

- Evaluating proposals for projects
- Assessing the risk involved with projects
- Deciding how to share resources between projects
- Taking account of dependencies between projects
- Removing duplication between projects
- Checking for gaps

There are three elements to PPM:

1. **Project portfolio definition**
   - Create a central record of all projects within an organization
   - Must decide whether to have ALL projects in the repository or, say, only ICT projects
   - Note difference between new product development (NPD) projects and renewal projects e.g. for process improvement
2. **Project portfolio management**

   Actual costing and performance of projects can be recorded and assessed
3. **Project portfolio optimization**

Information gathered above can be used achieve better balance of projects e.g. some that are risky but potentially very valuable balanced by less risky but less valuable projects

You may want to allow some work to be done outside the portfolio e.g. quick fixes

Cost benefit analysis (CBA)

This relates to an individual project. You need to:

- Identify all the costs which could be:

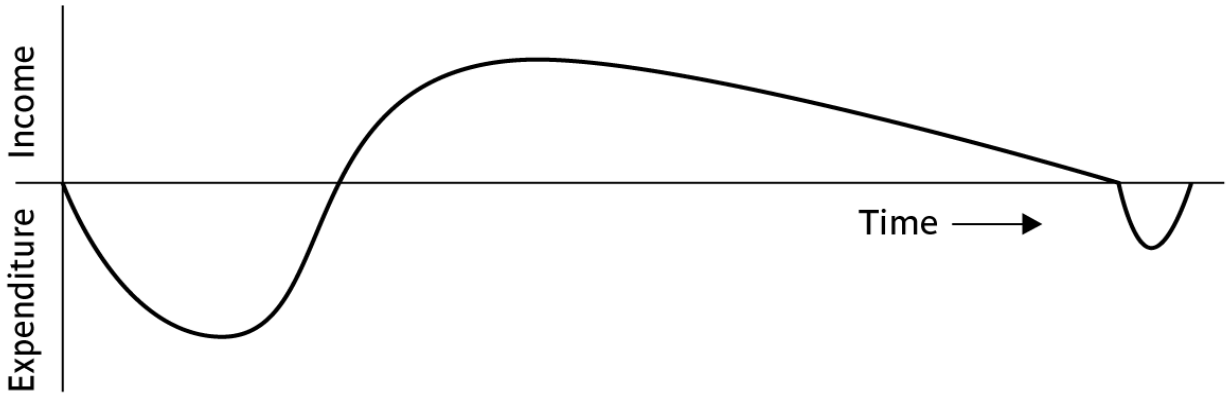
- Development costs
- Set-up
- Operational costs

- Identify the value of benefits
- Check benefits are greater than costs

Section 2.4 of the text expands this material.

Exercise 2.1 requires students to identify the potential costs and benefits of the Brightmouth College payroll application.

Product/system life cycle cash flows



- The timing of costs and income for a product of system needs to be estimated.
- The development of the project will incur costs.
- When the system or product is released it will generate income that gradually pays off costs
- Some costs may relate to decommissioning – think of demolishing a nuclear power station.

Net profit

Year	Cash-flow
0	-100,000
1	10,000
2	10,000
3	10,000
4	20,000
5	100,000
Net profit	50,000

'Year 0' represents all the costs before system is operation

'Cash-flow' is value of income less outgoing

Net profit value of all the cash-flows for the lifetime of the application

See Section 2.5 for further details. Exercise 2.3 is applicable here

Pay back period

This is the time it takes to start generating a surplus of income over outgoings. What would it be below?

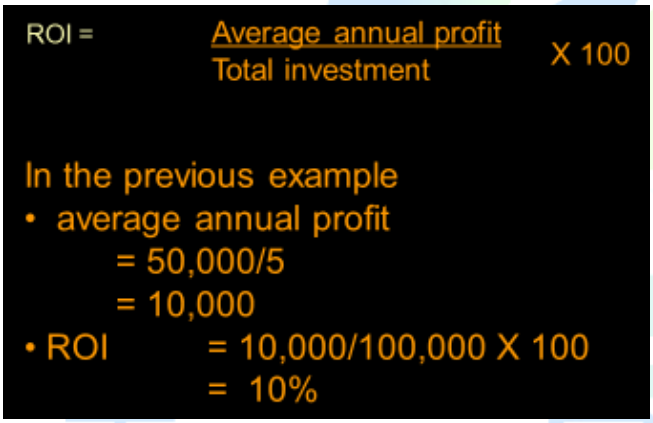
Year	Cash-flow	Accumulated
0	**-100,000**	**-100,000**
1	**10,000**	**-90,000**
2	**10,000**	**-80,000**

3	**10,000**	**-70,000**
4	**20,000**	**-50,000**
5	**100,000**	**50,000**

The payback period would be about 4.5 years. This can be calculated as the last year in which the accumulated cash flow was negative + (absolute accumulated cash flow at the end of that year / cash-flow for the next year) e.g. year 4 + (50,000/100,000). This assumes that the flow of cash is constant throughout the year in question e.g. £100,000/12 or £8,333 a month in year 5

Exercise 2.3. in the text is relevant here.

Return on investment (ROI)



Exercise 2.4. gives further practice is calculating ROI.

Net present value

Would you rather I gave you £100 today or in 12 months time?

If I gave you £100 now you *could* put it in savings account and get interest on it.

If the interest rate was 10% how much would I have to invest now to get £100 in a year's time?

This figure is the *net present value* of £100 in one year's time

If you invested £91 now you would get £9.10 in interest which would give you £100.10 in 12 months. The interest rate of 10% is used purely to make it easy to do the calculations, not because it is a realistic rate.

Discount factor

Discount factor = $1/(1+r)^t$

r is the interest rate (e.g. 10% is 0.10)

t is the number of years

In the case of 10% rate and one year

Discount factor = 1/(1+0.10) = 0.9091

In the case of 10% rate and two years

Discount factor = 1/(1.10 x 1.10) =0.8294

Applying discount factors

Year	Cash-flow	Discount factor	Discounted cash flow
0	-100,000	1.0000	-100,000
1	10,000	0.9091	9,091
2	10,000	0.8264	8,264
3	10,000	0.7513	7,513
4	20,000	0.6830	13,660
5	100,000	0.6209	62,090
		NPV	618

NPV is the sum of the discounted cash flows for all the years of the 'project' (note that in NPV terms the lifetime of the completed application is included in the 'project')

The figure of £618 means that £618 more would be made than if the money were simply invested at 10%. An NPV of £0 would be the same amount of profit as would be generated by investing at 10%.

Internal rate of return

- Internal rate of return (IRR) is the discount rate that would produce an NPV of 0 for the project
- Can be used to compare different investment opportunities
- There is a Microsoft Excel function which can be used to calculate

The Excel function in question is =IRR.

Dealing with uncertainty: Risk evaluation

- project A might appear to give a better return than B but could be riskier
- Draw up a project risk matrix for each project to assess risks – see next overhead
- For riskier projects could use higher discount rates

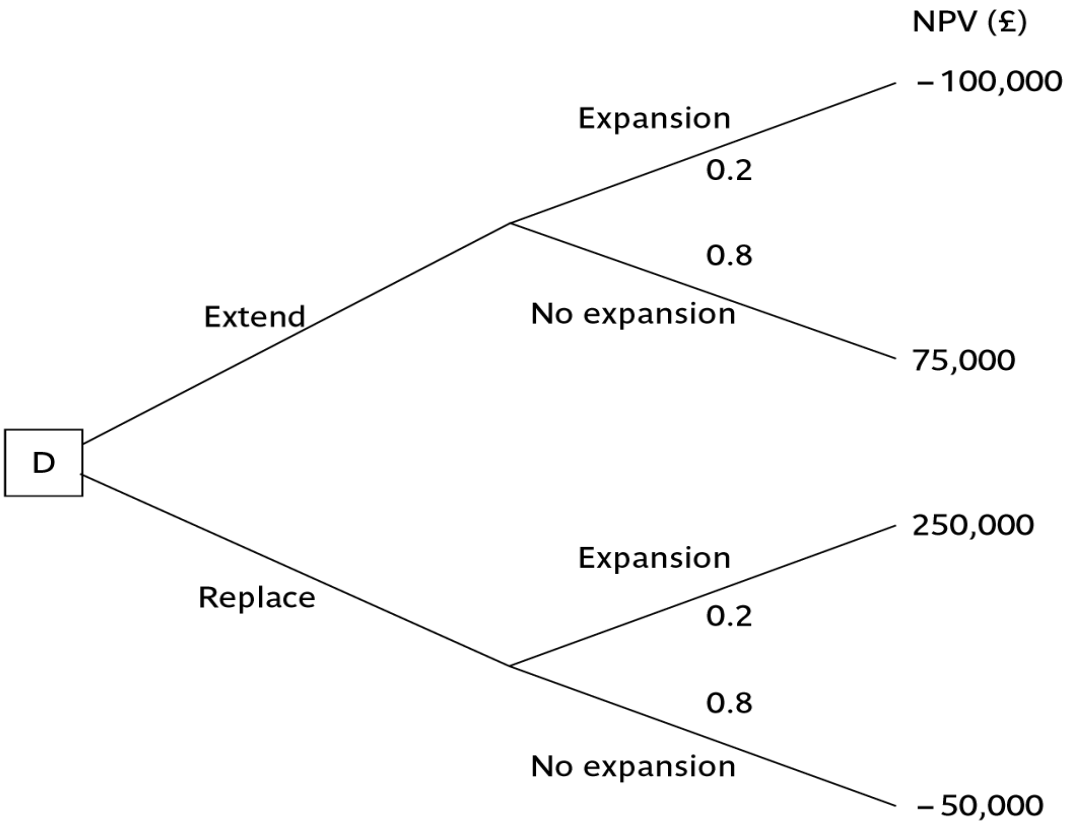
Example of a project risk matrix

Risk	Importance	Likelihood
Client rejects proposed look and feel of site	H	—
Competitors undercut prices	H	M
Warehouse unable to deal with increased demand	M	L
Online payment has security problems	M	M
Maintenance costs higher than estimated	L	L
Response times deter purchasers	M	M

TABLE 2.5 A fragment of a basic project/business risk matrix for an e-commerce application

In the table 'Importance' relates to the cost of the damage if the risk were to materialize and 'likelihood' to the probability that the risk will actual occur. 'H' indicates 'High', 'M' indicates 'medium' and 'L' indicates 'low'. The issues of risk analysis are explored in much more depth in chapter 7.

Decision trees



The diagram here is figure 2.2 in the text.

This illustrates a scenario that could relate to the IOE case study. Say Amanda is responsible for extending the invoicing system. An alternative would be to replace the whole of the system. The decision is influenced by the likelihood of IOE expanding their market. There is a strong rumour that they could benefit from their main competitor going out of business: in this case they could pick up a huge amount of new business, but the invoicing system could not cope. However replacing the system immediately would mean other important projects would have to be delayed.

The NPV of extending the invoicing system is assessed as £75,000 if there is no sudden expansion. If there were a sudden expansion then there would be a loss of £100,000. If the whole system were replaced and there was a large expansion there would be a NPV of £250,000 due to the benefits of being able to handle increased sales. If sales did not increase then the NPV would be -£50,000.

The decision tree shows these possible outcomes and also shows the estimated probability of each outcome.

The value of each outcome is the NPV multiplied by the probability of its occurring. The value of a path that springs from a particular decision is the sum of the values of the possible outcomes from that decision. If it is decided to extend the system the sum of the values of the outcomes is £40,000 (75,000 x 0.8 – 100,000 x 0.2) while for replacement it would be £10,000 (250,000 x 0.2 – 50,000 x 0.80). Extending the system therefore seems to be the best bet (but it is still a bet!).

Programme management

- **One definition:**

'a group of projects that are managed in a co-ordinated way to gain benefits that would not be possible were the projects to be managed independently' Ferns

The quotation is from a paper that appeared in the International Journal of Project Management August 1991

Programmes may be

- Strategic
- Business cycle programmes
- Infrastructure programmes
- Research and development programmes
- Innovative partnerships

- See Section 2.7
- ***Strategic***
- Several projects together implement a single strategy. For example, merging two organizations will involve many different activities e.g. physical re-organization of offices, redesigning the corporate image, merging ICT systems etc. Each of these activities could be project within an overarching programme.
- ***Business cycle programmes***

- A portfolio of project that are to take place within a certain time frame e.g. the next financial year
- ***Infrastructure programmes***
- In an organization there may be many different ICT-based applications which share the same hardware/software infrastructure
- ***Research and development programmes***
- In a very innovative environment where new products are being developed, a range of products could be developed some of which are very speculative and high-risk but potentially very profitable and some will have a lower risk but will return a lower profit. Getting the right balance would be key to the organization's long term success
- ***Innovative partnerships***
- e.g. pre-competitive co-operation to develop new technologies that could be exploited by a whole range of companies

Programme managers versus project managers

Programme manager	Project manager
Many simultaneous projects	One project at a time
Personal relationship with skilled resources	Impersonal relationship with resources
Optimization of resource use	Minimization of demand for resources
Projects tend to be seen as similar	Projects tend to be seen as unique

The programme manager may well have a pool of staff upon which to call. He/she will be concerned with ensuring the best use of staff e.g ensuring that staff have regular work with no periods of enforced idleness between project tasks. The project leader would think in terms of 'I need a Java programmer for four weeks' without being concerned which specific person it is (beyond obvious concerns that they are fully capable).

Strategic programmes

- Based on OGC approach
- Initial planning document is the **Programme Mandate** describing
  - The new services/capabilities that the programme should deliver
  - How an organization will be improved
  - Fit with existing organizatioal goals
- A **programme director** appointed a champion for the scheme

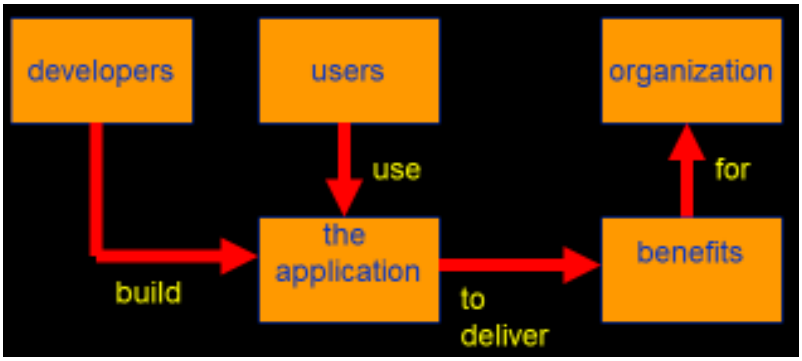
- *The material here is based on the UK government's Office of Government Commerce (OGC) approach which is described in detail in their publication Managing successful programmes.*
- *The programme director should be someone who is in a prominent position in the organization so that the seriousness and commitment of the organization to the programme are made clear.*
- *An example of what might be a programme is given in Section 2.9 in the text. It might be found at the IOE company that the customers' experience of the organization can be very variable and inconsistent. The employee who records the customer's requirements is different from the people who actually carry out the work and different again from the clerk who deals with accounts. Different maintenance engineers deal with different types of equipment. A business objective might be to present a consistent and uniform interface to the client. This objective might need changes to a number of different systems which until now have been largely self-contained. The work to reorganize each individual area might be treated as separate projects, co-ordinated at a higher level as a programme.*

Next stages/documents

- **The programme brief** – equivalent of a feasibility study: emphasis on costs and benefits
- **The vision statement** – explains the new capability that the organization will have
- **The blueprint** – explains the changes to be made to obtain the new capability

- The programme brief – is it worth it?
- The vision statement – the 'what'
- The blueprint – the 'how'

Benefits management



- Providing an organization with a capability does not guarantee that this will provide benefits envisaged – need for *benefits management*
- This has to be outside the project – project will have been completed

- Therefore done at *programme level*

To carry this out, you must:

- Define expected benefits
- Analyse balance between costs and benefits
- Plan how benefits will be achieved
- Allocate responsibilities for their achievement
- Monitor achievement of benefits

- ***Benefit profiles*** *can be produced that document when and how it is planned that the benefits will be experienced.*
- *As different components of the new capability are developed, a series of* ***tranches*** *of projects (projects grouped in different steps of the programme) may be completed, each with a set of associated benefits.*
- *The achievement of benefits might be made the responsibility of staff who are designated as* ***business change managers.***

Benefits

These might include:

- Mandatory requirement
- Improved quality of service

- Increased productivity
- More motivated workforce
- Internal management benefits

- *You could argue that as you have to comply with a mandatory requirement, the question of benefits is irrelevant in this case. However as failure to comply will a negative outcome (e.g. not being able to trade), avoiding that negative outcome is clearly a benefit which could be costed.*
- *'Internal management benefits' includes things like better decision-making. In the case of an insurance company a deeper analysis of insurance claims might help identify types of business that are most risky and allow the company to adjust premiums to cover these.*

- Risk reduction
- Economies
- Revenue enhancement/acceleration
- Strategic fit

- *'Economies' refers to cost-cutting e.g. using an automated telephone system to direct calls without human intervention could allow an organization to reduce staff.*
- *Revenue enhancement/acceleration e.g. the sooner that bills reach the customers, the sooner they can pay them.*
- *'Strategic fit' A change might not benefit any single group within an organization but might have to be made to obtain a benefit for the organization as a whole.*

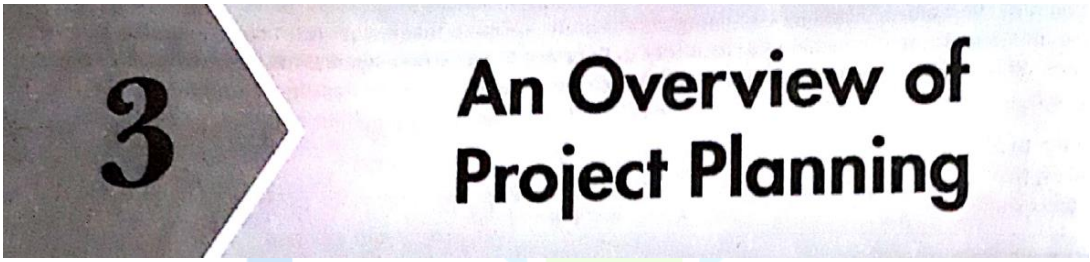
Quantifying benefits

Benefits can be:

- Quantified and valued e.g. a reduction of *x* staff saving *£y*
- Quantified but not valued e.g. a decrease in customer complaints by *x%*
- Identified but not easily quantified – e.g. public approval for a organization in the locality where it is based

Remember!

- A project may fail not through poor management but because it should never have been started
- A project may make a profit, but it may be possible to do something else that makes even more profit
- A real problem is that it is often not possible to express benefits in accurate financial terms
- Projects with the highest potential returns are often the most risky



'Step Wise' – aspirations

- Practicality
  - tries to answer the question 'what do I do now?'
- Scalability
  - useful for small project as well as large
- Range of application
- Accepted techniques
  - e.g. borrowed from PRINCE etc

The motivation for identifying an overall framework was that students, and others, were often at a loss as to where to start when new to project planning. A structured approach was seen as catering for the needs of such people.

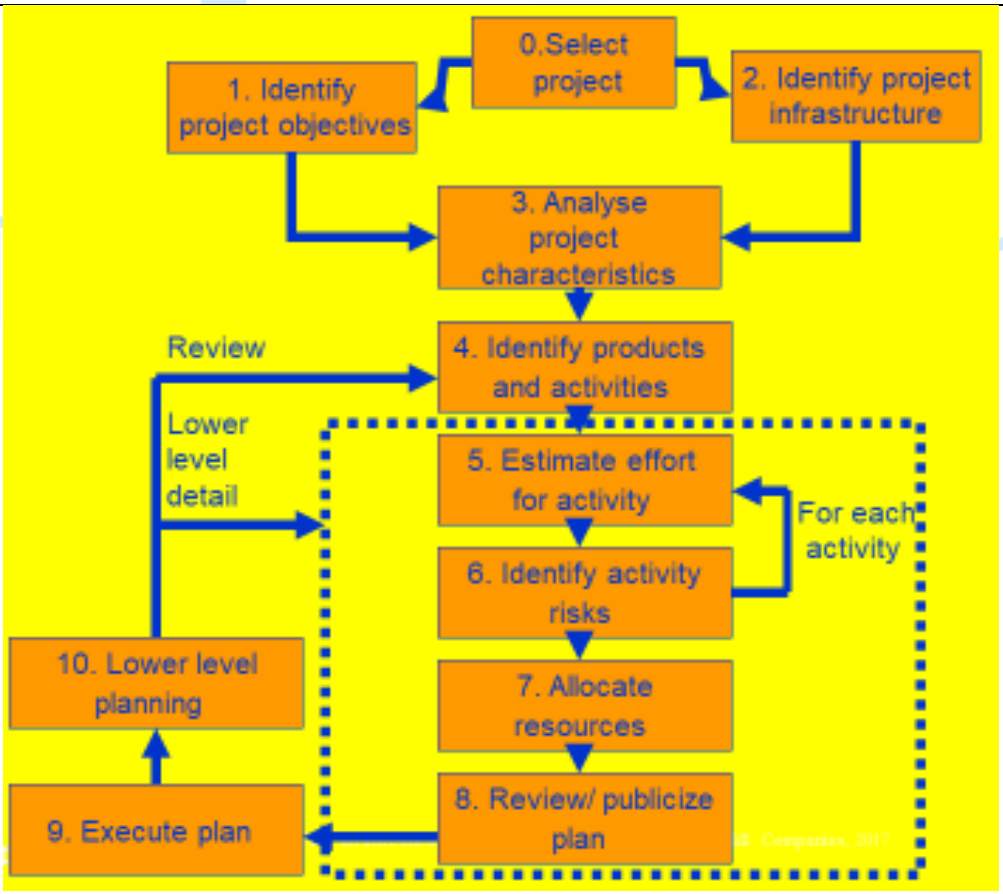
Students are perhaps most likely to come into contact with some kind of project planning (i) when they are involved in student projects, especially those carried in groups and (ii) if they are members of a project team when they are undertaking a placement year in industry. The first situation is probably

quite small-scale compared to the second. The same general principles of planning however relate to both.

The approach described here is designed to be applicable to a range of different types of project. For example, multimedia projects are not explicitly discussed, but one of the authors teaches a course project management for multimedia and the general approach described here seems to work satisfactorily when applied to these type of projects.

It might be asked why a standard approach such as PRINCE2 has not been adopted. In fact there is an outline of the PRINCE2 framework in an Appendix to the textbook. There has been caution about using PRINCE2 more centrally because:

- *PRINCE2 tend to be used mainly in the UK and many users of the Software Project Management textbook are from elsewhere*
- *The content of this course would be vulnerable to changes to the method imposed by its design authority*
- *PRINCE2 tends to focus more on procedural and bureaucratic matters at the expense of techniques – the few planning techniques that are associated with PRINCE, for example, the development of product flow diagrams, are used in the Step Wise approach as well.*



This is an overview of the main steps:

0. Select project There must be some process by which the project to be executed was selected. Chapter 3 on project evaluation looks at this in more detail.

1. Identify project objectives It is important that at the outset the main stakeholders are all aware of the precise objectives of the project. This has already been discussed in Chapter 1.

2. Identify project infrastructure This may not be a significant step where you are working on an in-house project in a very familiar environment. However, where the project is being carried out for external clients then you may need to investigate the characteristics of the environment in which the project is to be carried out.

3. Analyse project characteristics Different types of project will need different technical and management approaches. For example, a project to implement control software embedded in industrial equipment will need a different set of methods than a project to implement a business information system. A multimedia application would again need a different set of activities. (This is not to say that there could not be considerable overlaps in the approaches).

4. Identify products and activities With software projects, it is best to start by listing the products, both deliverable and intermediate, to be created. The activities needed to create the products can then be identified

5. Estimate effort for activity.

6. Identify activity risks Having assessed the amount of effort and the elapsed time for a project, the reasons why these might be vary during the actual execution of the project need to be considered. Where there is a very high risk of additional effort/time being needed then actions to reduce this risk may be formulated.

7. Allocate resources With software projects, these resources will mainly be staff, but could be equipment etc.

8. Review/publicize It is no good having a plan if no one knows about it

9. Execute Plan

10. Lower level planning Not all of a project, especially when it is large, can be planned in detail at the outset. Not all the information needed to plan the later stages will be available at the beginning: for example software development cannot be broken down into precise sub-tasks with realistic target times until more is known about what the overall design of the system is known.

A project scenario: Brightmouth College Payroll

- College currently has payroll processing carried out by a services company
- This is very expensive and does not allow detailed analysis of personnel data to be carried out
- Decision made to bring payroll 'in-house' by acquiring an 'off-the-shelf' application
- The use of the off-the-shelf system will require a new, internal, payroll office to be set up
- There will be a need to develop some software 'add-ons': one will take payroll data and combine it with time-table data to calculate the staff costs for each course run in the college
- The project manager is Brigette.

The last requirement – to build an 'add-on' – allows some software development issues to be addressed!

Step 1 establish project scope and objectives

- 1.1 Identify objectives and measures of effectiveness
  - 'how do we know if we have succeeded?'
- 1.2 Establish a project authority
  - 'who is the boss?'
- 1.3 Identify all stakeholders in the project and their interests
  - 'who will be affected/involved in the project?'

- ***1.1. Identifying objectives and measures of effectiveness.*** *This was discussed in chapter1. Key points are that the student project objectives must be such that a student can realistically be responsible for their achievement. For instance, an objective to reduce conflict between project team members would be at too high a level for a software developer: he or she is there to produce software and evaluation of the particular pyschometric test would be outside their capabilities. If the student was a psychology student and the project was regarded as a psychological one, then things might be different.*

- ***1.2.Establishment of a project authority*** *In the case of students on placement or carrying final year projects for outside organizations, the problem of identifying who has the final say on the project can occur surprisingly often, particular when different user groups have conflicting requirements. In larger, more formal projects, the project authority might reside in a Project Board or steering committee.*
- ***1.3 Identify all stakeholders in the project and their interests****. Stakeholders can be anyone who has an interest in the project. They may be users of the final application or might be involved in the development or implementation of the project.*

- 1.4 Modify objectives in the light of stakeholder analysis
  - 'do we need to do things to win over stakeholders?'
- 1.5 Establish methods of communication with all parties
  - 'how do we keep in contact?'

- ***1.4 Modify objectives in the light of stakeholder analysis.*** *The key point here is the need to ensure commitment to the project from the important stakeholders. This might need to be done by ensuring that there is some benefit from the project for them. Note this is a similar idea to Barry Boehm's 'Theory W' ('W' stands for 'everyone a* ***w****inner')*
- ***1.5 Establish methods of communication with all parties*** *In the case of a small student project, it might be mainly swapping email addresses and mobile phone numbers. With larger projects, it could involve setting up groups who meet regularly to co-ordinate action. Sometimes specific 'communication plans' are drawn up to deal with these issues.*

Back to the scenario

- Project authority
  - Brigette finds she has two different clients for the new system: the finance department and the personnel office. A vice principal agrees to be official client, and monthly meetings are chaired by the VP and attended by Brigette and the heads of finance and personnel
  - These meetings would also help overcome communication barriers

Applying these ideas to the scenario introduced earlier:

- ***Project authority***

- ***Stakeholders/revision to objectives*** *The application will not ultimately be a success if project team members are not happy to use the system. They might be happier to use the testing system if the results of their own tests were automatically notified to them personally by the software application, so that this might have to be added as a requirement for the project.*

- Stakeholders
  - For example, personnel office would supply details of new staff, leavers and changes (e.g. promotions)
  - To motivate co-operation Brigette might ensure new payroll system produces reports that are useful to personnel staff

Step 2 Establish project infrastructure

- 2.1 Establish link between project and any strategic plan
  - 'why did they want the project?'
- 2.2 Identify installation standards and procedures
  - 'what standards do we have to follow?'
- 2.3. Identify project team organization
  - 'where do I fit in?'

- *At the same time as establishing exactly what the project objectives are, the person responsible may know little about the organizational environment in which the application is to be developed and implemented. The actions in Step 2 address this problem.*

Step 3 Analysis of project characteristics

- 3.1 Distinguish the project as either objective or product-based.
  - Is there more than one way of achieving success?
- 3.2 Analyse other project characteristics (including quality based ones)
  - what is different about this project?

- *Step 3 is about examining the nature of the application to be built and the environment in which it is to be built and implemented and identifying the most appropriate technical approach.*
- ***3.1 Objective-based versus product-based projects**. With a product-based project the developers have to create a product, the specification of which is often (but not always) clearly defined. In an objective-based project, a problem is defined that needs to be solved but there could be more than one solution. For example, if an organization needed a payroll application they might consider (a) writing the system themselves (b) using a service company to do the payroll for them (c) acquire an off-the-shelf package*
- ***3.2 Analyse other project characteristics – such as is it an information system or an embedded real time or a multimedia application? Is it safety-critical? Etc etc.** The payroll application is clearly an information system. If an off-the-shelf application is adopted, there is plenty of guidance on how off-the-shelf applications can be accessed and selected that could be consulted by Brigette*

- Identify high level project risks
  - 'what could go wrong?'
  - 'what can we do to stop it?'
- Take into account user requirements concerning implementation
- Select general life cycle approach
  - waterfall? Increments? Prototypes?
- Review overall resource estimates
  - 'does all this increase the cost?'

- *Identifying high level risks could influence the general approach to the project. For example, if the users appeared to be uncertain about the precise nature of the requirement then a more iterative approach, including the use of prototypes to refine user needs, might be selected.*
- *If the application is very large and complex then breaking it down into increments might be the way to proceed. Chapter/lecture 4 looks at this in more detail.*

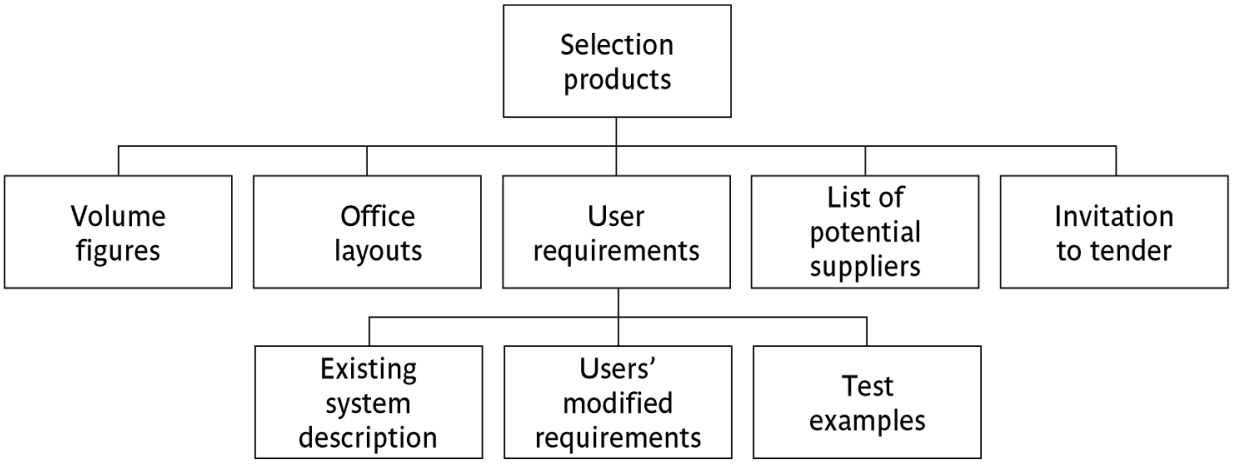
Back to the scenario

- Objectives vs. products
  - An objective-based approach has been adopted

- Some risks
  - There may not be an off-the-shelf package that caters for the way payroll is processed at Brightmouth College
- Answer?
  - Brigette decides to obtain details of how main candidate packages work as soon as possible; also agreement that if necessary processes will be changed to fit in with new system.

Step 4 Identify project products and activities

- **4.1 Identify and describe project products - 'what do we have to produce?'**



Here we follow the PRINCE approach of firstly identifying the products to be created. These products could be deliverables that will eventually be handed over to the customer, or intermediate products such as specifications and design documents, that are produced along the way.

The PBS is a way of listing these products.

*In the scenario, one set of products will relate to the products needed to produce one or more invitations to tender (ITTs) to supply the hardware and software needed to operate the new payroll application. In order to allow the most suitable configuration to be identified the number of transactions and the size of the database needed will have to be identified – **volume figures.** To set up an appropriate network attached to secure printers and servers, a **layout of the proposed office** will need to be created. A **user requirement** will need to be produced which describes the existing system, identifies additional requirements (such as the need to be able to access database details in order to produce one-off queries and reports), and some **test data and expected results** which further*

*illuminate the details of the requirements. This test data could form the basis of user acceptance tests. A **list of potential suppliers** to whom ITTs could be sent will be needed, and the actual ITT itself which will, among other things, explain how the proposals of potential suppliers are to be submitted.*

Products

- The result of an activity
- Could be (among other things)
  - physical thing ('installed pc'),
  - a document ('logical data structure')
  - a person ('trained user')
  - a new version of an old product ('updated software')
- The following are NOT normally products:
  - activities (e.g. 'training')
  - events (e.g. 'interviews completed')
  - resources and actors (e.g. 'software developer') - may be exceptions to this
- Products CAN BE *deliverable* or *intermediate*

Product description (PD)

- Product identity
- Description - what is it?
- Derivation - what is it based on?
- Composition - what does it contain?
- Format
- Relevant standards
- Quality criteria

Create a PD for 'test data'

The names of products on the PBS can be rather vague. If you were to ask some one to produce, for example, the 'analysis report' in the usability testing scenario, then you would need to explain exactly what you mean by that. This is done via a Product Description. PDs can usually be re-used from one project to another.

Note that they are different from specifications – the explain in general terms what a product is and the description is relevant to all instances of that product. A specification describes a particular instance within the class of products.

4.2 document generic product flows

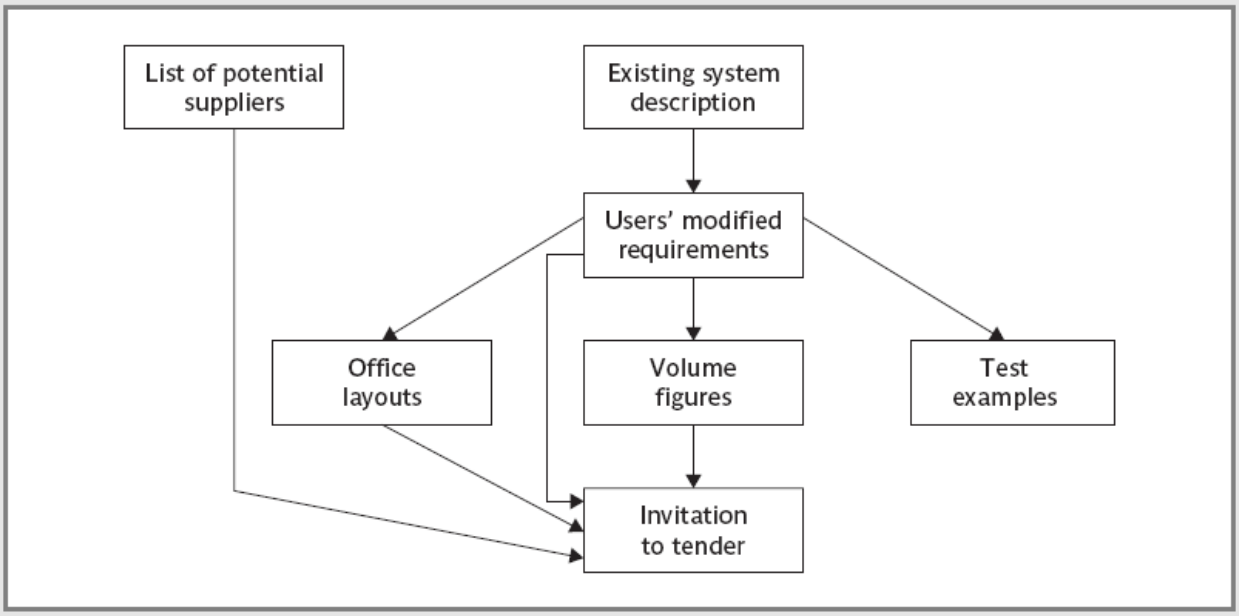


FIGURE B.1 Product Flow Diagram for the creation of an 'invitation to tender'

The product flow diagram shows the order in which the products have to be completed. Effectively it defines a method of working. The example above is a possible solution to Exercise 3.3 in the textbook.

*The flow of the PFD is generally from top to bottom and left to right. We do no put in lines which loop back. This is not because iterative and back-tracking is not accepted. Rather it is that you can in theory jump back to **any** preceding product.*

Step 4.3 Recognize product instances

- The PBS and PFD will probably have identified generic products e.g. 'software modules'
- It might be possible to identify specific instances e.g. 'module A', 'module B' …
- But in many cases this will have to be left to later, more detailed, planning

Step 4.4. Produce ideal activity network

- Identify the activities needed to create each product in the PFD
- More than one activity might be needed to create a single product
- Hint: Identify activities by verb + noun but avoid 'produce…' (too vague)
- Draw up activity network

An 'ideal' activity

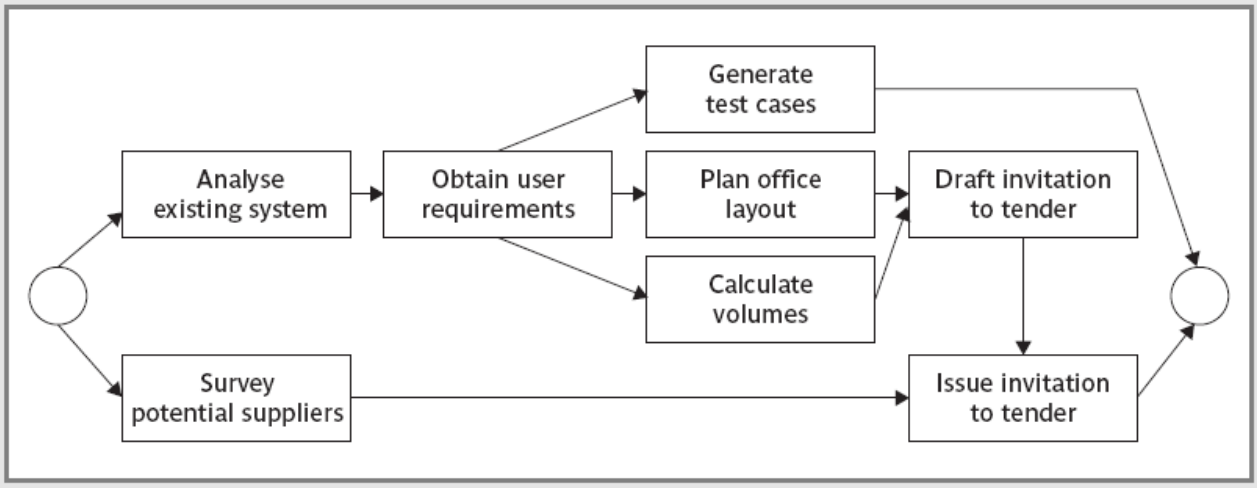
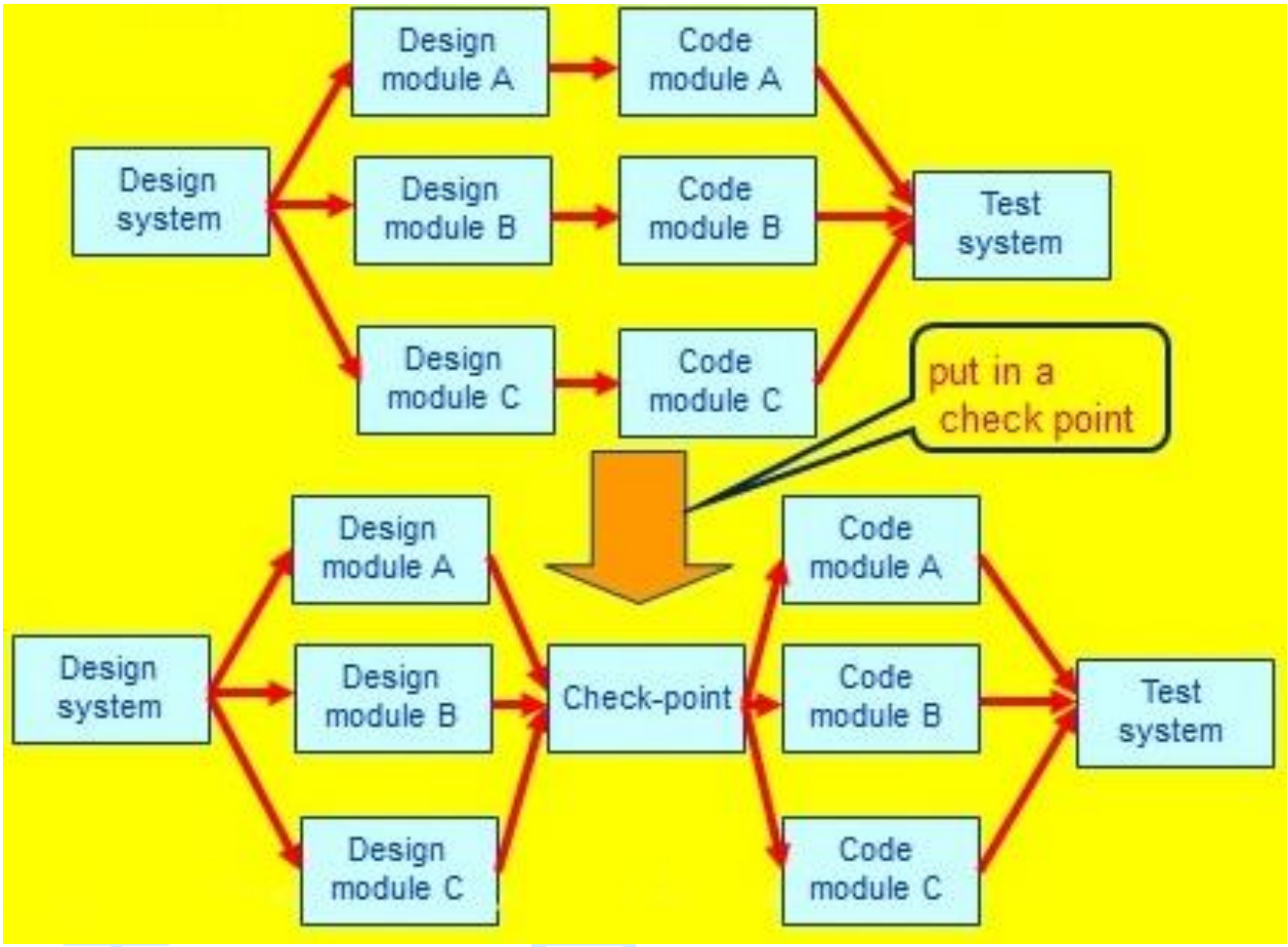


FIGURE B.2 Brightmouth College payroll project activity network fragment

The activity network is the basis of the data that is input to planning software tools like MS Project.

Step 4.5 Add check-points if needed



*There are some points in the project when we want to check that the quality of what we have done so far is a sound basis for further work. In the example we have decided to check that all the module designs are compatible with one another before continuing. Note that the benefit of reducing wasted work at a later stage when incompatibilities lead to products being reworked, has to be balanced against the delay caused by the check-point. The start of coding of modules A, B and C **all** have to wait for the completion of the design of **all** the modules A, B and C. With no check-point, module A could be coded as soon as the design of module A had been done without having to wait for B and C.*

Step 5:Estimate effort for each activity

- 5.1 Carry out bottom-up estimates
  - distinguish carefully between *effort* and *elapsed* time
- 5.2. Revise plan to create controllable activities

- break up very long activities into a series of smaller ones
- bundle up very short activities (create check lists?)

- *Effort is the total number of staff-hours (or days etc) needed to complete a task. Elapsed time is the calendar time between the time task starts and when it ends. If 2 people work on the same task for 5 days without any interruption, then the effort is 10 staff-days and the elapsed time is 5 days.*
- *Using the PBS/PFD to generate the activities often means that some activities are very small and others are huge. Often there is an activity called 'write software' which is 70% of a software development project. These large activities need to be broken down into more manageable small tasks. You should aim for the average length of your activities to be about the time between progress meetings e.g. if you have team progress meeting once a fortnight, try to make the tasks last about 2 weeks.*

Step 6: Identify activity risks

- 6.1.Identify and quantify risks for activities
  - damage if risk occurs (measure in time lost or money)
  - likelihood if risk occurring
- 6.2. Plan risk reduction and contingency measures
  - risk reduction: activity to stop risk occurring
  - contingency: action if risk does occur

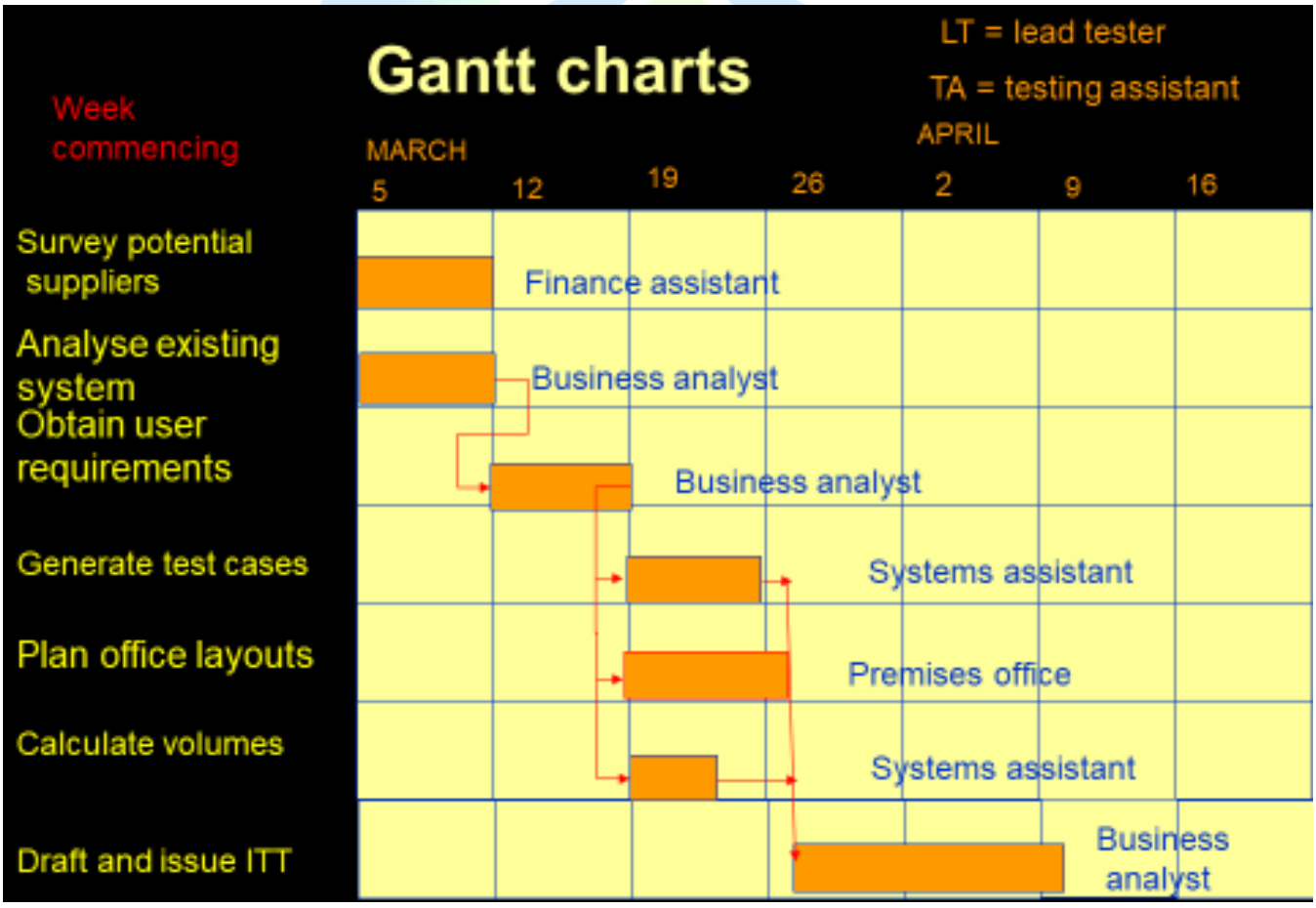
Note that we have already considered some high level risks that could affect the project as a whole at Step 3. Estimates of the effort and duration of activities can be quite tricky. When you produce an estimate for an activity it is worth reflecting about the events which could cause the assumptions upon which you have based your estimate to be wrong. Where the risk seems very high, then you might try to introduce new activities specifically designed to reduce the risk, or to formulate a contingency action if the risk should materialize. For example, there is not much you can do to stop people getting the flu at a critical time in the project, but you might be able to have a plan to bring in temporary cover for sickness in the case of time-critical activities.

- 6.3 Adjust overall plans and estimates to take account of risks
  - e.g. add new activities which reduce risks associated with other activities e.g. training, pilot trials, information gathering

Step 7: Allocate resources

- 7.1 Identify and allocate resources to activities
- 7.2 Revise plans and estimates to take into account resource constraints
  - e.g. staff not being available until a later date
  - non-project activities

You now need to allocate resources (in particular, staff) to the activities in the plan. Where there is a resource constraint, that is there are not enough staff (or other resource) of the right type to start all the activities that run in parallel at the planned time, then the start of some activities may need to be delayed until the appropriate resources are available.



We now have the basic information needed to produce a plan. One way of presenting the plan is by means of a Gantt chart (named after Henry Gantt).

Step 8: Review/publicise plan

- 8.1 Review quality aspects of project plan
- 8.2 Document plan and obtain agreement

Step 9 and 10: Execute plan and create lower level plans

We have noted already that it is not feasible to produce a detailed plan for all stages of the project right at the beginning of the project planning process and not all the information needed for the detailed planning of the later stages is available at the outset. Initially an outline plan for the whole project would be produced, plus a detailed plan for the first stage.

Key points

- Establish your objectives
- Think about the characteristics of the project
- Discover/set up the infrastructure to support the project (including standards)
- Identify **products** to be created and the **activities** that will create them
- Allocate resources
- Set up quality processes

Unit II

Learning Objective:

- Building versus buying software
- Taking account of the characteristics of the project
- Process models
  - Waterfall
  - Prototyping and iterative approaches
  - Incremental delivery
- Agile approaches

It can be argued that agile approaches are often a repackaging of prototyping and incremental delivery.

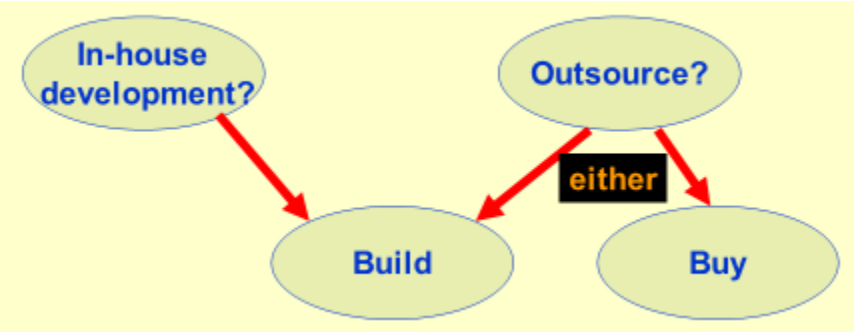
Selection of project approaches

- This part concerned with choosing the right approach to a particular project: variously called *technical planning, project analysis, methods engineering* and *methods tailoring*
- In-house: often the methods to be used dictated by organizational standards
- Suppliers: need for tailoring as different customers have different needs

Section 4.1 and 4.3 of the textbook discuss these issues. Different types of project need different types of approach. If you are working in one particular environment which specializes in one type of software, then the approach is likely not to change much from one project to another. Where you work

for different clients in different organizations developing a variety of applications then the approach for each project may need to be tailored.

Build or buy?



In-house development almost always involves developing new code.

If it is decided to use a specialist organization to implement system, the supplier could either build a 'bespoke' system for you, or could install a pre-existing application. There are hybrids of these options, e.g. to build in-house but use temporary contract staff, or Customised Off-the-Shelf (COTS) where a basic existing core system is modified for your particular requirements.

Some advantages of off-the-shelf (OTS) software

- Cheaper as supplier can spread development costs over a large number of customers
- Software already exists
  - Can be trialled by potential customer
  - No delay while software being developed
- Where there have been existing users, bugs are likely to have been found and eradicated

Some possible disadvantages of off-the-shelf

- Customer will have same application as everyone else: no competitive advantage, *but* competitive advantage may come from the *way* application is used
- Customer may need to change the way they work in order to fit in with OTS application
- Customer does not own the code and cannot change it
- Danger of over-reliance on a single supplier

One concern is what happens if the supplier goes out of business. Customer might not then be able to maintain system: hence the use of 'escrow' services where a 3rd party retains a copy of the code.

General approach

- Look at risks and uncertainties e.g.
  - are requirement well understood?
  - are technologies to be used well understood?
- Look at the type of application being built e.g.
  - information system? embedded system?
  - criticality? differences between target and development environments?
- Clients' own requirements
  - need to use a particular method

See section 4.3 of the textbook.

Structure versus speed of delivery

Structured approach

- Also called 'heavyweight' approaches
- Step-by-step methods where each step and intermediate product is carefully defined
- Emphasis on getting quality right first time
- Example: use of UML and USDP
- Future vision: Model-Driven Architecture (MDA). UML supplemented with Object Constraint Language, press the button and application code generated from the UML/OCL model

Agile methods

- Emphasis on speed of delivery rather than documentation
- RAD Rapid application development emphasized use of quickly developed prototypes

- JAD Joint application development. Requirements are identified and agreed in intensive workshops with users

Processes versus Process Models

- Starting from the inception stage:
  - A product undergoes a series of transformations through a few identifiable stages
  - Until it is fully developed and released to the customer.
  - This forms its life cycle or development process.
- Life cycle model (also called a process model):
  - A graphical or textual representation of the life cycle.

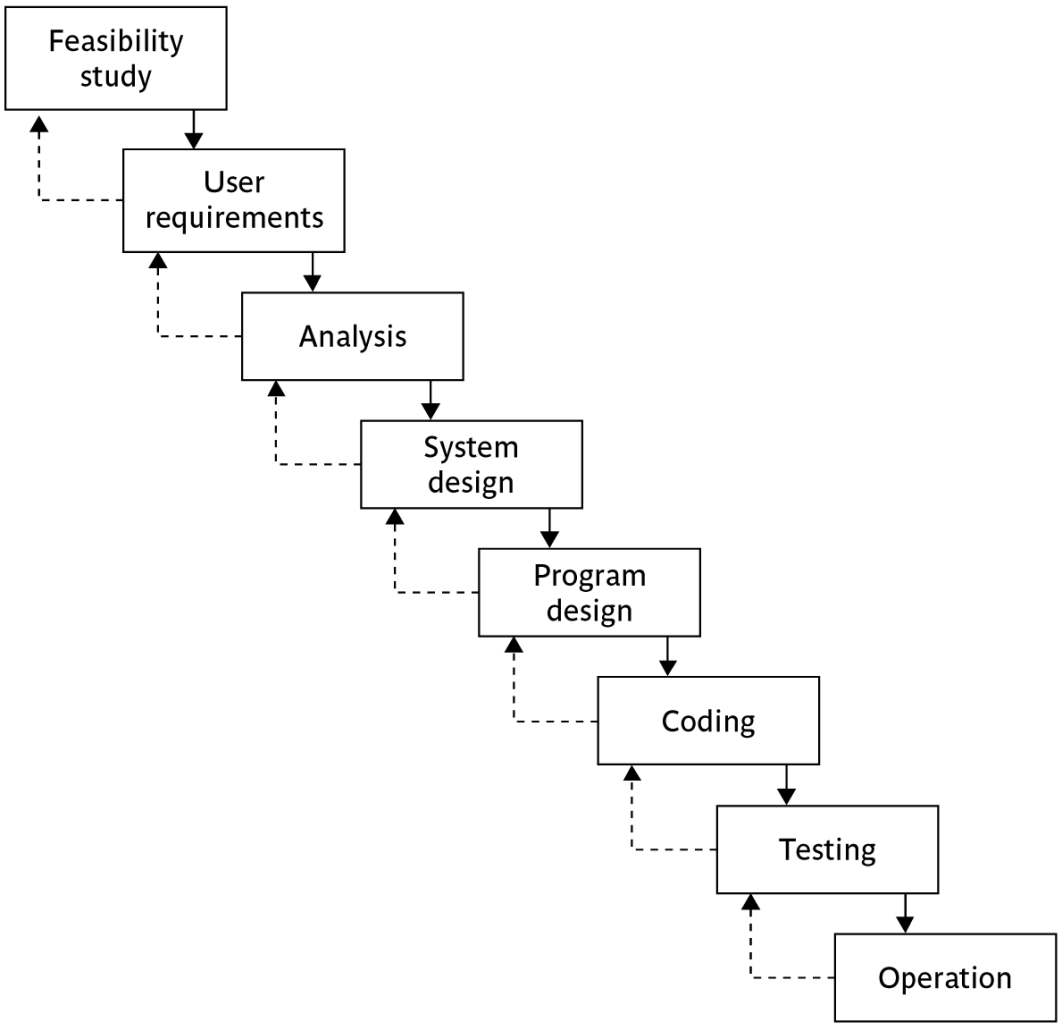
Choice of process models

- 'waterfall' also known as 'one-shot', 'once-through'
- incremental delivery
- evolutionary development

Also use of 'agile methods' e.g. extreme programming

It could be argued that extreme programming is mainly a particular way of carrying out incremental and evolutionary development

Waterfall



Section 4.6 of the textbook discusses this topic.

We have already touched upon the Waterfall approach in Chapter 1 when we discussed the ISO 12207. The way that the Waterfall approach is usually interpreted, it implies that all work on one phase has to be completed and checked off before the next one can start. However, it can be argued that ISO 12207 is really a technical model showing the order technical processes need to be carried out on a software component. You might break down an application into component increments, but the technical processes relating to that increment are carried out in the ISO 12207 sequence. You can, within the ISO 12207 sequence, loop back in an iterative manner, but the technical sequence still remains.

- the 'classical' model
- imposes structure on the project
- every stage needs to be checked and signed off

- BUT
  - limited scope for iteration
- V model approach is an extension of waterfall where different testing phases are identified which check the quality of different development phases

Evolutionary delivery: prototyping

‘ *An iterative process of creating quickly and inexpensively live and working models to test out requirements and assumptions’*

Sprague and McNurlin

main types

- ‘throw away’ prototypes
- evolutionary prototypes

what is being prototyped?

- human-computer interface
- functionality

Reasons for prototyping

- learning by doing
- improved communication
- improved user involvement
- a feedback loop is established
- reduces the need for documentation
- reduces maintenance costs i.e. changes after the application goes live
- prototype can be used for producing expected results

- *A prototype is a working model of one or more aspects of the project application. It is constructed quickly and inexpensively in order to test out assumptions.*

- *Sections 4.9, 4.10 and 4.11 discuss this topic.*
- *learning by doing - useful where requirements are only partially known*
- *improved communication - users are reluctant to read massive documents, but when system is 'live' you get a better feeling for it*
- *improved user involvement - user ideas and requests are quickly implemented*
- *the reduction of maintenance costs – the idea is that if you do not have a prototype then the first release of the application will effectively become a prototype as users will find things that they do not like and then ask for them to be changed. It is easier and safer to make such changes before the application becomes operational.*
- *testing - involves devising test cases and then documenting the results expected when the test cases are run. If these are done by hand, then effectively you have a manual prototype. Why not get the software prototype to generate the expected results?*

prototyping: some dangers

- users may misunderstand the role of the prototype
- lack of project control and standards possible
- additional expense of building prototype
- focus on user-friendly interface could be at expense of machine efficiency

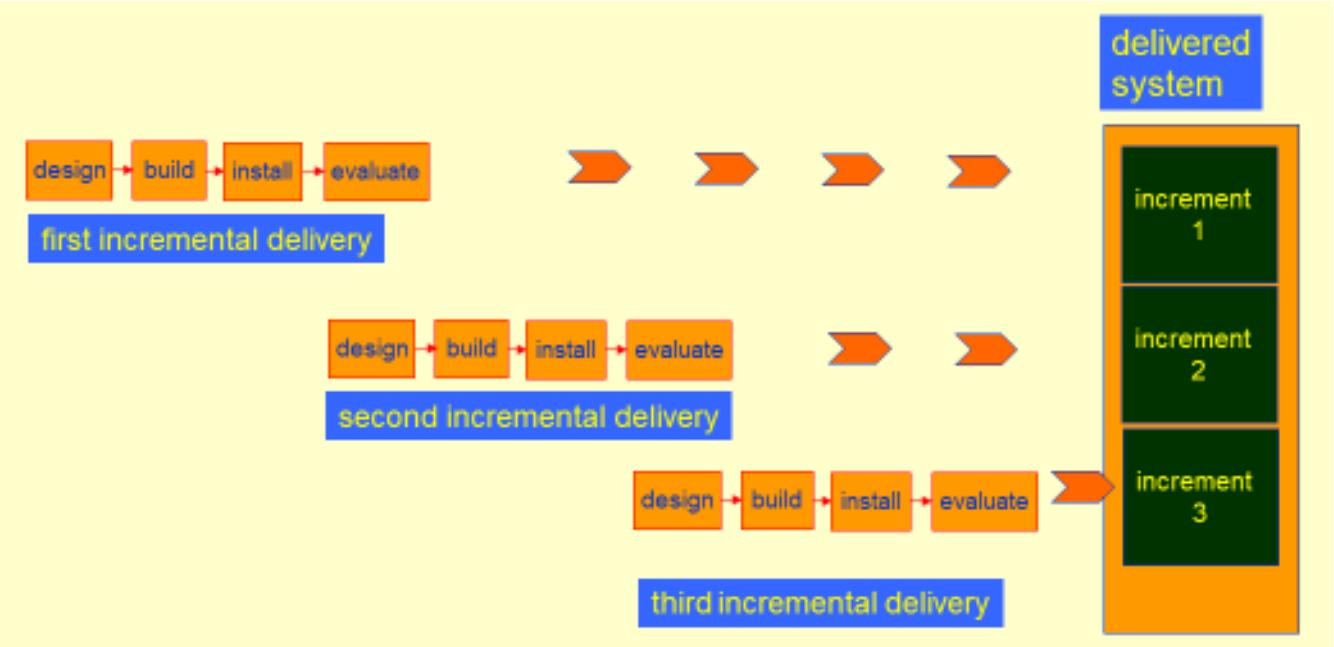
other ways of categorizing prototyping

- what is being learnt?
  - organizational prototype
  - hardware/software prototype ('experimental')
  - application prototype ('exploratory')
- to what extent
  - mock-ups
  - simulated interaction
  - partial working models: *vertical* versus *horizontal*

When a prototype is to be produced as part of a student's final year project, then the learning objectives that the prototypes will help be achieved need to defined at project inception. The details of

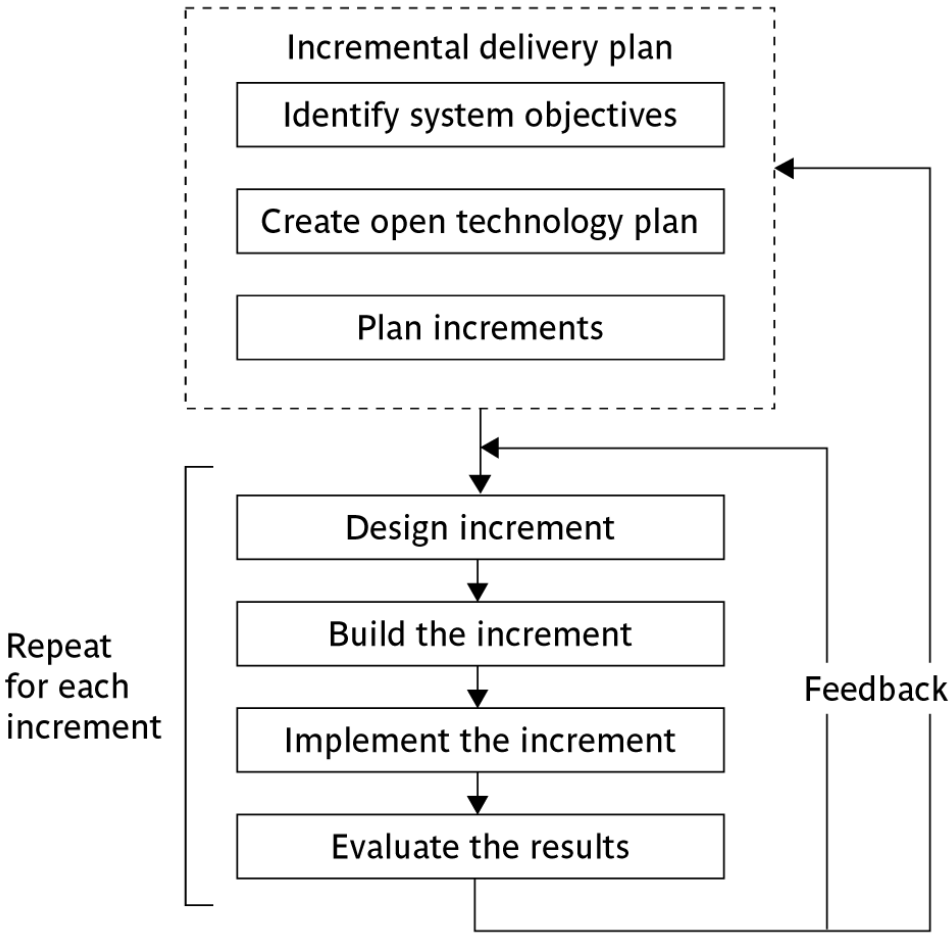
what has been learnt through the development and exercising of the prototype should be documented during the project.

Incremental delivery



The application to be delivered is broken down into a number of components each of which will be actually used by the users. Each of these is then developed as a separate 'mini-project' or increment.

The incremental process



This approach has had a very vocal advocate in Tom Gilb (see the book Principles of Software Engineering Management published by Addison-Wesley (1988)).

Gilb argues that the initial focus should be on high level business objectives. There may be many ways in which important objectives can be achieved and we ought to allow ourselves freedom in the way we try to achieve the objectives.

Open technology plan – we need to ensure that the technologies we use are ones that facilitate the addition of components to an existing application.

Plan increments – the nature and order of the increments to be delivered needs to be delivered to the users have to be planned at the outset.

Incremental approach: benefits

- feedback from early stages used in developing latter stages
- shorter development thresholds
- user gets some benefits earlier
- project may be put aside temporarily
- reduces 'gold-plating'

BUT there are some possible disadvantages

- loss of economy of scale
- 'software breakage'

Advantages of prototyping

- feedback from early stages used in developing latter stages
- shorter development thresholds - important when requirements are likely to change
- user gets some benefits earlier - may assist cash flow
- project may be put aside temporarily - more urgent jobs may emerge
- reduces 'gold-plating' i.e. features requested but not used
- *But there are possible disadvantages*
- loss of economy of scale - some costs will be repeated
- 'software breakage' - later increments might change earlier increments

Overview of incremental plan

- steps ideally 1% to 5% of the total project
- non-computer steps should be included
- ideal if a step takes one month or less:
  - not more than three months
- each step should deliver some benefit to the user
- some steps will be physically dependent on others

which step first?

- some steps will be pre-requisite because of physical dependencies
- others may be in any order
- value to cost ratios may be used
  - V/C where
  - V is a score 1-10 representing value to customer
  - C is a score 0-10 representing value to developers

V/C ratios: an example

step	value	cost	ratio	
profit reports	9	1	9	2nd
online database	1	9	0.11	5th
ad hoc enquiry	5	5	1	4th
purchasing plans	9	4	2.25	3rd
profit- based pay for managers	9	0	inf	1st

Genesis of 'Agile' methods

Structured development methods have several disadvantages

- produce large amounts of documentation which can largely remain unread
- documentation has to be kept up to date
- division into specialist groups and need to follow procedures stifles communication
- users can be excluded from decision process
- long lead times to deliver anything etc. etc

The answer? 'Agile' methods?

Agile Methods

- Agile is an umbrella term that refers to a group of development processes:
  - Crystal technologies
  - Atern (formerly DSDM)
  - Feature-driven development
  - Scrum
  - Extreme Programming (XP)
- Similar themes:
  - Some variations

Important Themes of Agile Methods

- Base on the incremental approach:
  - At a time, only one increment is planned, developed, and then deployed at the customer site.
- Agile model emphasizes face-to-face communication over written documents.
- An agile project usually includes a customer representative in the team.
- Agile development projects usually deploy pair programming.

Atern/Dynamic system development method (DSDM)

- UK-based consortium
- *arguably* DSDM can be seen as replacement for SSADM
- DSDM is more a project management approach than a development approach
- Can still use DFDs, LDSs etc!
- An update of DSDM has been badged as 'Atern'

A fuller explanation can be found in the DSDM Atern Pocket Book published by the Atern/DSDM Consortium.

SSADM is Structured Systems Analysis and Design Method a very heavy-weight and bureaucratic methodology that was promoted by the UK government

DFD = Data Flow Diagram

LDS = Logical Data Structure, effectively an Entity-Relationship Diagram

Six core Atern/DSDM principles

1. Focus on business need
2. Delivery on time – use of time-boxing
3. Collaborate
4. Never compromise qualitiy
5. Deliver iteratively
6. Build incrementally

This can be seen as a re-packaging of a lot of the ideas that have been discussed under the incremental and evolutionary approaches

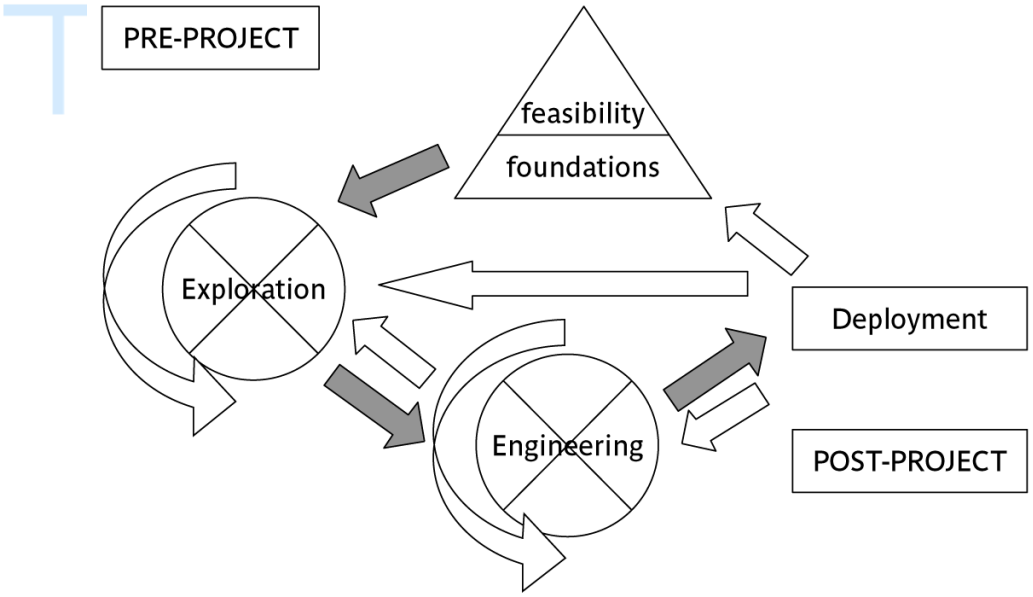


Fig. Atern/DSDM framework

The feasibility/business study stage will not only look at the business feasibility of the proposed project, but also as whether DSDM would be the best framework for it. Applications where there is a prominent user interface would be prime candidates.

Atern DSDM: time-boxing

- *time-box* fixed deadline by which *something* has to be delivered
- typically two to six weeks
- MOSCOW priorities
  - Must have - essential
  - Should have - very important, but system could operate without
  - Could have
  - Want - but probably won't get!

- Time-boxes mean that the focus moves from having a fixed set of functional requirements and then extending the planned project completion until they have all been developed. The deadline is fixed and we deliver what we have completed so far even if it is not everything that was originally planned.
- In order to make the delivered package coherent and as useful as possible to the users, requirements are prioritized according to the MoSCoW rules.

Extreme Programming Model

- Extreme programming (XP) was proposed by Kent Beck in 1999.
- The methodology got its name from the fact that:
  - Recommends taking the best practices to extreme levels.
  - If something is good, why not do it all the time.

Extreme programming

- increments of one to three weeks
  - customer can suggest improvement at any point

- argued that distinction between design and building of software are artificial
- code to be developed to meet current needs only
- frequent re-factoring to keep code structured

- *Associated with Kent Beck –*
- *Developed originally on Chrysler C3 payroll (Smalltalk) project*
- *Agile methods include Jim Highsmith's Adaptive Software Development and Alistair Cocburn's Chrystal Lite methods*

Taking Good Practices to Extreme

- If code review is good:
  - Always review --- pair programming
- If testing is good:
  - Continually write and execute test cases --- test-driven development
- If incremental development is good:
  - Come up with new increments every few days
- If simplicity is good:
  - Create the simplest design that will support only the currently required functionality.
- If design is good,
  - everybody will design daily (refactoring)
- If architecture is important,
  - everybody will work at defining and refining the architecture (metaphor)
- If integration testing is important,
  - build and integrate test several times a day (continuous integration)
- developers work in pairs
- test cases and expected results devised *before* software design

- after testing of increment, test cases added to a consolidated set of test cases

Limitations of extreme programming

- Reliance on availability of high quality developers
- Dependence on personal knowledge – after development knowledge of software may decay making future development less easy
- Rationale for decisions may be lost e.g. which test case checks a particular requirement
- Reuse of existing code less likely

Grady Booch's concern

Booch, an OO authority, is concerned that with requirements driven projects:

'Conceptual integrity sometimes suffers because this is little motivation to deal with scalability, extensibility, portability, or reusability beyond what any vague requirement might imply'

Tendency towards a large number of discrete functions with little common infrastructure?

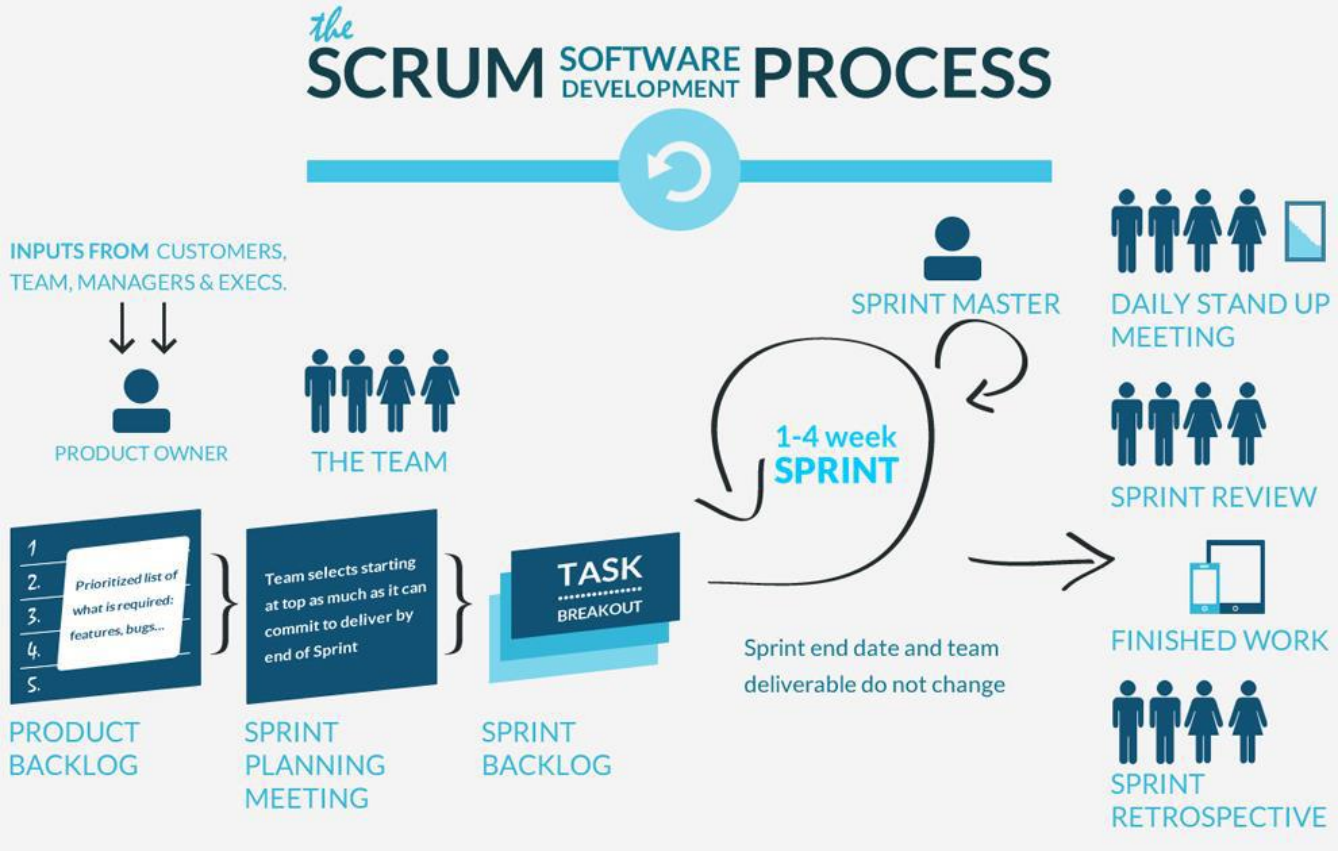
Project Characteristics that Suggest Suitability of Extreme Programming

- Projects involving new technology or research projects.
  - In this case, the requirements change rapidly and unforeseen technical problems need to be resolved.
- Small projects:
  - These are easily developed using extreme programming.

Scrum

- One of the "agile processes"
- Self-organizing teams
- Product progresses in a series of month-long "sprints"

- Requirements are captured as items in a list of "product backlog"



Sprints

- Scrum projects make progress in a series of "sprints"
- Analogous to XP iterations
- Target duration is one month
  - +/- a week or two
- During a sprint, a product increment is designed, coded, and tested

Key Roles in Scrum Process

- Product Owner
  - Acts on behalf of customers to represent their interests.
- Development Team

- Team of five-nine people with cross-functional skill sets.

- Scrum Master
  - Facilitates scrum process and resolves impediments at the team and organization level by acting as a buffer between the team and outside interference.

Scrum Ceremonies

- Sprint Planning Meeting
- Sprint
- Daily Scrum
- Sprint Review Meeting

Sprint Planning

- In this meeting, the product owner and the team members decide which Backlog Items the Team will work on in the next sprint
- Scrum Master should ensure that the Team agrees to realistic goals

Sprint

- Fundamental process flow of Scrum
- A month-long iteration, during which an incremental product functionality completed
- NO outside influence can interfere with the Scrum team during the Sprint
- Each Sprint begins with the Daily Scrum Meeting

Daily Scrum

Held daily:

- Short meeting
- Lasts for about 15mins only

Main objective is to answer three questions:

- What did you do yesterday
- What will you do today?

- What obstacles are in your way?

Sprint Review Meeting

- Team presents what it accomplished during the sprint
  - Typically takes the form of a demo of new features or underlying architecture
- Informal meeting:
  - The preparation time should not exceed about 2-hours

Sprint Artefacts

- Product backlog
- Sprint backlog
- Sprint burndown chart

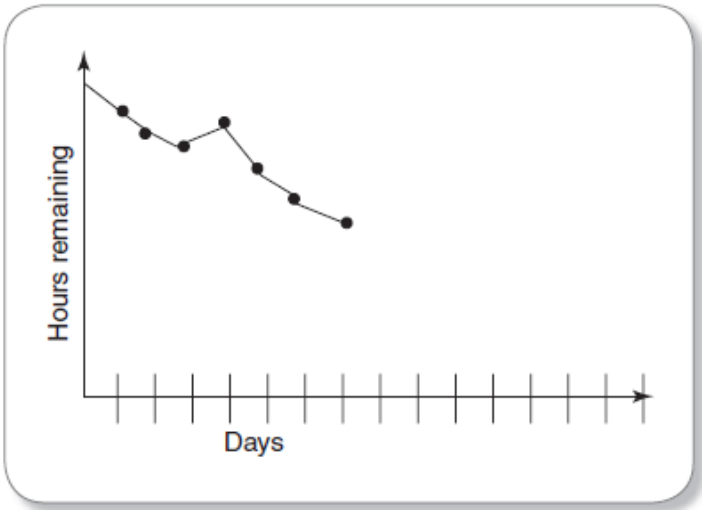
Product Backlog

- A list of all desired work on the project --- usually a combination of :
  - story-based work (e.g. “let user search and replace”)
  - task-based work (“improve exception handling”)
- List is prioritized by the Product Owner

Sprint Backlog

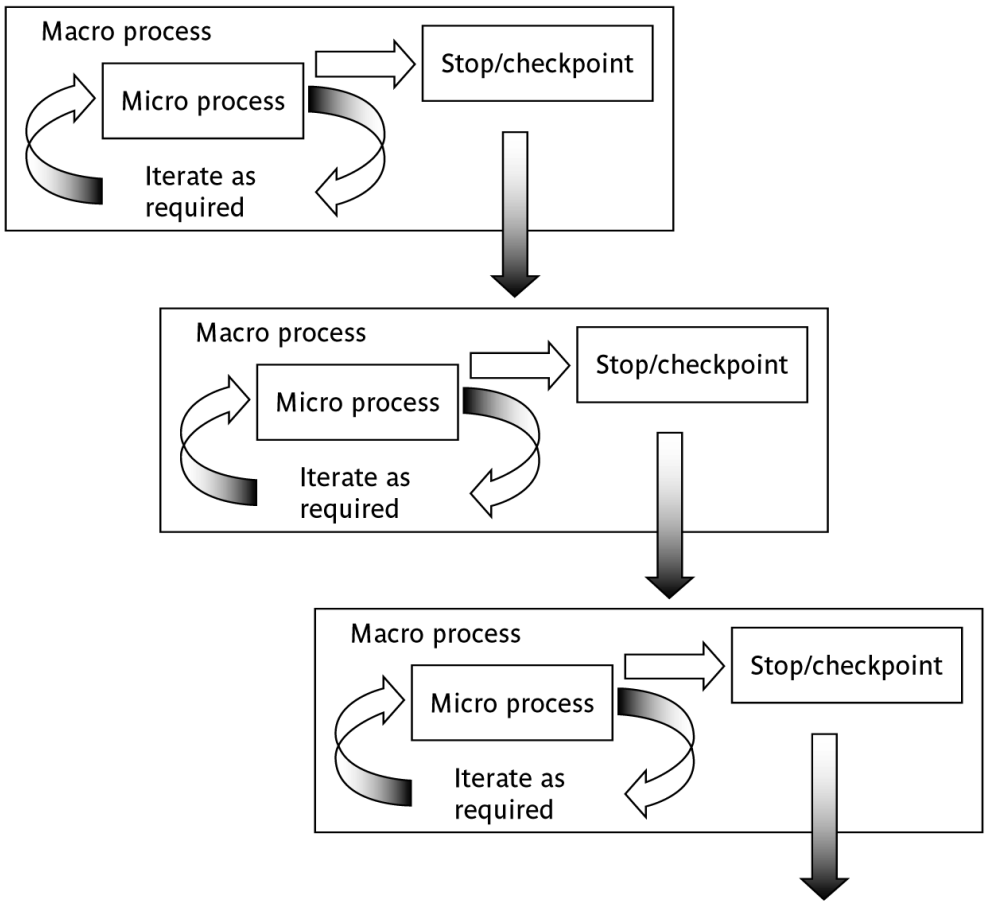
- A subset of Product Backlog Items, which define the work for a Sprint
  - Created by Team members
  - Each Item has it’s own status
  - Updated daily

Sprint Burndown Chart



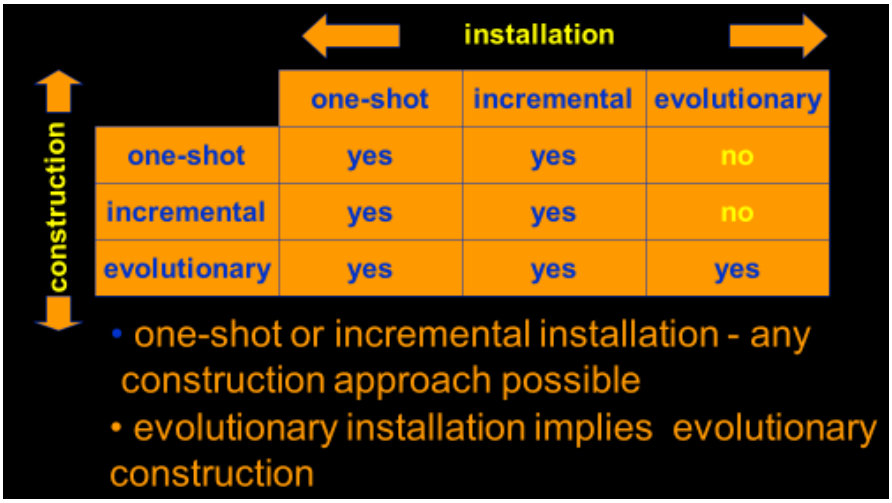
- Day-wise depicts the total Sprint Backlog hours remaining
- Ideally should burn down to zero to the end of the Sprint

macro and micro processes



Section 4.14 discusses these ideas in a little more detail

combinations of approach



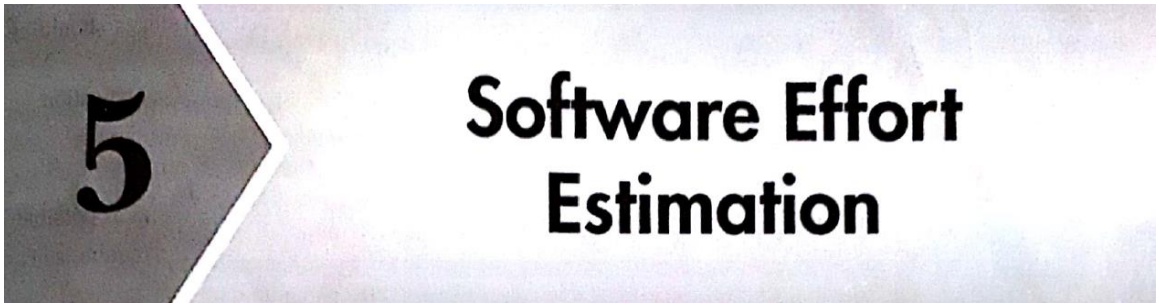
'rules of thumb' about approach to be used

IF uncertainty is high
THEN use evolutionary approach

IF complexity is high but uncertainty is not
THEN use incremental approach

IF uncertainty and complexity both low
THEN use one-shot

IF schedule is tight
THEN use evolutionary or incremental



Learning Objectives:

- why estimating is problematic (or 'challenging')
- the main generic approaches to estimating, including:
  - Bottom-up versus top-down estimating
  - Parametric models
  - Estimating by analogy
- With regard to parametric models, some particularly well-known methods, namely function points and COCOMO are then discussed in a little more detail. However, the aim is to provide an overview of the principles, not detailed counting rules.

What makes a successful project?

Delivering:	**Stages:**
• agreed functionality • on time at the agreed cost • with the required quality	1. Set targets 2. Attempt to achieve targets

BUT what if the targets are not achievable?

A key point here is that developers may in fact be very competent, but incorrect estimates leading to unachievable targets will lead to extreme customer dissatisfaction.

Some problems with estimating

- Subjective nature of much of estimating
  - It may be difficult to produce evidence to support your precise target
- Political pressures
  - Managers may wish to reduce estimated costs in order to win support for acceptance of a project proposal
- Changing technologies
  - these bring uncertainties, especially in the early days when there is a 'learning curve'
- Projects differ
  - Experience on one project may not be applicable to another

- *Section 5.1. of the textbook.*
- *Exercise 5.1 where the reader is asked to calculate productivity rates for activities on a real project illustrates some of the difficulties of interpreting project data.*

Over and under-estimating

Parkinson's Law: 'Work expands to fill the time available' An over-estimate is likely to cause project to take longer than it would otherwise	Weinberg's Zeroth Law of reliability: 'a software project that does not have to meet a reliability requirement can meet any other requirement'

The answer to the problem of over-optimistic estimates might seem to be to pad out all estimates, but this itself can lead to problems. You might miss out to the competition who could underbid you, if you were tendering for work. Generous estimates also tend to lead to reductions in productivity. On the other hand, having aggressive targets in order to increase productivity could lead to poorer product quality.

Note that 'zeroth' is what comes before first.

This is discussed in Section 5.3 of the text which also covers Brooks' Law.

Basis for successful estimating

- Information about past projects
  - Need to collect performance details about past project: how big were they? How much effort/time did they need?
- Need to be able to measure the amount of work involved
  - Traditional size measurement for software is 'lines of code' – but this can have problems

Despite our reservation about past project data –we still need to collect and analyse it! If we don't know how big the job is, we can't estimate how long it will take to do.

A taxonomy of estimating methods

- Bottom-up - activity based, analytical
- Parametric or algorithmic models e.g. function points
- Expert opinion - just guessing?
- Analogy - case-based, comparative
- Parkinson and 'price to win'

- *This taxonomy is based loosely on Barry Boehm's in the big blue book, 'Software Engineering Economics'.*
- *One problem is that different people call these approaches by different names. In the case of bottom-up and analogy some of the alternative nomenclatures have been listed.*
- *'Parkinson' is setting a target based on the amount of staff effort you happen to have available at the time. 'Price to win' is setting a target that is likely to win business when tendering for work. Boehm is scathing about these as methods of estimating. However, sometimes you might have to start with an estimate of an acceptable cost and then set the scope of the project and the application to be built to be within that cost.*
- *This is discussed in Section 5.5 of the textbook.*

Parameters to be Estimated

- Size is a fundamental measure of work
- Based on the estimated size, two parameters are estimated:
  - Effort

- Duration

- Effort is measured in person-months:
  - One person-month is the effort an individual can typically put in a month.

Person-Month

- Suppose a project is estimated to take 300 person-months to develop:
  - Is one person working for 30 days same as 30 persons working for 1 day?
  - Yes/No? why?
- How many hours is a man month?
  - Default Value: 152 hours per month
  - 19 days at 8 hours per day.

Mythical Man-Month

- "Cost varies as product of men and months, progress does not."
  - Hence the man-month as a unit for measuring the size of job is a dangerous and deceptive myth.
- The myth of additional manpower
  - Brooks Law: "Adding manpower to a late project makes it later"

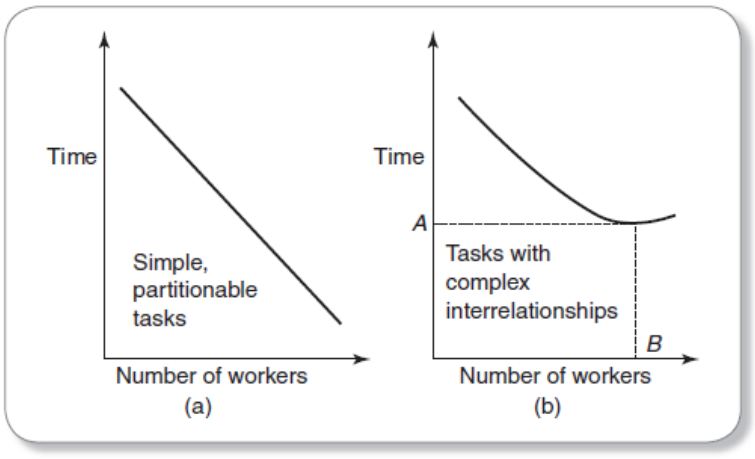


FIGURE 5.2 Impact of addition of workers on the completion time for various types of projects

For tasks with complex interrelationship, addition of manpower to a late project does not help.

Measure of Work

- The project size is a measure of the problem complexity in terms of the effort and time required to develop the product.
- Two metrics are used to measure project size:
  - Source Lines of Code (SLOC)
  - Function point (FP)
- FP is now-a-days favoured over SLOC:
  - Because of the many shortcomings of SLOC.

Major Shortcomings of SLOC

- Difficult to estimate at start of a project
- Only a code measure
- Programmer-dependent
- Does not consider code complexity

Bottom-up versus top-down

- Bottom-up
  - use when no past project data
  - identify all tasks that have to be done – so quite time-consuming
  - use when you have no data about similar past projects
- Top-down
  - produce overall estimate based on project cost drivers
  - based on past project data
  - divide overall estimate between jobs to be done

There is often confusion between the two approaches as the first part of the bottom-up approach is a top-down analysis of the tasks to be done, followed by the bottom-up adding up of effort for all the work to be done.

Bottom-up estimating

1. Break project into smaller and smaller components

2. Stop when you get to what one person can do in one/two weeks

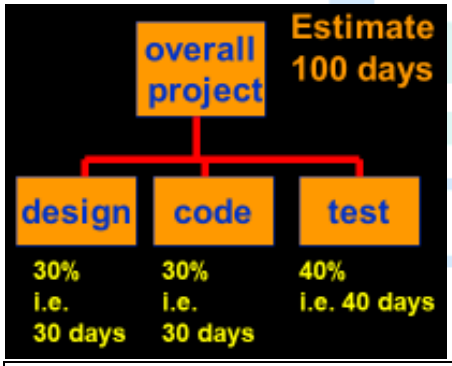
3. Estimate costs for the lowest level activities

4. At each higher level calculate estimate by adding estimates for lower levels

The idea is that even if you have never done something before you can imagine what you could do in about a week.

Exercise 5.3 relates to bottom-up estimating

Top-down estimates

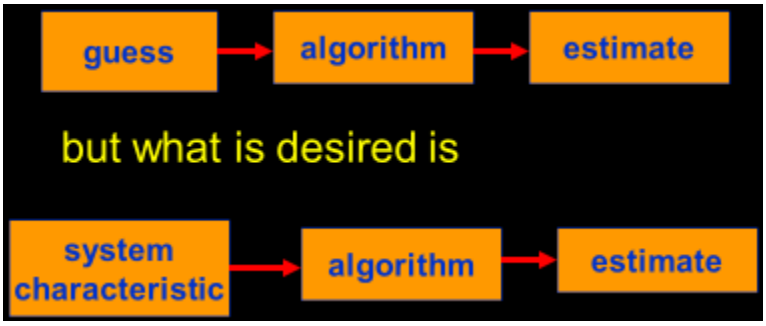


- Produce overall estimate using effort driver(s)
- distribute proportions of overall estimate to components

The initial overall estimate might have been produced using a parametric model, by analogy or just expert judgement.

Algorithmic/Parametric models

- COCOMO (lines of code) and function points examples of these
- Problem with COCOMO etc:



The problems with COCOMO is that the input parameter for system size is an estimate of lines of code. This is going to have to be an estimate at the beginning of the project.

Function points, as will be seen, counts various features of the logical design of an information system and produced an index number which reflects the amount of information processing it will have to carry out. This can be crudely equated to the amount of code it will need.

Parametric models - the need for historical data

- simplistic model for an estimate

  estimated effort = (system size) / productivity

- e.g.

  system size = lines of code

  productivity = lines of code per day

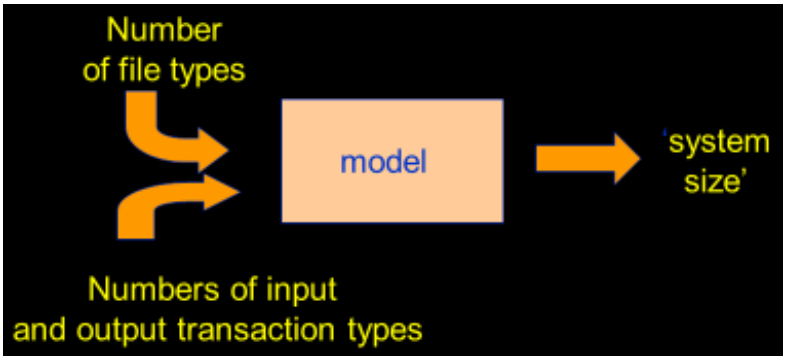
- productivity = (system size) / effort
  - based on past projects

This is analogous to calculating speed from distance and time.

Parametric models

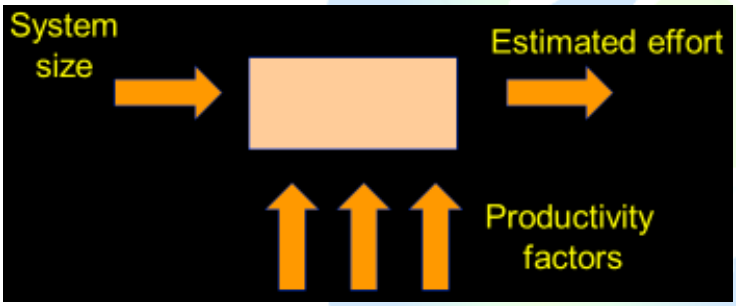
Some models focus on task or system size e.g. Function Points

FPs originally used to estimate Lines of Code, rather than effort



'System size' here can be seen as an index that allows the size of different applications to be compared. It will usually correlate to the number of lines of code required.

- Other models focus on productivity: e.g. COCOMO
- Lines of code (or FPs etc) an input

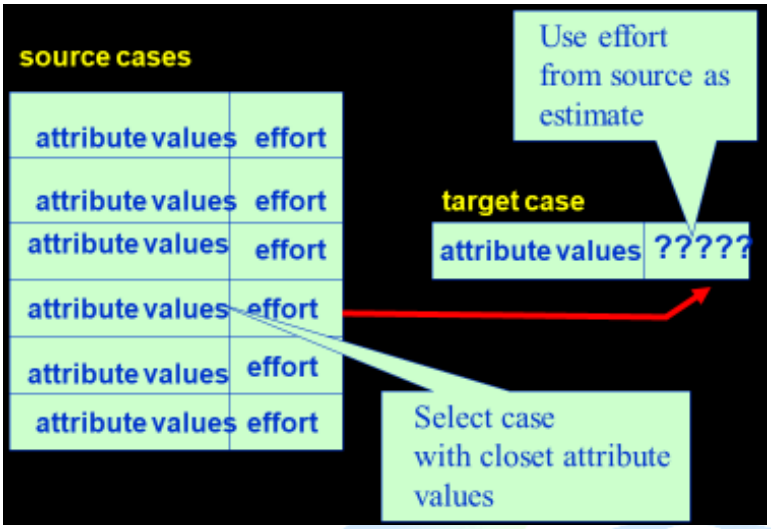


COCOMO originally was based on a size parameter of lines of code (actually 'thousand of delivered sourcecode instructions' or kdsi). Newer versions recognize the use of functions points as a size measure, but convert them to a number called 'equivalent lines of code (eloc).

Expert judgement

- Asking someone who is familiar with and knowledgeable about the application area and the technologies to provide an estimate
- Particularly appropriate where existing code is to be modified
- Research shows that experts judgement in practice tends to be based on analogy

Estimating by analogy

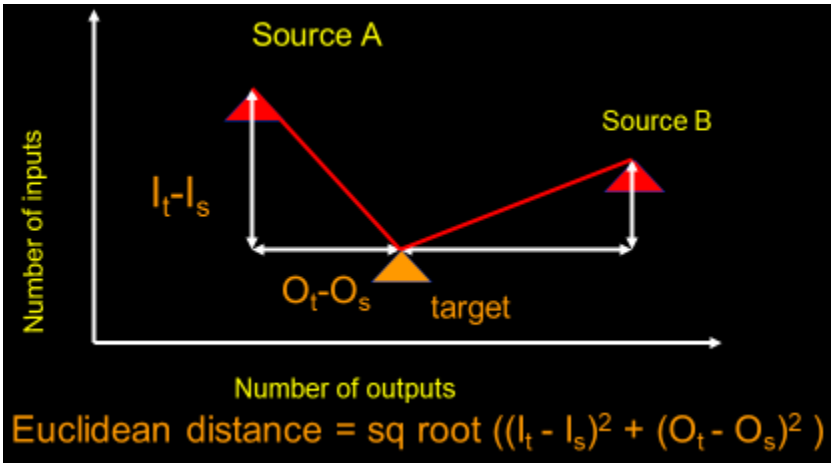


The source cases, in this situation, are completed projects. For each of details of the factors that would have a bearing on effort are recorded. These might include lines of code, function points (or elements of the FP counts such as the number of inputs, outputs etc), number of team members etc etc. For the values for the new project are used to find one or more instances from the past projects than match the current one. The actual effort from the past project becomes the basis of the estimate for the new project.

Stages: identify

- Significant features of the current project
- previous project(s) with similar features
- differences between the current and previous projects
- possible reasons for error (risk)
- measures to reduce uncertainty

Machine assistance for source selection (ANGEL)



Parametric models

We are now looking more closely at four parametric models:

1. Albrecht/IFPUG function points
2. Symons/Mark II function points
3. COSMIC function points
4. COCOMO81 and COCOMO II

Recall that function points model system size, while COCOMO focuses on productivity factors.

Albrecht/IFPUG function points

- Albrecht worked at IBM and needed a way of measuring the relative productivity of different programming languages.
- Needed some way of measuring the size of an application without counting lines of code.
- Identified five types of component or functionality in an information system
- Counted occurrences of each type of functionality in order to get an indication of the size of an information system

Different programming languages have different degrees of 'power' which relate to the ratio between the length of programming commands and the amount of processing they create. This is analogous to trying to assess the productivity of bricklayers where different bricklayers work with bricks of different sizes. One way of dealing with this problem is to say that what is important is the size of the wall being built not the number of bricks it contains. FPs are way of measuring the amount of functionality in an application without counting the lines of code in the application.

Five function types

1. **Logical interface file (LIF) types** – equates roughly to a data store in systems analysis terms. Created and accessed by the target system
2. **External interface file types (EIF)** – where data is retrieved from a data store which is actually maintained by a different application.
3. **External input (EI) types** – input transactions which update internal computer files
4. **External output (EO) types** – transactions which extract and display data from internal computer files. Generally involves creating reports.
5. **External inquiry (EQ) types** – user initiated transactions which provide information but do not update computer files. Normally the user inputs some data that guides the system to the information the user needs.

Albrecht complexity multipliers

External user types	Low complexity	Medium complexity	High complexity
EI	**3**	**4**	**6**
EO	**4**	**5**	**7**
EQ	**3**	**4**	**6**
LIF	**7**	**10**	**15**
EIF	**5**	**7**	**10**

The complexity of each instance of each 'user type' is assessed and a rating applied. Originally this assessment was largely intuitive, but later versions, developed by IFPUG (the International FP User Group) have rules governing how complexity is rated.

Examples

Payroll application has:

1. Transaction to input, amend and delete employee details – an EI that is rated of medium complexity

2. A transaction that calculates pay details from timesheet data that is input – an EI of high complexity
3. A transaction of medium complexity that prints out pay-to-date details for each employee – EO
4. A file of payroll details for each employee – assessed as of medium complexity LIF
5. A personnel file maintained by another system is accessed for name and address details – a simple EIF

What would be the FP counts for these?

FP counts

1.	Medium EI	4 FPs
2.	High complexity EI	6 FPs
3.	Medium complexity EO	5 FPs
4.	Medium complexity LIF	10 FPs
5.	Simple EIF	5 FPs
Total		**30 FPs**

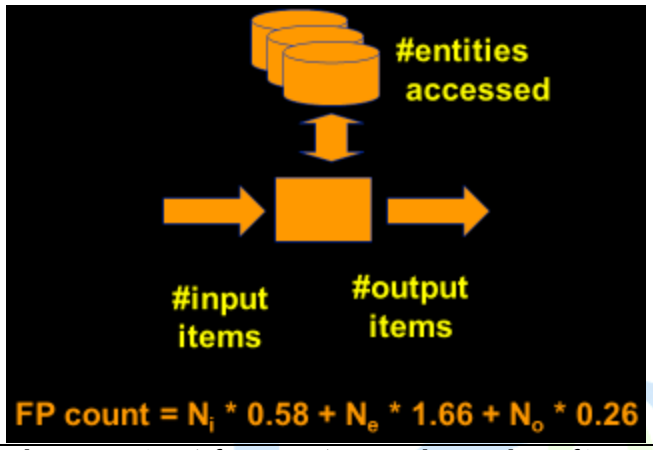
If previous projects delivered 5 FPs a day, implementing the above should take 30/5 = 6 days

Function points Mark II

- Developed by Charles R. Symons
- 'Software sizing and estimating - Mk II FPA', Wiley & Sons, 1991.
- Builds on work by Albrecht
- Work originally for CCTA:
  - should be compatible with SSADM; mainly used in UK
- has developed in parallel to IFPUG FPs
- A simpler method

Mark II FPs is a version of function points developed in the UK and is only used by a minority of FP specialists. The US-based IFPUG method (developed from the original Albrecht approach) is more widely used. I use the Mark II version because it has simpler rules and thus provides an easier

introduction to the principles of FPs. Mark II FPs are explained in more detail in Section 5.11. If you are really keen on teaching the IFPUG approach then look at Section 5.10. The IFPUG rules are really quite tricky in places and for the full rules it is best to consult IFPUG documentation.



- For each transaction, count
  - data items input (N_i)
  - data items output (N_o)
  - entity types accessed (N_e)

For each transaction (cf use case) count the number of input types (not occurrences e.g. where a table of payments is input on a screen so the account number is repeated a number of times), the number of output types, and the number of entities accessed. Multiply by the weightings shown and sum. This produces an FP count for the transaction which will not be very useful. Sum the counts for all the transactions in an application and the resulting index value is a reasonable indicator of the amount of processing carried out. The number can be used as a measure of size rather than lines of code. See calculations of productivity etc discussed earlier.

There is an example calculation in Section 5.9 (Example 5.3) and Exercise 5.9 should give a little practice in applying the method.

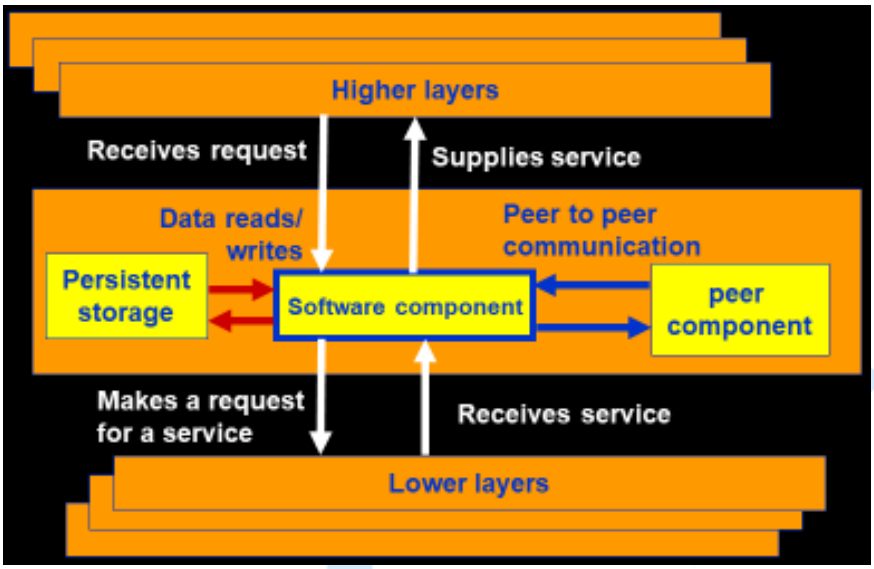
Function points for embedded systems

- Mark II function points, IFPUG function points were designed for information systems environments
- COSMIC FPs attempt to extend concept to embedded systems
- Embedded software seen as being in a particular 'layer' in the system
- Communicates with other layers and also other components at same level

- *Attempts have been made to extend IFPUG FPs to real-time and embedded systems, but this has not been very convincing (IMHO).*

- *Embedded software component is seen as at a particular level in the system. It receives calls for services from a higher layer and requests services from lower layers. It will receive responses from lower levels and will send responses to higher levels.*

Layered software



Each arrow represents an enter or exit if in black, or a read/write if in red.

COSMIC FPs

The following are counted:

- Entries: movement of data into software component from a higher layer or a peer component
- Exits: movements of data out
- Reads: data movement from persistent storage
- Writes: data movement to persistent storage

Each counts as 1 'COSMIC functional size unit' (Cfsu)

Exercise 5.10 gives some practice in applying the technique.

COCOMO81

- Based on industry productivity standards - database is constantly updated
- Allows an organization to benchmark its software development productivity
- **Basic model**

  effort = c x size^k

- C and k depend on the type of system: organic, semi-detached, embedded
- Size is measured in 'kloc' ie. Thousands of lines of code

COCOMO81 is the original version of the model which has subsequently been developed into COCOMO II some details of which are discussed in Section 5.13. For full details read Barry Boehm et al. Software estimation with COCOMO II Prentice-Hall 2002.

The COCOMO constants

System type	c	k
Organic (broadly, information systems)	2.4	1.05
Semi-detached	3.0	1.12
Embedded (broadly, real-time)	3.6	1.20

k exponentiation – 'to the power of…' adds disproportionately more effort to the larger projects takes account of bigger management overheads

An interesting question is what a 'semi-detached' system is exactly. To my mind, a project that combines elements of both real-time and information systems (i.e. has a substantial database) ought to be even more difficult than an embedded system.

Another point is that COCOMO was based on data from very large projects. There are data from smaller projects that suggest larger projects tend to be more productive because of economies of scale. At some point the diseconomies of scale caused by the additional management and communication overheads then start to make themselves felt.

Development effort multipliers (dem)

According to COCOMO, the major productivity drivers include:

Product attributes: required reliability, database size, product complexity

Computer attributes: execution time constraints, storage constraints, virtual machine (VM) volatility

Personnel attributes: analyst capability, application experience, VM experience, programming language experience

Project attributes: modern programming practices, software tools, schedule constraints

Virtual machine volatility is where the operating system that will run your software is subject to change. This could particularly be the case with embedded control software in an industrial environment.

Schedule constraints refers to situations where extra resources are deployed to meet a tight deadline. If two developers can complete a task in three months, it does not follow that six developers could complete the job in one month. There would be additional effort needed to divide up the work and co-ordinate effort and so on.

Using COCOMO development effort multipliers (dem)

An example: for analyst capability:

- Assess capability as very low, low, nominal, *high* or very high
- Extract multiplier:

very low	1.46
low	1.19
nominal	1.00
high	*0.80*
very high	0.71

- Adjust nominal estimate e.g. 32.6 x 0.80 = 26.8 staff months

As Time Passed… COCOMO 81 Showed Limitations…

- COCOMO 81 was developed with the assumption:
  - Waterfall process would be used and that all software would be developed from scratch.
- Since its formulation, there have been many changes in software engineering practices:
  - Made it difficult to use COCOMO meaningfully.

Major Changes in Program Development Practices

- Software reuse
- Application generation of programs
- Object oriented approaches
- Need for rapid development
- Agile models

COCOMO II

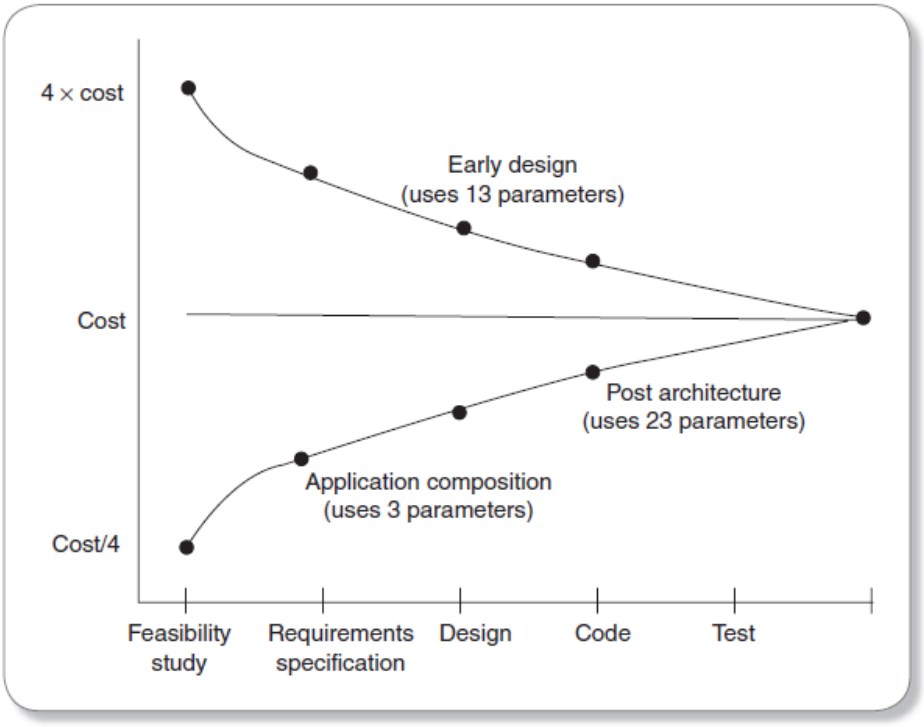


FIGURE 5.4 Accuracy of different COCOMO II estimations

COCOMO II Models

- COCOMO 2 incorporates a range of sub-models:
- Produces increasingly accurate estimates.
- The 4 sub-models in COCOMO 2 are:
  - Application composition model. Used when software is composed from existing parts.
  - Early design model. Used when requirements are available but design has not yet started.
  - Reuse model. Used to compute the effort of integrating reusable components.
  - Post-architecture model. Used once the system architecture has been designed and more information about the system is available.

An updated version of COCOMO:

- There are different COCOMO II models for estimating at the 'early design' stage and the 'post architecture' stage when the final system is implemented. We'll look specifically at the first.
- The core model is:

  $pm = A(size)^{(sf)} \times(em_1) \times(em_2) \times(em_3)\ldots.$

where **pm** = person months, **A** is 2.94, **size** is number of thousands of lines of code, **sf** is the scale factor, and $\mathbf{em_i}$ is an effort multiplier

A could possibly change as more productivity data is collected, but the value of 2.94 remains unchanged since 2000. Section 5.13.

COCOMO II Scale factor

Based on five factors which appear to be particularly sensitive to system size

1. Precedentedness (PREC). Degree to which there are past examples that can be consulted
2. Development flexibility (FLEX). Degree of flexibility that exists when implementing the project

3. Architecture/risk resolution (RESL). Degree of uncertainty about requirements
4. Team cohesion (TEAM).
5. Process maturity (PMAT) could be assessed by CMMI – see Section 13.10

Exercise 5.11 provides an exercise about the calculation of the COCOMO II scale factor.

COCOMO II Scale factor values

Driver	Very low	Low	Nom-inal	High	Very high	Extra high
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Example of scale factor

- A software development team is developing an application which is very similar to previous ones it has developed.
- A very precise software engineering document lays down very strict requirements. PREC is very high (score 1.24).
- FLEX is very low (score 5.07).
- The good news is that these tight requirements are unlikely to change (RESL is high with a score 2.83).
- The team is tightly knit (TEAM has high score of 2.19), but processes are informal (so PMAT is low and scores 6.24)

Scale factor calculation

The formula for sf is

$$sf = B + 0.01 \times \Sigma \text{ scale factor values}$$

i.e. $sf = 0.91 + 0.01$

$\times (1.24 + 5.07 + 2.83 + 2.19 + 6.24)$

$= 1.0857$

If system contained 10 kloc then estimate would be $2.94 \times 10^{1.0857} = 35.8$ person months

Using exponentiation ('to the power of') adds disproportionately more to the estimates for larger applications

Effort multipliers

As well as the scale factor effort multipliers are also assessed:

RCPX Product reliability and complexity

RUSE Reuse required

PDIF Platform difficulty

PERS Personnel capability

FCIL Facilities available

SCED Schedule pressure

Effort multipliers

	Extra low	Very low	Low	Nom-inal	High	Very high	Extra high
RCPX	**0.49**	**0.60**	**0.83**	**1.00**	**1.33**	**1.91**	**2.72**
RUSE			**0.95**	**1.00**	**1.07**	**1.15**	**1.24**

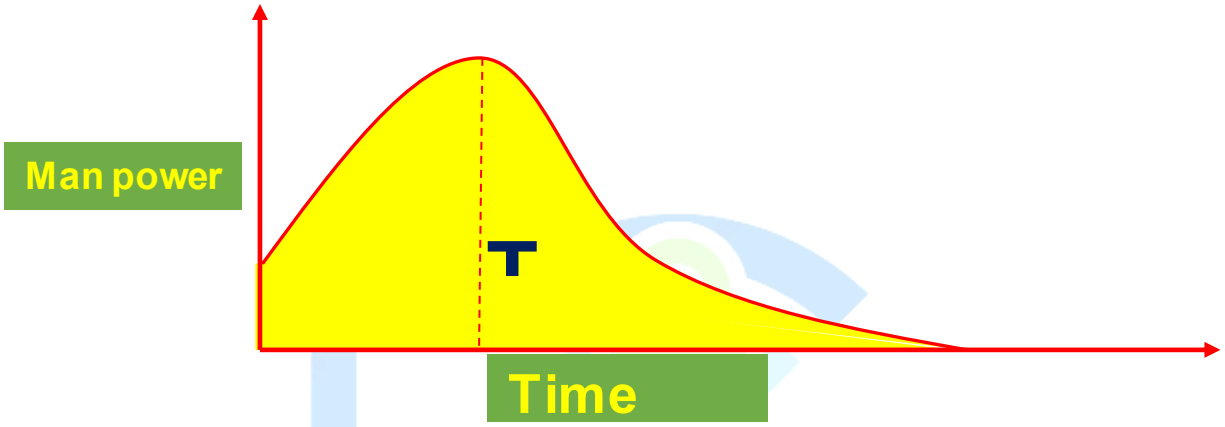
PDIF			**0.87**	**1.00**	**1.29**	**1.81**	**2.61**
PERS	**2.12**	**1.62**	**1.26**	**1.00**	**0.83**	**0.63**	**0.50**
PREX	**1.59**	**1.33**	**1.12**	**1.00**	**0.87**	**0.74**	**0.62**
FCIL	**1.43**	**1.30**	**1.10**	**1.00**	**0.87**	**0.73**	**0.62**
SCED		**1.43**	**1.14**	**1.00**	**1.00**	**1.00**	

Example

- Say that a new project is similar in most characteristics to those that an organization has been dealing for some time
- **except**
  - the software to be produced is exceptionally complex and will be used in a safety critical system.
  - The software will interface with a new operating system that is currently in beta status.
  - To deal with this the team allocated to the job are regarded as exceptionally good, but do not have a lot of experience on this type of software.
- RCPX very high 1.91
- PDIF very high 1.81
- PERS extra high 0.50
- PREX nominal 1.00
- All other factors are nominal
- Say estimate is 35.8 person months
- With effort multipliers this becomes 35.8 x 1.91 x 1.81 x 0.5 = 61.9 person months

Staffing

- Norden was one of the first to investigate staffing pattern:
  - Considered general research and development (R&D) type of projects.
- Norden concluded:
  - Staffing pattern for any R&D project can be approximated by the Rayleigh distribution curve



Putnam's Work

- Putnam adapted the Rayleigh-Norden curve:
  - Related the number of delivered lines of code to the effort and the time required to develop the product.
  - Studied the effect of schedule compression:

$$pm_{new} = pm_{org} \times \left(\frac{td_{org}}{td_{new}} \right)^4$$

Example

- If the estimated development time using COCOMO formulas is 1 year:
  - Then to develop the product in 6 months, the total effort required (and hence the project cost) increases 16 times.

Boehm's Result

- There is a limit beyond which a software project cannot reduce its schedule by buying any more personnel or equipment.
  - This limit occurs roughly at 75% of the nominal time estimate for small and medium sized projects

Capers Jones' Estimating Rules of Thumb

- Empirical rules:
  - Formulated based on observations
  - No scientific basis
- Because of their simplicity,:
  - These rules are handy to use for making off-hand estimates.
  - Give an insight into many aspects of a project for which no formal methodologies exist yet.

Capers Jones' Rules

- *Rule 1: SLOC-function point equivalence:*
  - One function point = 125 SLOC for C programs.
- Rule 2: Project duration estimation:
  - Function points raised to the power 0.4 predicts the approximate development time in calendar months.
- Rule 3: Rate of requirements creep:
  - User requirements creep in at an average rate of 2% per month from the design through coding phases.
- Rule 4: Defect removal efficiency:
  - Each software review, inspection, or test step will find and remove 30% of the bugs that are present.

- Rule 5: Project manpower estimation:
  - The size of the software (in function points) divided by 150 predicts the approximate number of personnel required for developing the application.
- Rule 6: Number of personnel for maintenance
  - *Function points divided by 500 predicts the approximate number of personnel required for regular maintenance activities.*
- Rule 7: Software development effort estimation:
  - The approximate number of staff months of effort required to develop a software is given by the software development time multiplied with the number of personnel required.

Some conclusions: how to review estimates

Ask the following questions about an estimate

- What are the task size drivers?
- What productivity rates have been used?
- Is there an example of a previous project of about the same size?
- Are there examples of where the productivity rates used have actually been found?

Unit III

6 Activity Planning

Learning Objectives

- Produce an activity plan for a project
- Estimate the overall duration of a project
- Create a critical path and a precedence network for a project

6.1 Introduction

In earlier chapters we looked at methods for forecasting the effort required for a project – both for the project as a whole and for individual activities. A detailed plan for the project, however, must also include a schedule indicating the start and completion times for each activity. This will enable us to:

- ensure that the appropriate resources will be available precisely when required;
- avoid different activities competing for the same resources at the same time;
- produce a detailed schedule showing which staff carry out each activity;
- produce a detailed plan against which actual achievement may be measured;
- produce a timed cash flow forecast;
- replan the project during its life to correct drift from the target.

To be effective, a plan must be stated as a set of targets, the achievement or non-achievement of which can be unambiguously measured. The activity plan does this by providing a target start and completion date for each activity (or a window within which each activity may be carried out). The starts and completions of activities must be clearly visible and this is one of the reasons why it is advisable to ensure that each and every project activity produces some tangible product or 'deliverable'. Monitoring the project's progress is then, at least in part, a case of ensuring that the products of each activity are delivered on time.

Project monitoring is discussed in more detail in Chapter 9.

6.2 Objectives of Activity Planning

In addition to providing project and resource schedules, activity planning aims to achieve a number of other objectives which may be summarized as follows.

- *Feasibility assessment* Is the project possible within required timescales and resource constraints? In Chapter 5 we looked at ways of estimating the effort for various project tasks. However, it is not until we have constructed a detailed plan that we can forecast a completion date with any reasonable knowledge of its achievability. The fact that a project may have been estimated as requiring two work-years' effort might not mean that it would be feasible to complete it within, say, three months were eight people to work on it – that will depend upon the availability of staff and the degree to which activities may be undertaken in parallel.

- *Resource allocation* What are the most effective ways of allocating resources to the project. When should the resources be available? The project plan allows us to investigate the relationship between timescales and resource availability (in general, allocating additional resources to a project shortens its duration) and the efficacy of additional spending on resource procurement.
- *Detailed costing* How much will the project cost and when is that expenditure likely to take place? After producing an activity plan and allocating specific resources, we can obtain more detailed estimates of costs and their timing.

Chapter 11 discusses motivation in more detail.

- *Motivation* Providing targets and being seen to monitor achievement against targets is an effective way of motivating staff, particularly where they have been involved in setting those targets in the first place.

This coordination will normally form part of Programme Management.

- *Coordination* When do the staff in different departments need to be available to work on a particular project and when do staff need to be transferred between projects? The project plan, particularly with large projects involving more than a single project team, provides an effective vehicle for communication and coordination among teams. In situations where staff may need to be transferred between project teams (or work concurrently on more than one project), a set of integrated project schedules should ensure that such staff are available when required and do not suffer periods of enforced idleness.

Activity planning and scheduling techniques place an emphasis on completing the project in a minimum time at an acceptable cost or, alternatively, meeting a set target date at minimum cost. These are not, in themselves, concerned with meeting quality targets, which generally impose constraints on the scheduling process.

One effective way of shortening project durations is to carry out activities in parallel. Clearly we cannot undertake all the activities at the same time – some require the completion of others before they can start and there are likely to be resource constraints limiting how much may be done simultaneously. Activity scheduling will, however, give us an indication of the cost of these constraints in terms of lengthening timescales and provide us with an indication of how timescales may be shortened by relaxing those constraints. If we try relaxing precedence constraints by, for example, allowing a program coding task to commence before the design has been completed, it is up to us to ensure that we are clear about the potential effects on product quality.

6.3 When to Plan

Planning is an ongoing process of refinement, each iteration becoming more detailed and more accurate than the last. Over successive iterations, the emphasis and purpose of planning will shift.

During the feasibility study and project start-up, the main purpose of planning will be to estimate timescales and the risks of not achieving target completion dates or keeping within budget. As the project proceeds beyond the feasibility study, the emphasis will be placed upon the production of activity plans for ensuring resource availability and cash flow control.

Throughout the project, until the final deliverable has reached the customer, monitoring and replanning must continue to correct any drift that might prevent meeting time or cost targets.

Scheduling

'Time is nature's way of stopping everything happening at once'

Having

- worked out a method of doing the project
- identified the tasks to be carried
- assessed the time needed to do each task

need to allocate dates/times for the start and end of each activity

Activity networks

These help us to:

- Assess the feasibility of the planned project completion date
- Identify when resources will need to be deployed to activities
- Calculate when costs will be incurred

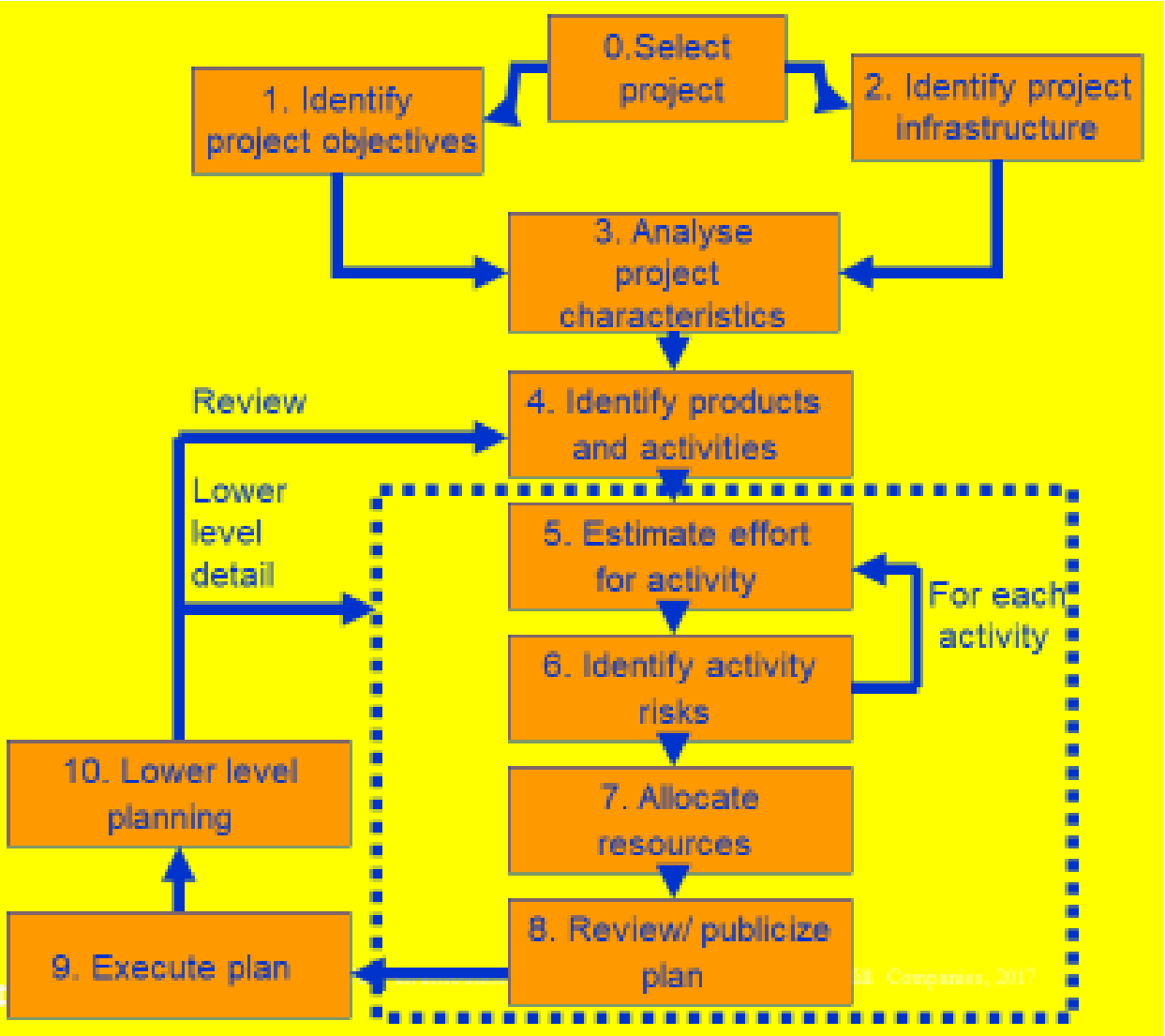
This helps the co-ordination and motivation of the project team

Defining activities

Activity networks are based on some assumptions:

- A project is:
  - Composed of a number of activities
  - May start when at least one of its activities is ready to start
  - Completed when all its activities are completed
- An activity
  - Must have clearly defined start and end-points
  - Must have resource requirements that can be forecast: these are assumed to be constant throughout the project

- Must have a duration that can be forecast
- May be dependent on other activities being completed first (precedence networks)



Identifying activities

Essentially there are three approaches to identifying the activities or tasks that make up a project – we shall call them the *activity-based approach, the product-based approach* and the *hybrid approach.*

The activity-based approach

The activity-based approach consists of creating a list of all the activities that the project is thought to involve. This might require a brainstorming session involving the whole project team or it might stem from an analysis of similar past projects. When listing activities, particularly for a large project, it might be helpful to subdivide the project into the main life-cycle stages and consider each of these separately.

Rather than doing this in an ad hoc manner, with the obvious risks of omitting or double-counting tasks, a much favoured way of generating a task list is to create a *Work Breakdown Structure* (WBS). This involves identifying the main (or high-level) tasks required to complete a project and then breaking each of these down into a set of lower-level tasks. Figure 6.2 shows a fragment of a WBS where the design task has been broken down into three tasks and one of these has been further decomposed into two tasks.

WBSs are advocated by BS 6079, the British Standards Institution's *Guide to Project Management.*

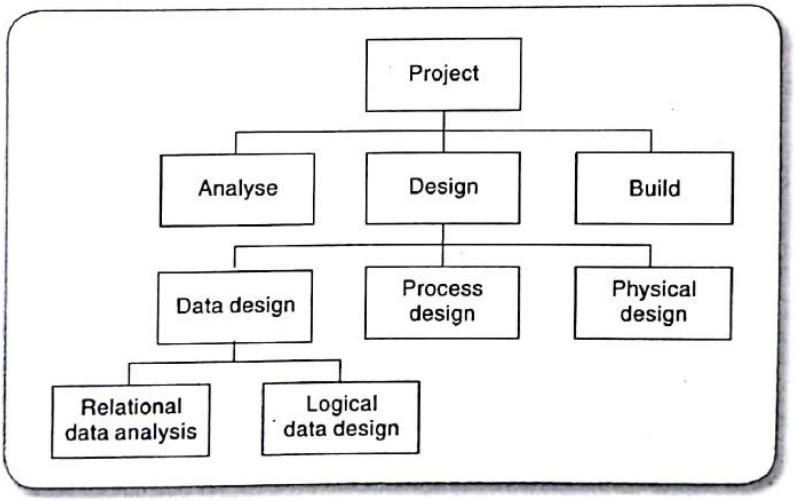


FIGURE 6.2 A fragment of an activity-based Work Breakdown Structure

Activities are added to a branch in the structure if they contribute directly to the task immediately above – if they do not contribute to the parent task, then they should not be added to that branch. The tasks at each level in any branch should include everything that is required to complete the task at the higher level.

When preparing a WBS, consideration must be given to the final level of detail or depth of the structure. Too great a depth will result in a large number of small tasks that will be difficult to manage, whereas a too shallow structure will provide insufficient detail for project control. Each branch should, however, be broken down at least to a level where each leaf may be assigned to an individual or responsible section within the organization.

A complete task catalogue will normally include task definitions along with task input and output products and other task-related information.

Advantages claimed for the WBS approach include the belief that it is much more likely to result in a task catalogue that is complete and is composed of non-overlapping activities. Note that it is only the leaves of the structure that comprise the list of activities in the project – higher-level nodes merely represent collections of activities.

The WBS also represents a structure that may be refined as the project proceeds. In the early part of a project we might use a relatively high-level or shallow WBS, which can be developed as information becomes available, typically during the project's analysis and specification phases.

Once the project's activities have been identified (whether or not by using a WBS), they need to be sequenced in the sense of deciding which activities need to be completed before others can start.

- **Work-based:** draw-up a Work Breakdown Structure listing the work items needed
- **Product-based approach**
  - list the deliverable and intermediate products of project – product breakdown structure (PBS)
  - Identify the order in which products have to be created
  - work out the activities needed to create the products

Product-based approach

The product-based approach, used in PRINCE2 and Step Wise, has already been described in Chapter 3. It consists of producing a Product Breakdown Structure and a Product Flow Diagram. The PFD indicates, for each product, which other products are required as inputs. The PFD can therefore be easily transformed into an ordered list of activities by identifying the transformations that turn some products into others. Proponents of this approach claim that it is less likely that a product will be left out of a PBS than that an activity might be omitted from an unstructured activity list.

This approach is particularly appropriate if using a methodology such as SSADM or USDP (Unified Software Development Process), which clearly specifies, for each step or task, each of the products required and the activities required to produce it. For example, the SSADM Reference Manual provides a set of generic PBSs for each stage in SSADM, which can be used as a basis for generating a project specific PBS.

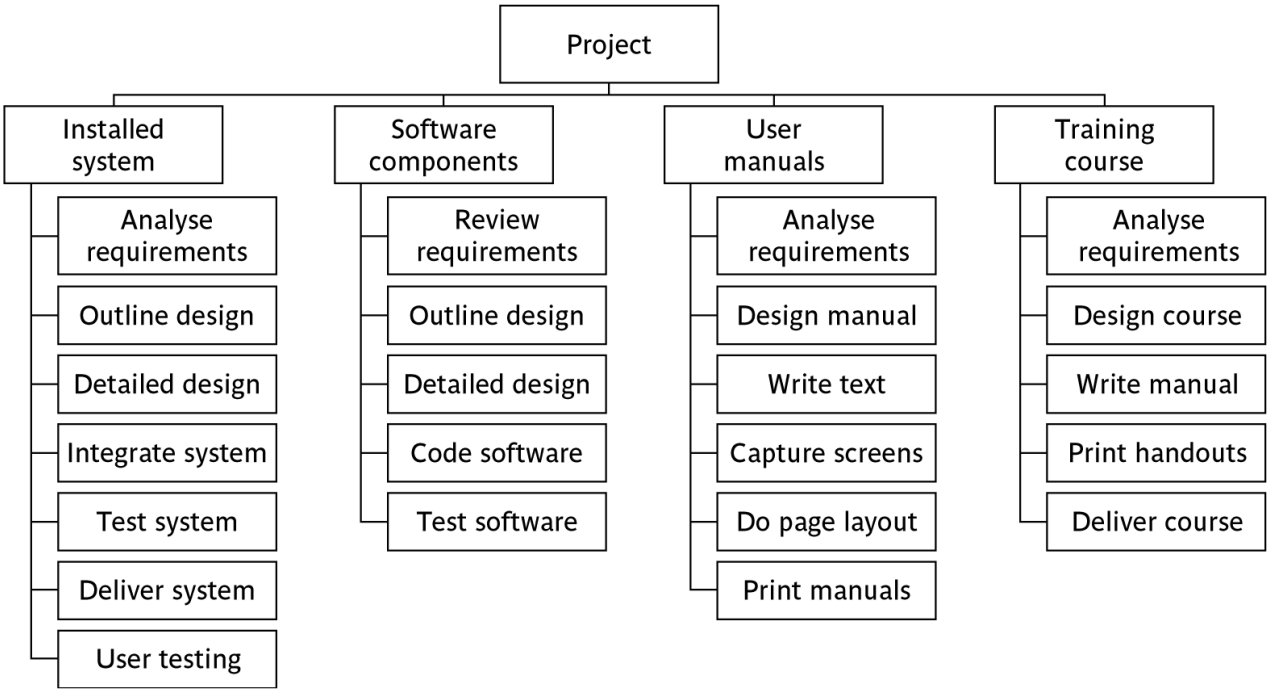
See I. Jacobson, G. Booch and J. Rumbaugh (1999) *The Unified Software Development Process*, Addison-Wesley.

In the USDP, products are referred to as *artifacts* – see Figure 6.3 – and the sequence of activities needed to create them is called a *workflow*– see Figure 6.4 for an example. Some caution is needed in drawing up an activity network from these workflows. USDP emphasizes that processes are iterative. This means that it may not be possible to map a USDP process directly onto a single activity in a network. In Section 4.18 we saw how one or more iterated processes could be hidden in the single execution of a larger activity. All projects, whether they contain iterations or not, will need to have some fixed milestones or time-boxes if progress towards a planned delivery date is to be maintained. These larger activities with the fixed completion dates would be the basis of the activity network.

Hybrid approach

The IBM MITP approach suggested the following 5 levels

- Level 1: Project
- Level 2: Deliverables
- Level 3: Components – which are key work items needed to produce the deliverables
- Level 4: Work packages: groups of tasks needed to produce the components
- Level 5: Tasks



The final outcome of the planning process

6.6 Sequencing and Scheduling Activities

Throughout a project, we will require a schedule that clearly indicates when each of the project's activities is planned to occur and what resources it will need. We shall be considering scheduling in more detail in Chapter 8, but let us consider in outline how we might present a schedule for a small project. One way of presenting such a plan is to use a bar chart as shown in Figure 6.6.

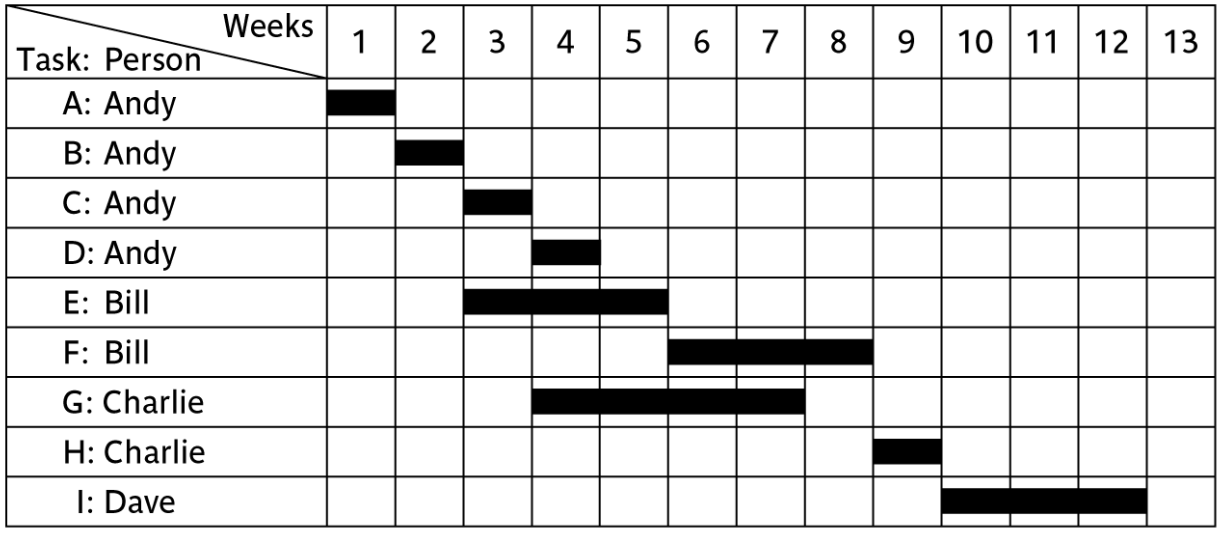
The chart shown has been drawn up taking account of the nature of the development process (that is, certain tasks must be completed before others may start) and the resources that are available (for example, activity C follows activity B because Andy cannot work on both tasks at the same time). In drawing up the chart, we have therefore done two things – we have sequenced the tasks (that is, identified the dependencies among activities dictated by the development process) and scheduled them (that is, specified when they should take place). The scheduling has had to take account of the availability of staff and the ways in which the activities have been allocated to them. The schedule might look quite different were there a different number of staff or were we to allocate the activities differently.

The bar chart does not show why certain decisions have been made. It is not clear, for example, why activity H is not scheduled to start until week 9. It could be that it cannot start until activity F has been completed or it might be because Charlie is going to be on holiday during week 8.

Separating the logical sequencing from the scheduling may be likened to the principle in systems analysis of separating the logical system from its physical implementation.

In the case of small projects, this combined sequencing–scheduling approach might be quite suitable, particularly where we wish to allocate individuals to particular tasks at an early planning stage. However, on larger projects it is better to separate out these two activities: to sequence the tasks according to their logical relationships and then to schedule them taking into account resources and other factors.

Approaches to scheduling that achieve this separation between the logical and the physical use networks to model the project and it is these approaches that we will consider in subsequent sections of this chapter.



Activity key

A: Overall design
B: Specify module 1
C: Specify module 2
D: Specify module 3
E: Code module 1
F : Code module 3
G: Code module 2
H: Integration testing
I : System tesing

Fig 6.6. A project plan as a bar chart

*The chart tells us **who** is doing **what** and **when**.*

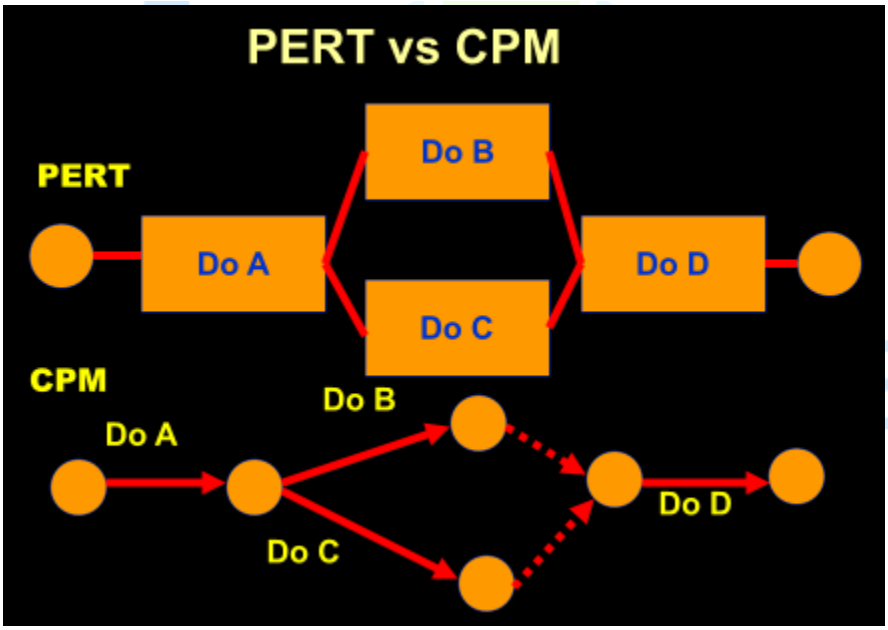
6.7 Network Planning Models

CPM was developed by the DuPont Chemical Company which published the method in 1958, claiming that it had saved them $1 million in its first year of use.

These project scheduling techniques model the project's activities and their relationships as a network. In the network, time flows from left to right. These techniques were originally developed in the 1950s – the two best known being CPM (Critical Path Method) and PERT (Program Evaluation Review Technique).

Both of these techniques used an *activity-on-arrow* approach to visualizing the project as a network where activities are drawn as arrows joining circles, or nodes, which represent the possible start and/or completion of an activity or set of activities. More recently a variation on these techniques, called *precedence networks*, has become popular. This method uses *activity-on-node* networks where activities are represented as nodes and the links between nodes represent precedence (or sequencing) requirements. This latter approach avoids some of the problems inherent in the activity-on-arrow representation and provides more scope for easily representing certain situations. It is this method that is adopted in the majority of computer applications currently available. These three methods are very similar and it must be admitted that many people use the same name (particularly CPM) indiscriminately to refer to any or all of the methods.

In the following sections of this chapter, we will look at the critical path method applied to precedence (activity-on-node) networks followed by a brief introduction to activity-on-arrow networks – a discussion of PERT will be reserved for Chapter 7 when we look at risk analysis.



PERT was devised to support the development of the Polaris missile in the late 1950's. CPM was developed by Du Pont Chemical Company who published the method in 1958.

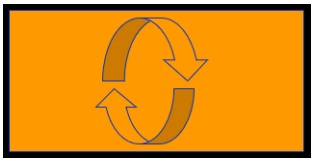
PERT is an activity-on-node notation – the 'nodes' are the boxes which represent activities

CPM uses an activity-on-arrow notation where the arrows are the activities.

The approached described here is based on PERT but the other approach is described in the textbook as well – see Section 6.16

Drawing up a PERT diagram

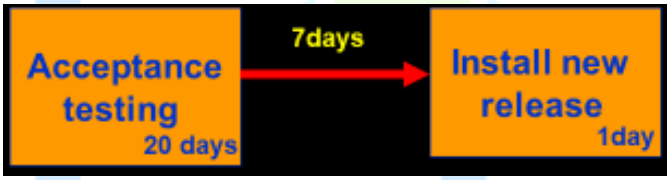
- No looping back is allowed – deal with iterations by hiding them within single activities



- *milestones* – 'activities', such as the start and end of the project, which indicate transition points. They have zero duration.

Lagged activities

- where there is a fixed delay between activities e.g. seven days notice has to be given to users that a new release has been signed off and is to be installed



Representing lagged activities

We might come across situations where we wish to undertake two activities in parallel so long as there is a lag between the two. We might wish to document amendments to a program as it is being tested – particularly if evaluating a prototype. In such a case we could designate an activity 'test and document amendments'. This would, however, make it impossible to show that amendment recording could start, say, one day after testing had begun and finish a little after the completion of testing.

Where activities can occur in parallel with a time lag between them, we represent the lag with a duration on the linking arrow as shown in Figure 6.13. This indicates that documenting amendments can start one day after the start of prototype testing and will be completed two days after prototype testing is completed.

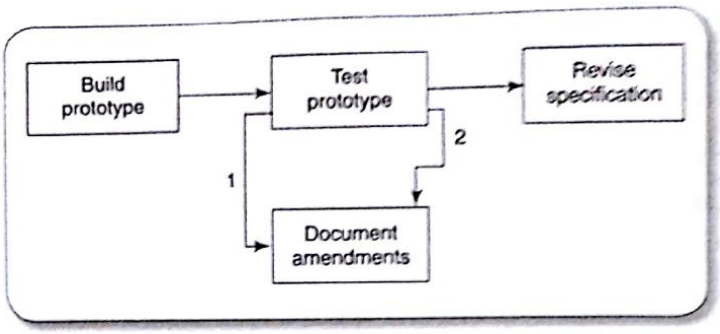
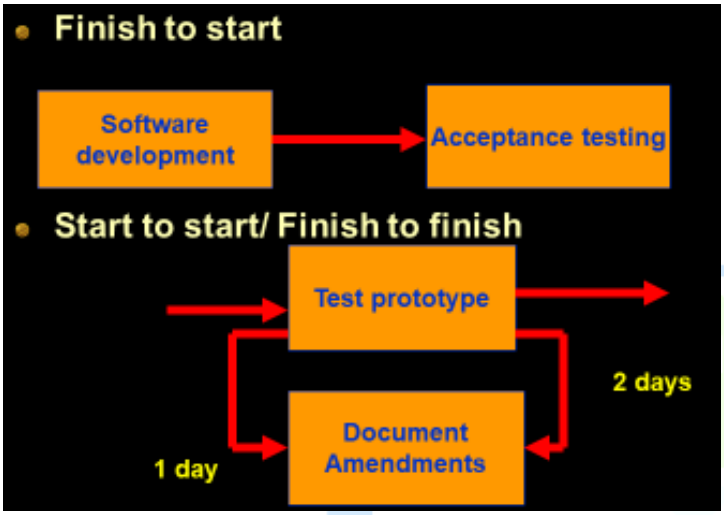


FIGURE 6.13 Indicating lags

Hammock activities

Hammock activities are activities which, in themselves, have zero duration but are assumed to start at the same time as the first 'hammocked' activity and to end at the same time as the last one. They are normally used for representing overhead costs or other resources that will be incurred or used at a constant rate over the duration of a set of activities.

Types of links between activities



Finish to start: The following activity starts when the previous one has been finished

e.g. testing starts when coding has been completed

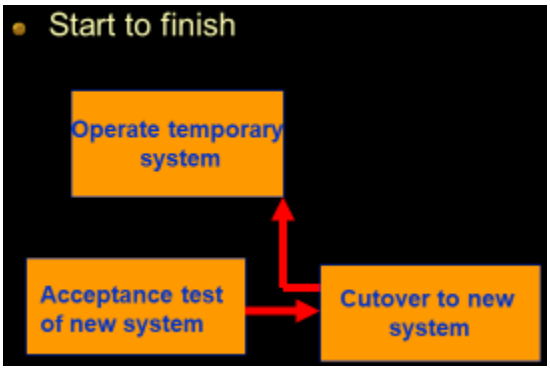
Start to start: When one activity starts another has to start as well

e.g. when prototype testing starts amendment documentation has to start as well

Finish to finish: when one activity finishes the other must finish too

e.g. when the testing of the prototype is completed so is the documentation of any amendments

You could use these with lags e.g. documentation of the changes to the prototype starts 1 day after the testing and finishes 2 days after testing has been completed



Start to finish – in the example when the cutover to the new system takes place, the operation of the temporary system is no longer needed. Although the cutover depends of the acceptance testing to be completed, the implication is that the cutover might not start straight after acceptance testing.

Start and finish times

- Activity 'write report software'
- Earliest start (ES)
- Earliest finish (EF) = ES + duration
- Latest finish (LF) = latest task can be completed without affecting project end Latest start = LF - duration

- The time that an activity can start depends on its relationship with the other tasks in the project. The earliest start is when the earliest of the preceding activities upon which the current activity depends will be completed, so that the current one can start. If it starts at this time the earliest the current activity can finish is the earliest start plus its duration.
- However, it may be that the activity, although it *can* start, can be delayed because later activities do not have to start right way. This gives us a latest finish date. The latest start date is the latest finish date less the duration of the activity.
- When a student is given coursework to do they do not necessarily start it straight away. They might note when it has got to be handed in, work out that it will only take about three days to do - with a bit of luck - and wait until three days before the hand-in before they start.

Example

- earliest start = day 5
- latest finish = day 30
- duration = 10 days
- earliest finish = ?
- latest start = ?

Float = LF - ES – duration

What is it in this case?

The earliest finish (EF) would be day 5 plus 10 days i.e. day 15.

The latest start (LS) would be day 30 – 10 days i.e. day 20

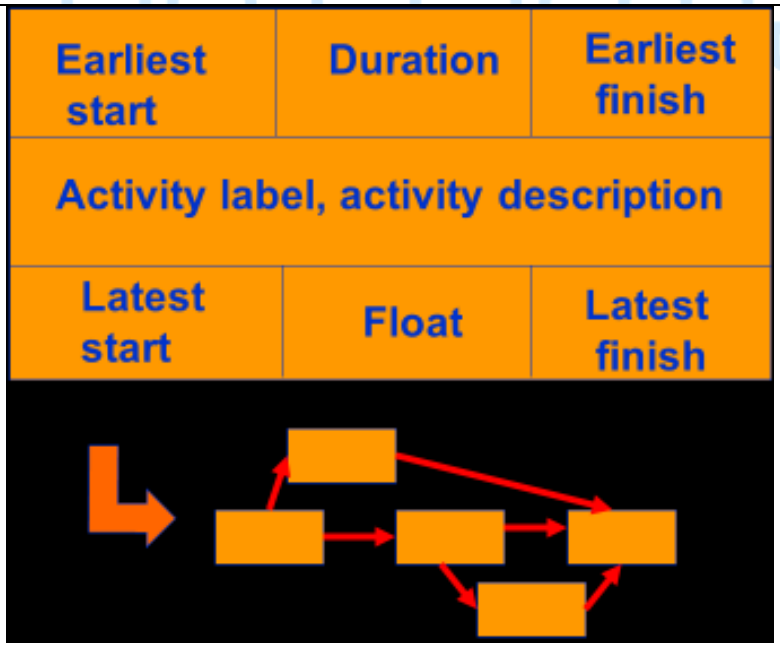
The float would be 30 – 5 – 10 = 15 days

This also is the same as LF – EF or LS - ES

'Day 0'

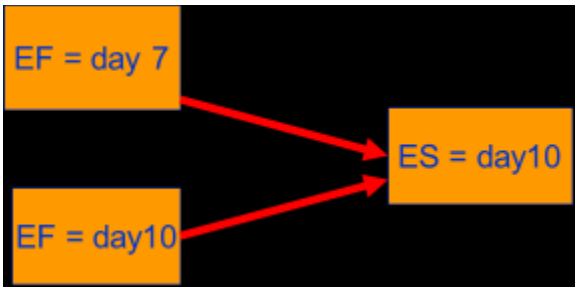
- Note that in the last example, day numbers used rather than actual dates
- Makes initial calculations easier – not concerned with week-ends and public holidays
- For **finish** date/times Day 1 means at the END of Day 1.
- For a **start** date/time Day 1 also means at the END of Day 1.
- The first activity therefore begin at Day 0 i.e. the end of Day 0 i.e. the start of Day 1

- This also means activity A could finish on Day 5, for example, and a dependent activity B could then start on Day 5 as well (not Day 6).
- All this may make more sense if you think about an activities that finish and start halfway through a particular day.
- Although we talk about 'Days' other time units could be equally valid e.g. Hours, weeks, months. In case of weeks point out that we usually assume 5 day working weeks in our calculations – unless otherwise specified.

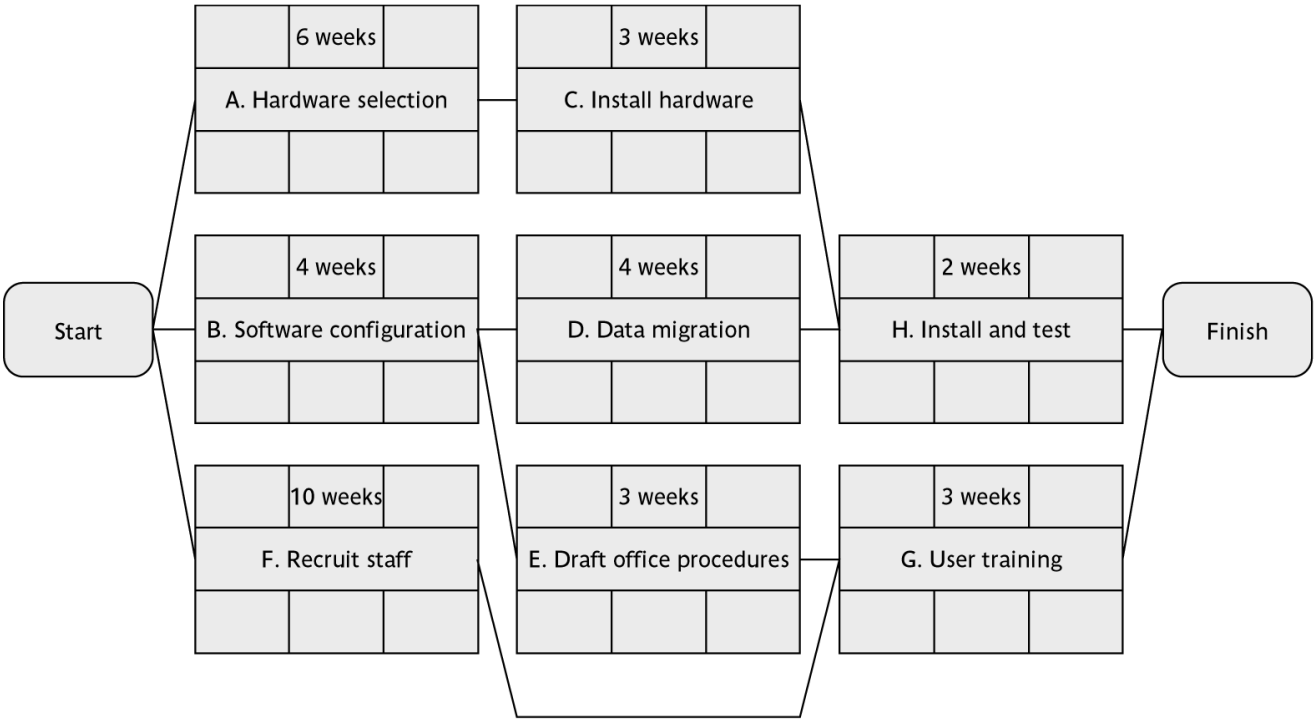


Forward pass

- Start at beginning (Day 0) and work forward following chains.
- Earliest start date for the *current* activity = earliest finish date for the *previous*
- When there is more than one previous activity, take the *latest* earliest finish



Example of an activity network

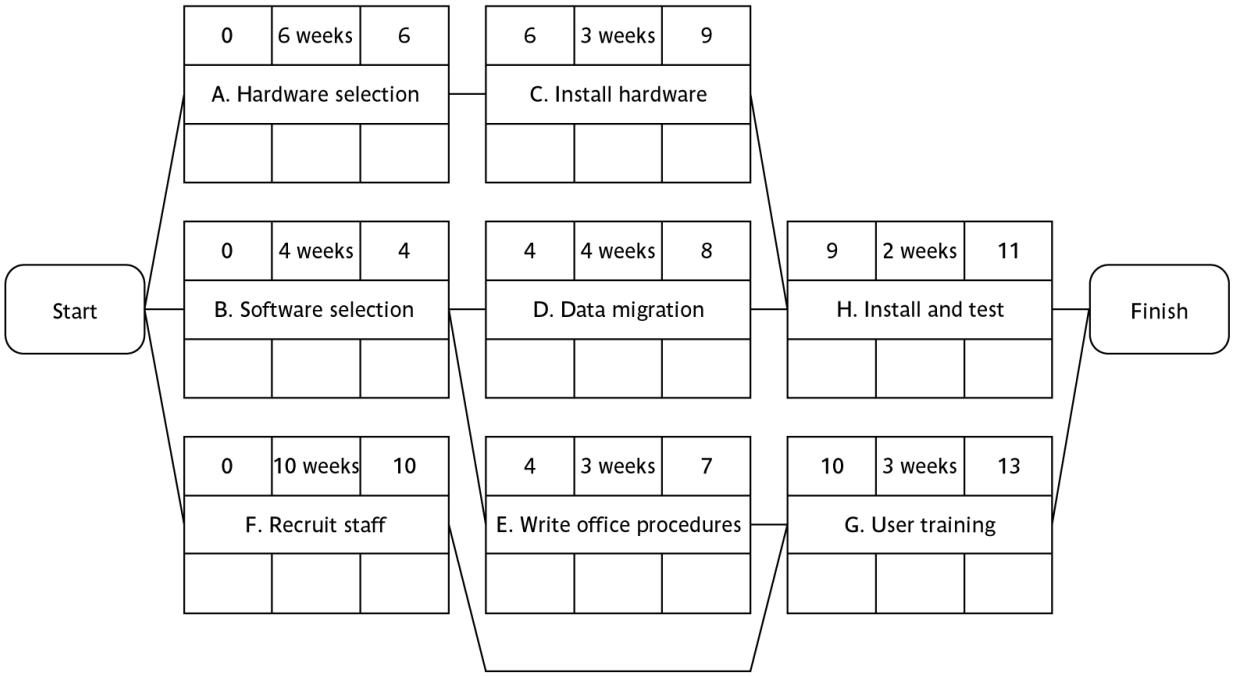


Backward pass

- Start from the *last* activity
- Latest finish (LF) for last activity = earliest finish (EF)
- work backwards

- Latest finish for *current* activity = Latest start for the *following*
- More than one following activity - take the *earliest* LS
- Latest start (LS) = LF for activity - duration

Example: LS for all activities?



Float

Float = Latest finish - Earliest start - Duration

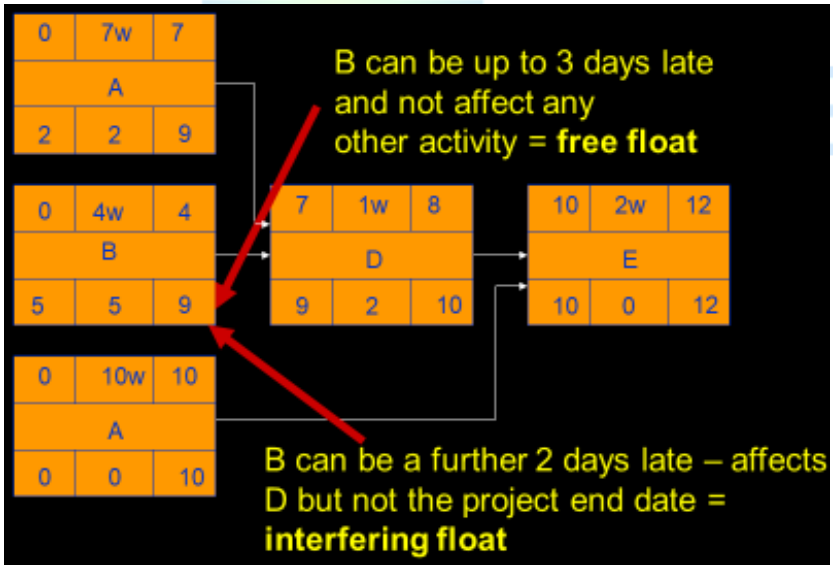
Note that Float can also be calculated as the difference between the earliest and latest start dates for an activity *or* the difference between the earliest and latest finish dates.

Critical path

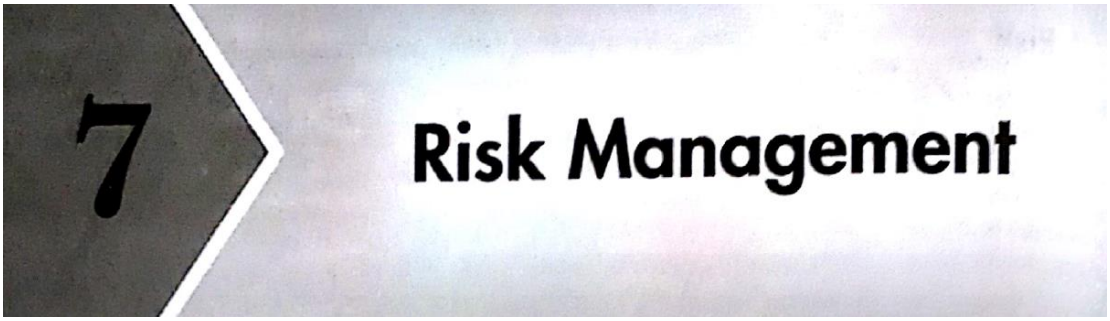
- Note the path through network with zero floats
- Critical path: any delay in an activity on this path will delay whole project
- Can there be more than one critical path?
- Can there be no critical path?
- Sub-critical paths

- Yes, there could be more than one critical path if the two longest paths through the network were of equal length.
- Where the target completion date for the project was imposed rather than calculated from the earliest finish dates, it might be later that the earliest finish date. In this case there would be no chains of activities with zero floats
- The durations of activities are only estimates to start with. As the project proceeds, the estimates will be replaced by actual durations which could be different. This could change the sequence of activities identified as the critical path. Sub-critical paths are chains of activities, not on the planned critical path, but which have small floats and which could easily become the critical path as the project develops.

Free and interfering float



Total float = LF – ES – duration (or LS-ES or LF-EF)
Free float = ES for following activity less EF for the current
Interfering float = total float – free float



Risk management

Learning Objectives:

- Definition of 'risk' and 'risk management'
- Some ways of categorizing risk
- Risk management
  - Risk identification – what are the risks to a project?
  - Risk analysis – which ones are really serious?
  - Risk planning – what shall we do?
  - Risk monitoring – has the planning worked?
- We will also look at PERT risk and critical chains

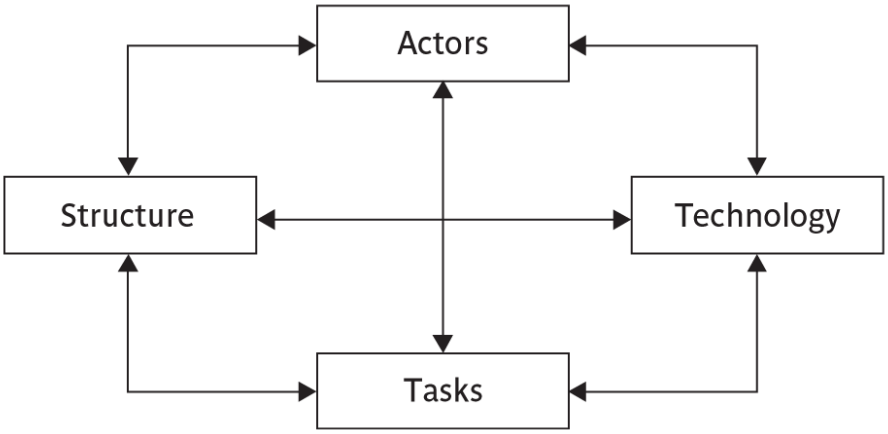
Some definitions of risk

'the chance of exposure to the adverse consequences of future events' PRINCE2

'an uncertain event or condition that, if it occurs, has a positive or negative effect on a project's objectives' PM-BOK

- Risks relate to **possible future** problems, not current ones
- They involve a possible cause and its effect(s) e.g. developer leaves > task delayed

Categories of risk



This is based on Lyytinen's sociotechnical model of risk

- **Actors** relate to all those involved in the project including both developers, users and managers e.g. a risk could be that high staff turnover leads to information of importance to the project being lost
- **Technology** – both that used to implement the project and that embedded in the project deliverables – risk could be that the technologies selected are not in fact appropriate.
- **Structure** – this includes management procedures, risk here is that a group who need to carry out a particular project task are not informed of this need because they are not part of the project communication network
- **Tasks** – the work to be carried out. A typical risk is that the amount of effort needed to carry out the task is underestimated.

A risk could be well belong to more than one of the four areas – for example, estimates being wrong could be influenced by problems with actors (e.g. lack of experience with a technical domain) or the structure (over optimism of managers keen to win work).

Risk Management Approaches

- **Proactive:**
  - The proactive approaches try to anticipate the possible risks that the project is susceptible to.
  - After identifying the possible risks, actions are taken to eliminate the risks.

- **Reactive:**
  - Reactive approaches take no action until an unfavourable event occurs.
  - Once an unfavourable event occurs, these approaches try to contain the adverse effects associated with the risk and take steps to prevent future occurrence of the same risk events.

A framework for dealing with risk

The planning for risk includes these steps:

- Risk identification – what risks might there be?
- Risk analysis and prioritization – which are the most serious risks?
- Risk planning – what are we going to do about them?
- Risk monitoring – what is the current state of the risk?

Risk identification

Approaches to identifying risks include:

- **Use of checklists** – usually based on the experience of past projects
- **Brainstorming** – getting knowledgeable stakeholders together to pool concerns
- **Causal mapping** – identifying possible chains of cause and effect

Boehm's top 10 development risks

Barry Boehm surveyed software engineering project leaders to find out the main risks that they had experienced with their projects. For each risk, some risk reduction techniques has been suggested.

7.6 Risk Identification

The two main approaches to the identification of risks are the use of *checklists* and *brainstorming*.

Checklists are simply lists of the risks that have been found to occur regularly in software development projects. A specialized list of software development risks by Barry Boehm appears in Table 7.1 in a modified version. Ideally a group of representative project stakeholders examines a checklist identifying risks applicable to their project. Often the checklist suggests potential countermeasures for each risk.

TABLE 7.1 Software project risks and strategies for risk reduction

Risk	Risk reduction techniques
Personnel shortfalls	Staffing with top talent; job matching; teambuilding; training and career development; early scheduling of key personnel
Unrealistic time and cost estimates	Multiple estimation techniques; design to cost; incremental development; recording and analysis of past projects; standardization of methods
Developing the wrong software functions	Improved software evaluation; formal specification methods; user surveys; prototyping; early user manuals
Developing the wrong user interface	Prototyping; task analysis; user involvement
Gold plating	Requirements scrubbing; prototyping; cost–benefit analysis; design to cost
Late changes to requirements	Stringent change control procedures; high change threshold; incremental development (deferring changes)
Shortfalls in externally supplied components	Benchmarking; inspections; formal specifications; contractual agreements; quality assurance procedures and certification
Shortfalls in externally performed tasks	Quality assurance procedures; competitive design or prototyping; contract incentives
Real-time performance shortfalls	Simulation; benchmarking; prototyping; tuning; technical analysis
Development technically too difficult	Technical analysis; cost–benefit analysis; prototyping; staff training and development

This top ten list of software risks is based on one presented by Barry Boehm in his *Tutorial on Software Risk Management*, IEE Computer Society, 1989.

Project management methodologies, such PRINCE2, often recommend that on completion of a project a review identifies any problems during the project and the steps that were (or should have been) taken to resolve or avoid them. These problems could in some cases be added to an organizational risk checklist for use with new projects.

The 'lessons learnt' report differs from a 'post implementation review' (PIR). It is written on project completion and focuses on project issues. A PIR, produced when the application has been operational for some time, focuses on business benefits.

'Gold plating' refers to inclusion of features that in fact are unnecessary and which end up never actually being used.

Risk prioritization

Risk exposure (RE) = (potential damage) x (probability of occurrence)

Ideally

Potential damage: a money value e.g. a flood would cause £0.5 millions of damage

Probability 0.00 (absolutely no chance) to 1.00 (absolutely certain) e.g. 0.01 (one in hundred chance)

RE = £0.5m x 0.01 = £5,000

Crudely analogous to the amount needed for an insurance premium

7.7 Risk Assessment

A common problem with risk identification is that a list of risks is potentially endless. A way is needed of distinguishing the damaging and likely risks. This can be done by estimating the *risk exposure* for each risk using the formula:

$$\textit{risk exposure} = (\textit{potential damage}) \times (\textit{probability of occurrence})$$

Using the most rigorous – but not necessarily the most practical – approach, the potential damage would be assessed as a money value. Say a project depended on a data centre vulnerable to fire. It might be estimated that if a fire occurred a new computer configuration could be established for £500,000. It might also be estimated that where the computer is located there is a 1 in 1000 chance of a fire actually happening, that is a probability of 0.001.

The risk exposure in this case would be:

$$£500{,}000 \times 0.001 = £500$$

A crude way of understanding this value is as the minimum sum an insurance company would require as a premium. If 1000 companies, all in the same position, each contributed £500 to a fund then, when the 1 in 1000 chance of the fire actually occurred, there would be enough money to cover the cost of recovery.

Risk probability: qualitative descriptors

Probability level	*Range*
High	Greater than 50% chance of happening
Significant	30-50% chance of happening
Moderate	10-29% chance of happening
Low	Less than 10% chance of happening

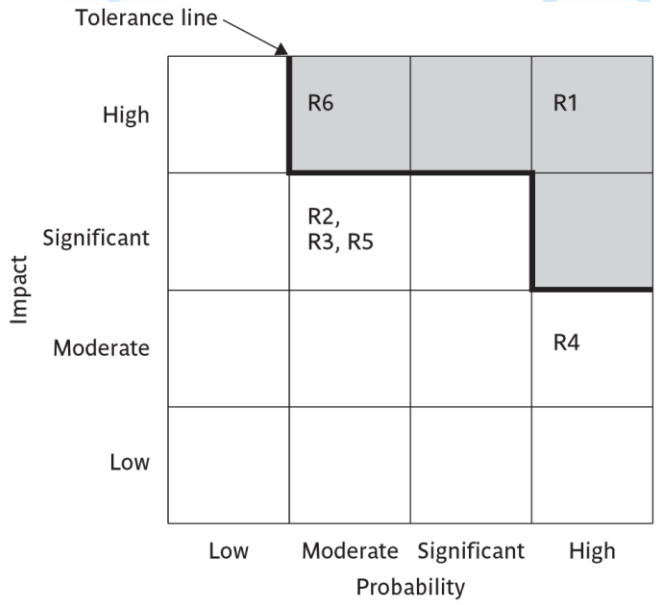
Qualitative descriptors of impact on cost and associated range values

Impact level	*Range*
High	Greater than 30% above budgeted expenditure
Significant	20 to 29% above budgeted expenditure
Moderate	10 to 19% above budgeted expenditure
Low	Within 10% of budgeted expenditure.

Similar tables can be produced for the impact on project duration and on the quality of project deliverables.

The problem with the qualitative approach is how do you combine the judgements about probability and impact – you can't multiply them together.

Probability impact matrix



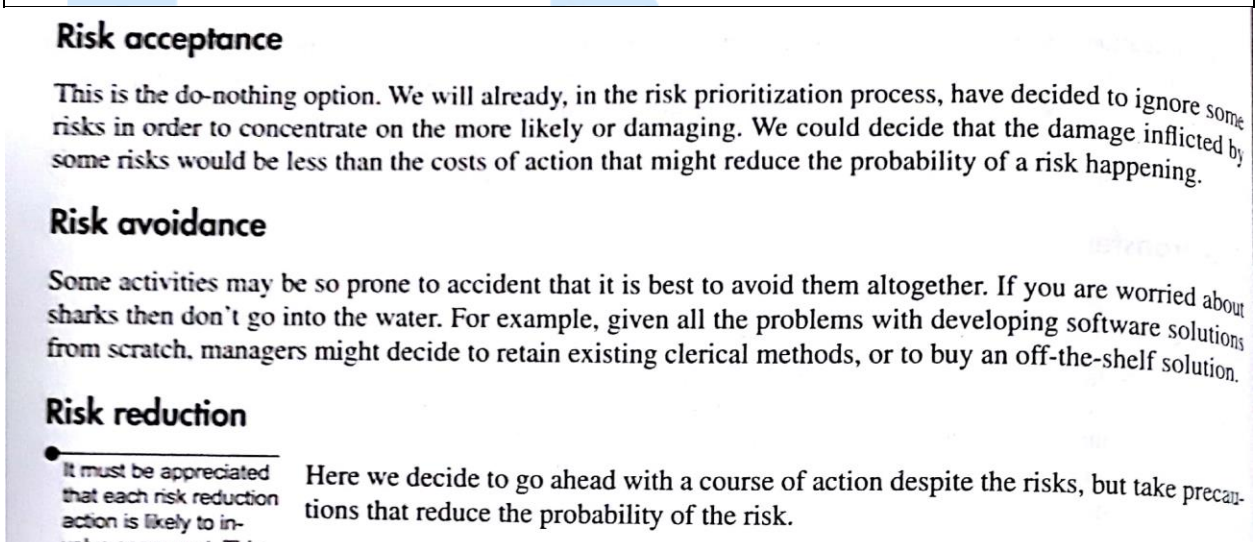
R1, R2 etc refer to particular risks. They are located on the grid according to the likelihood and impact ratings that have been allocated to them. A zone around the top right hand corner of the grid can be designated and risks falling within that zone are treated as requiring urgent action.

Risk planning

Risks can be dealt with by:

- Risk acceptance
- Risk avoidance
- Risk reduction
- Risk transfer
- Risk mitigation/contingency measures

- **Risk acceptance** – the cost of avoiding the risk may be greater than the actual cost of the damage that might be inflicted
- **Risk avoidance** – avoid the environment in which the risk occurs e.g. buying an OTS application would avoid a lot of the risks associated with software development e.g. poor estimates of effort.
- **Risk reduction** – the risk is accepted but actions are taken to reduce its likelihood e.g. prototypes ought to reduce the risk of incorrect requirements
- **Risk transfer** – the risk is transferred to another person or organization. The risk of incorrect development estimates can be transferred by negotiating a fixed price contract with an outside software supplier.
- **Risk mitigation** – tries to reduce the impact if the risk does occur e.g. taking backups to allow rapid recovery in the case of data corruption



Risk acceptance

This is the do-nothing option. We will already, in the risk prioritization process, have decided to ignore some risks in order to concentrate on the more likely or damaging. We could decide that the damage inflicted by some risks would be less than the costs of action that might reduce the probability of a risk happening.

Risk avoidance

Some activities may be so prone to accident that it is best to avoid them altogether. If you are worried about sharks then don't go into the water. For example, given all the problems with developing software solutions from scratch, managers might decide to retain existing clerical methods, or to buy an off-the-shelf solution.

Risk reduction

It must be appreciated that each risk reduction action is likely to in-

Here we decide to go ahead with a course of action despite the risks, but take precautions that reduce the probability of the risk.

volve some cost. This is discussed in the next section.

This chapter started with a scenario where two of the staff scheduled to work on Amanda's development project at IOE departed for other jobs. If this has been identified as a risk, steps might have been taken to reduce possible departures of staff. For instance, the developers might have been promised generous bonuses to be paid on successful completion of the project.

Recall that Brigette had a problem at Brightmouth College: after the purchase of the payroll package, a requirement for the payroll database to be accessed by another application was identified. Unfortunately, the application that had been bought did not allow such access. An alternative scenario might have been that Brigette identified this as a possible risk early on in the project. She might have come across Richard Fairley's four COTS (commercial off-the-shelf) software acquisition risks – see Table 7.5 – where one risk is difficulty in integrating the data formats and communication protocols of different applications. Brigette might have specified that the selected package must use a widely accepted data management system like Oracle that allows easier integration.

TABLE 7.5 Fairley's four commercial off-the-shelf (COTS) software acquisition risks

See R. Fairley (1994) 'Risk management for software projects' *IEEE Software* 11(3) 57–67.

Integration	Difficulties in integrating the data formats and communication protocols of different applications.
Upgrading	When the supplier upgrades the package, the package might no longer meet the users' precise requirements. Sticking with the old version could mean losing the supplier's support for the package.
No source code	If you want to enhance the system, you might not be able to do so as you do not have access to the source code.
Supplier failures or buyouts	The supplier of the application might go out of business or be bought out by a rival supplier.

Risk mitigation can sometimes be distinguished from risk reduction. *Risk reduction* attempts to reduce the likelihood of the risk occurring. *Risk mitigation* is action taken to ensure that the impact of the risk is lessened when it occurs. For example, taking regular back-ups of data storage would reduce the impact of data corruption but not its likelihood. Mitigation is closely associated with contingency planning which is discussed presently.

Risk transfer

Risk transfer is what effectively happens when you buy insurance.

In this case the risk is transferred to another person or organization. With software projects, an example of this would be where a software development task is outsourced to an outside agency for a fixed fee. You might expect the supplier to quote a higher figure to cover the risk that the project takes longer than the 'average' expected time. On the other hand, a well-established external organization might have productivity advantages as its developers are experienced in the type of development to be carried out. The need to compete with other software development specialists would also tend to drive prices down.

7.9 Risk Management

Contingency

Risk reduction activities would appear to have only a small impact on reducing the probability of some risks, for example staff absence through illness. While some employers encourage their employees to adopt a healthy lifestyle, it remains likely that some project team members will at some point be brought down by minor illnesses such as flu. These kinds of risk need a *contingency plan*. This is a planned action to be carried out if the particular risk materializes. If a team member involved in urgent work were ill then the project manager might draft in another member of staff to cover that work.

The preventative measures that were discussed under the 'Risk reduction' heading above will usually incur some cost regardless of the risk materializing or not. The cost of a contingency measure will only be incurred if the risk actually materializes. However, there may be some things that have to be done in order for the contingency action to be feasible. An obvious example is that back-ups of a database have to be taken if the contingency action when the database is corrupted is to restore it from back-ups. There would be a cost associated with taking the back-ups.

Exercise 7.4

In the case above where staff could be absent through illness, what preconditions could facilitate contingency actions such as getting other team members to cover on urgent tasks? What factors would you consider in deciding whether these preparatory measures would be worthwhile?

Deciding on the risk actions

Five generic responses to a risk have been discussed above. For each actual risk, however, specific actions have to be planned. In many cases experts have produced lists recommending practical steps to cope with the likelihood of particular risks; see, for example, Boehm's 'top ten' software engineering risks in Table 7.1.

Whatever the countermeasures that are considered, they must be cost-effective. On those occasions where a risk exposure value can be calculated as a financial value using the (*value of damage*) × (*probability of occurrence*) formula – recall Section 7.7 – the cost-effectiveness of a risk reduction action can be assessed by calculating the *risk reduction leverage* (*RRL*).

Risk reduction leverage

Risk reduction leverage = $(RE_{before} - RE_{after})$/ (cost of risk reduction)

RE_{before} is risk exposure before risk reduction e.g. 1% chance of a fire causing £200k damage

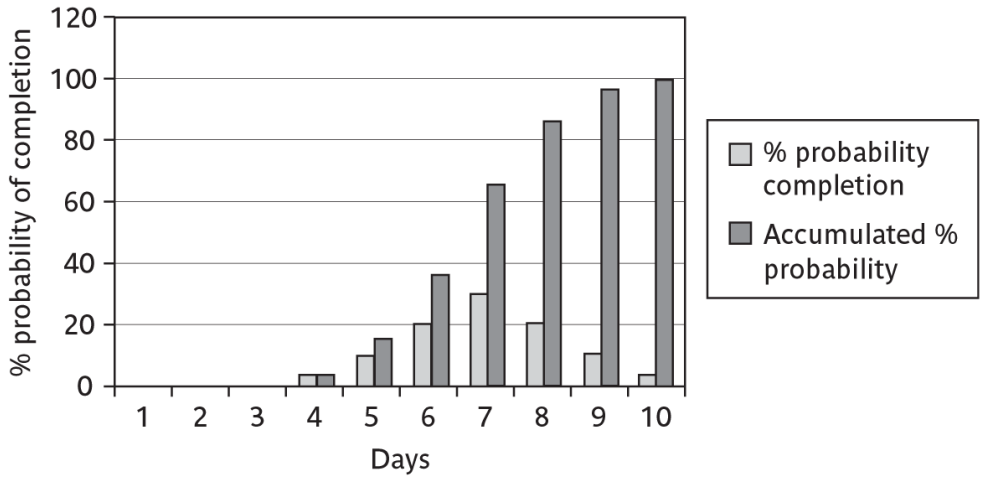
RE_{after} is risk exposure after risk reduction e.g. fire alarm costing £500 reduces probability of fire damage to 0.5%

RRL = (1% of £200k)-(0.5% of £200k)/£500 = 2

RRL > 1.00 therefore worth doing

You could think in terms of the analogy to insurance. An insurance company might reduce the fire insurance premium from £2k to £1k on condition that a fire alarm is installed. The insured would save £1k a year by investing £500 so it would be worth doing.

Probability chart



As was pointed out in Chapter 5, an estimate of activity duration is realistically a range of values clustered around the most likely value are trailing off on either side of this central value.

How can we take account of this in project planning? An answer might be the PERT risk technique.

7.11 Boehm's Top 10 Risks and Counter Measures

Boehm has identified the top 10 risks that a typical project suffers from and has recommended a set of countermeasures for each. We briefly review these in the following.

Barry W. Boehm, "Software Risk Management: Principles and Practices," *IEEE Software*, Volume 8, Issue 1, January 1991.

1. **Personnel shortfall:** This risk concerns shortfall of project personnel. The shortfall may show up as either project personnel may lack some specific competence required for the project tasks or personnel leaving the project (called manpower turnover) before project completion. The countermeasures suggested include staffing with top talent, job matching, team building, and cross-training of personnel.
2. **Unrealistic schedules and budgets:** The suggested counter measures include the project manager working out the detailed milestones and making cost and schedule estimations based on it. Other counter measures are incremental development, software reuse, and requirements scrubbing. It may be mentioned that requirements scrubbing involves removing the overly complex and unimportant requirements, in consultation with the customers.
3. **Developing the wrong functions:** The suggested countermeasures include user surveys and user participation, developing prototypes and eliciting user feedback, and early production users' manuals and getting user feedback on it.

4. **Developing the wrong user interface:** The countermeasures suggested for this risk include prototyping, scenarios and task analysis, and user participation.
5. **Gold-plating:** Gold-plating as discussed in Chapter 1, concerns development of features that the team members consider nice to have and, therefore, decide to develop those even though the customer has not expressed any necessity for those. The countermeasures suggested for this risk includes requirements scrubbing, prototyping and cost-benefit analysis.
6. **Continuing stream of requirements changes:** The countermeasures suggested for this risk include incremental development, high change threshold and information hiding.
7. **Shortfalls in externally-furnished components:** This concerns the risk that the components developed by third party are not up to the mark. The countermeasures suggested for this risk include benchmarking, inspections, reference checking and compatibility analysis.
8. **Shortfalls in externally performed tasks**: This concerns the risk that the work performed by the contractors may not be up to the mark. The countermeasures suggested for this risk include reference checking, pre-award audits, award-fee contracts, competitive design or prototyping and team building.
9. **Real-time performance shortfalls:** The countermeasures suggested for this risk include simulation, benchmarking, modelling, prototyping, instrumentation and tuning.
10. **Straining computer science capabilities**: The countermeasures suggested for this risk include technical analysis, cost-benefit analysis and prototyping.

Using PERT to evaluate the effects of uncertainty

Three estimates are produced for each activity

- *Most likely time (m)*
- *Optimistic time (a)*
- *Pessimistic (b)*
- 'expected time' $t_e = (a + 4m + b) / 6$
- 'activity standard deviation' $S = (b-a)/6$

- *Most likely time (m)* the time we would expect the task to take normally
- *Optimistic time (a)* the shortest time that could be realistically be expected
- *Pessimistic (b)* worst possible time (only 1% chance of being worse, say)
- Some straightforward activities (data input of standing data might perhaps be an example) might have little uncertainty and therefore have a low standard deviation, while others (software design, for instance?) have more uncertainty and would have a bigger standard deviation.

A chain of activities

Task	a	m	b	t_e	s
A	10	12	16	?	?
B	8	10	14	?	?
C	20	24	38	?	?

Fill the missing gaps?

Task A t_e = (10+ (12 x 4) + 16)/ 6 i.e. 12.66 s = (16-10)/6 i.e. 1

Task B t_e = (8 + (10 x 4) + 14)/ 6 i.e. 10.33 s = (14-8)/6 i.e. 1

Task C T_e = (20 + (24 x 4) + 38)/6 i.e. 25.66 s = (38-20)/6 i.e. 3

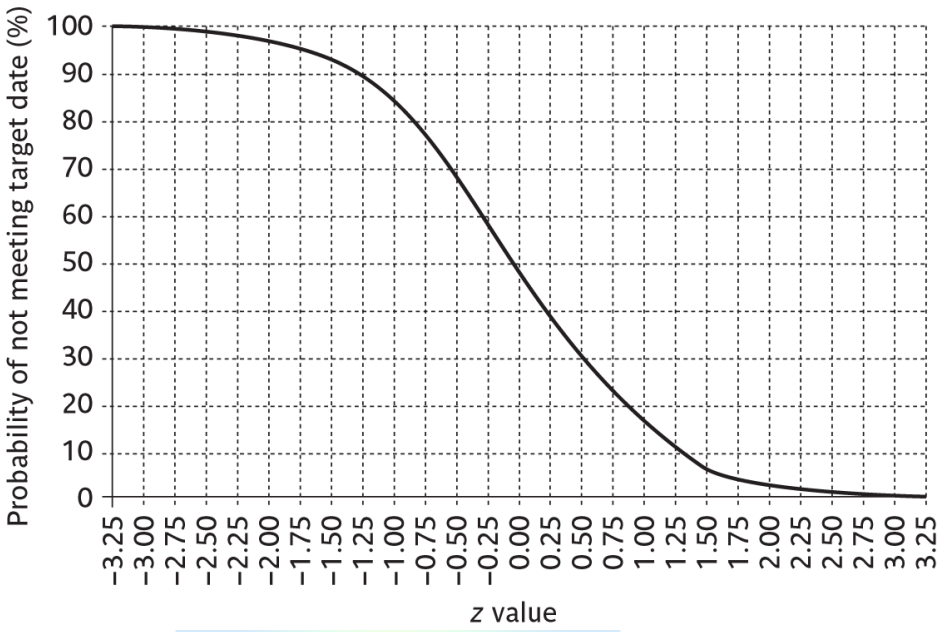
- What would be the expected duration of the chain A + B + C?
- Answer: 12.66 + 10.33 + 25.66 i.e. 48.65
- What would be the standard deviation for A + B+ C?
- Answer: square root of ($1^2 + 1^2 + 3^2$) i.e. 3.32

Assessing the likelihood of meeting a target

- Say the target for completing A+B+C was 52 days (T)
- Calculate the z value thus
  $z = (T - t_e)/s$
- In this example z = (52-48.33)/3.32 i.e. 1.01
- Look up in table of z values – see next overhead

The target, in this case 52 days, is one that could be based on an externally imposed deadline e.g. the start of the London Olympics. The STANDARDIZE calculation in Excel can be used to calculate the Z value.

Graph of z values



There is about a 15% chance of not meeting the target of 52 days. The Excel NORMSDIST can be used to do this calculation.

Monte Carlo Simulation

- An alternative to PERT.
- A class of general analysis techniques:
  - Valuable to solve any problem that is complex, nonlinear, or involves more than just a couple of uncertain parameters.
- Monte Carlo simulations involve repeated random sampling to compute the results.
- Gives more realistic results as compared to manual approaches.

Steps of a Monte Carlo Analysis

1. Assess the range for the variables being considered.
2. Determine the probability distribution of each variable.
3. For each variable, select a random value based on the probability distribution.
4. Run a deterministic analysis or one pass through the model.
5. Repeat steps 3 and 4 many times to obtain the probability distribution of the model's results.

Critical chain concept

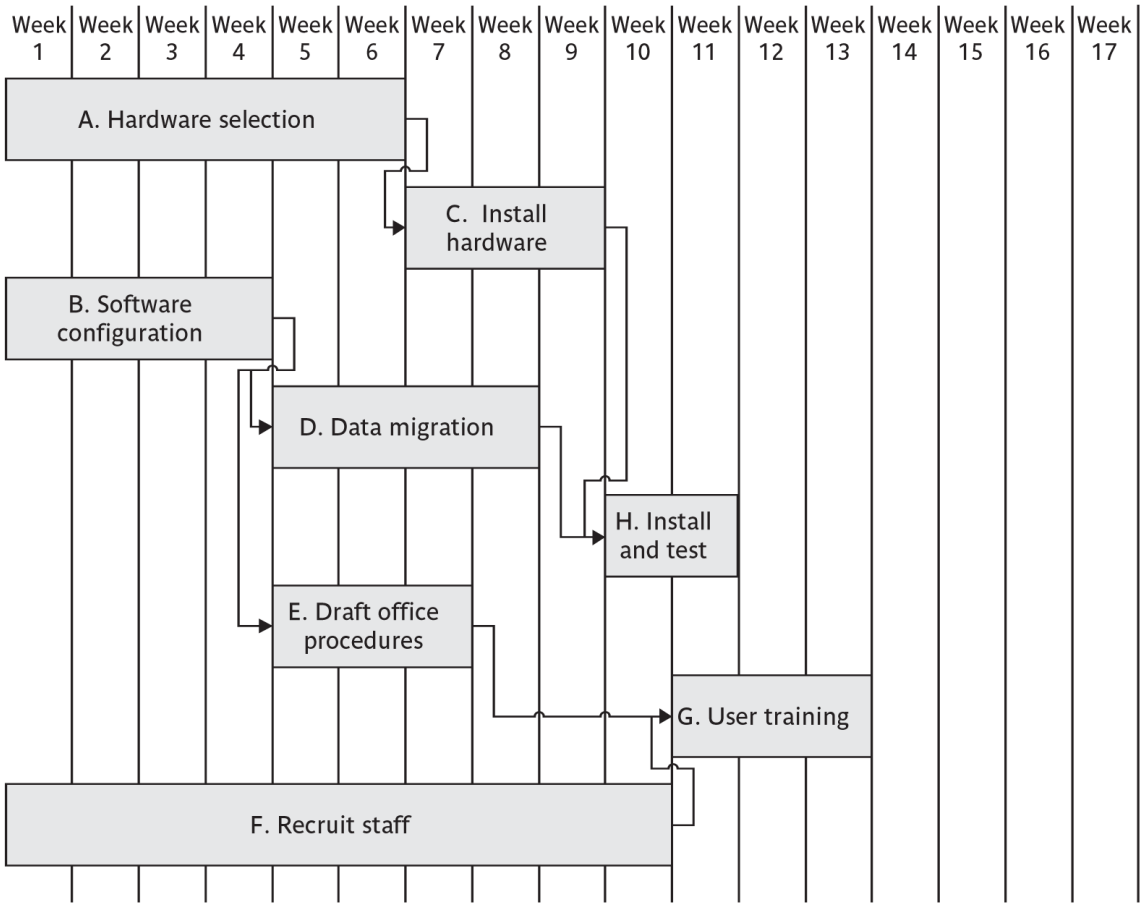


Fig. Traditional planning approach

Note that this plan tends to start activities as early as possible – for example see Activities E and H. While the float on these activities could be used as a buffer if they were late, other activities e.g. F. although of substantial duration do not have the same kind of cushion. In the case of F, the planner would have to make any safety buffer part of the core activity duration.

Critical chain approach

One problem with estimates of task duration:

- Estimators add a safety zone to estimate to take account of possible difficulties
- Developers work to the estimate + safety zone, so time is lost
- No advantage is taken of opportunities where tasks can finish early – and provide a buffer for later activities

Developers will tend to start activities as late as is compatible with meeting the target date as they often have other urgent work to be getting on with in the mean time.

One answer to this:

1. Ask the estimators for two estimates
   - Most likely duration: 50% chance of meeting this
   - Comfort zone: additional time needed to have 95% chance
2. Schedule all activities using most likely values and starting all activities on latest start dates

This approach means that the 'safety buffer' in the estimate for an activity is moved from the individual developer to the project as a whole.

Using latest start dates for activities

Working backwards from the target completion date, each activity is scheduled to start as late as possible. Among other things, this should reduce the chance of staff being pulled off the project on to other work. It is also argued – with some justification according to van Genuchten's research above – that most developers would tend to start work on the task at the latest start time anyway. However, it does make every activity 'critical'. If one is late the whole project is late. That is why the next steps are needed.

Activity	Most likely	Plus comfort zone	Comfort zone
A	6	8	2
B	4	5	1
C	3	3	0
D	4	5	1
E	3	4	1
F	10	15	5
G	3	4	1
H	2	2.5	0.5

TABLE 7.8 Most likely and comfort zone estimates (days)

3. Identify the critical chain – same a critical path but resource constraints also taken into account
4. Put a project buffer at the end of the critical chain with duration 50% of sum of comfort zones of the activities on the critical chain.
5. Where subsidiary chains of activities feed into critical chain, add feeding buffer
6. Duration of feeding buffer 50% of sum of comfort zones of activities in the feeding chain
7. Where there are parallel chains, take the longest and sum those activities

Worked example

Figure 7.12 shows the results of this process. The critical chain in this example happens to be the same as the critical path, that is activities F and G which have comfort zones of 5 weeks and 1 week respectively, making a total of 6 weeks. The project buffer is therefore 3 weeks.

Subsidiary chains feed into the critical chain where activity H links into the project buffer and where activity E links into G which is part of the critical chain. Feeding buffers are inserted at these points. For the first buffer the duration would be 50% of the saved comfort zones of A, C and H, that is $(2 + 0 + 0.5)/2 = 1.25$ weeks. It could be argued that B, D and H could form a feeder chain which also has a combined comfort zone of $(1 + 1 + 0.5)/2 = 1.25$ weeks. In the situations where there are parallel alternative paths on a feeder chain, the practice is to base the feeding buffer on whichever comfort zone total is greater. This because if one or other or both parallel paths were late they could still use the same buffer. (Imagine that in the example above there are two cyclists who live 45 minutes away from work and they both have the same important

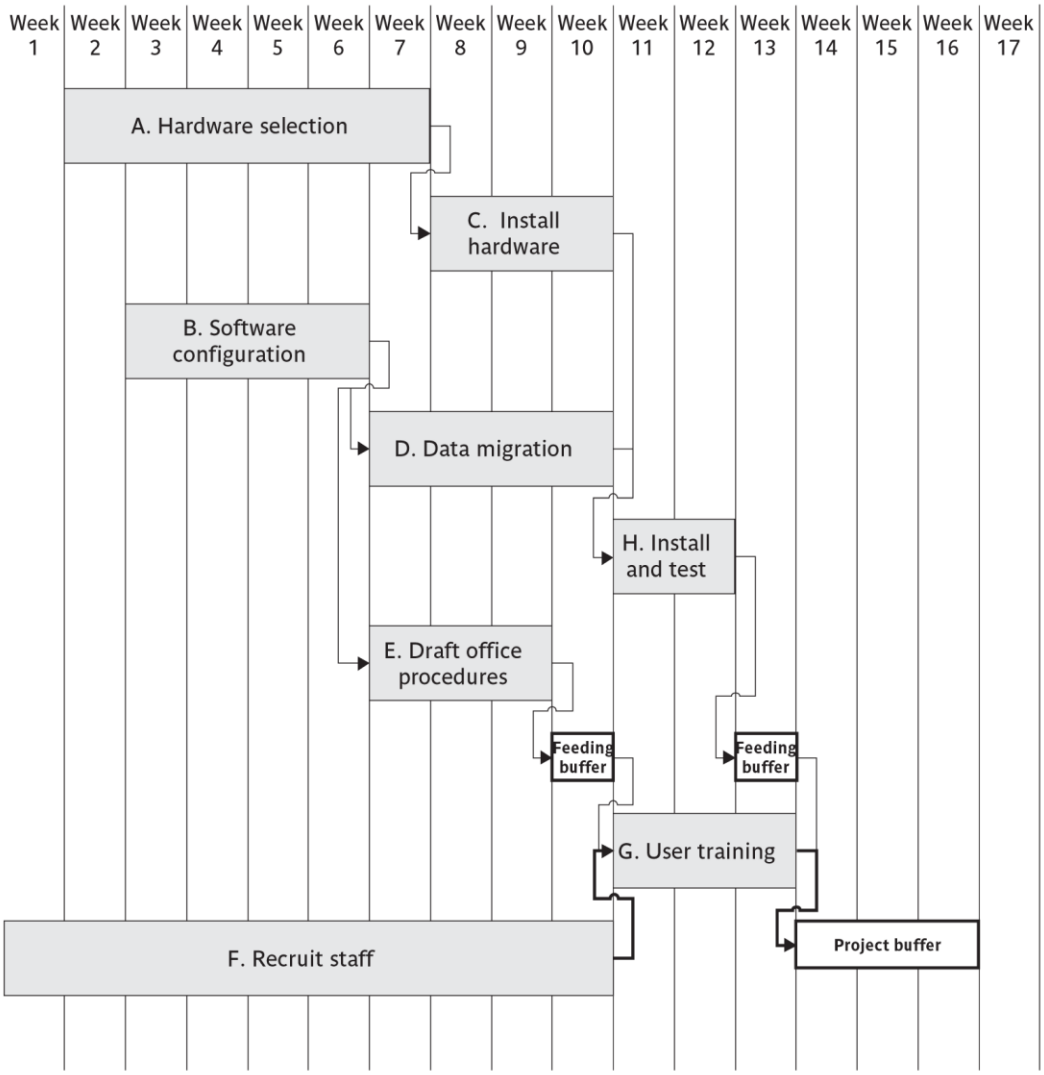


Fig:7.12 Plan employing critical chain concepts

meeting – they might each add a 15-minute comfort zone to the ride on that day but that 15 minutes could effectively be the same 15 minutes between 7.45 and 8.00 a.m. in the morning). It could be argued that the feeding buffer and the final project buffer could also be merged, but explanations of critical chain planning, such as that of Larry Leach (see above), make clear that this is not to be done. This could be because a delay penetrating the feeding buffer time does not affect the completion date of the project, while penetrating the project buffer does.

In the second place, where a feeder chain of activities joins the critical chain, the feeder buffer would be 50% of the comfort zones of activities B and E, that is 1 week.

Executing the critical chain-based plan

- No **chain** of tasks is started earlier than scheduled, but once it has started is finished as soon as possible
- This means the activity following the current one starts as soon as the current one is completed, even if this is early – the relay race principle

Buffers are divided into three zones:

- **Green:** the first 33%. No action required
- **Amber :** the next 33%. Plan is formulated
- **Red :** last 33%. Plan is executed.

Project execution

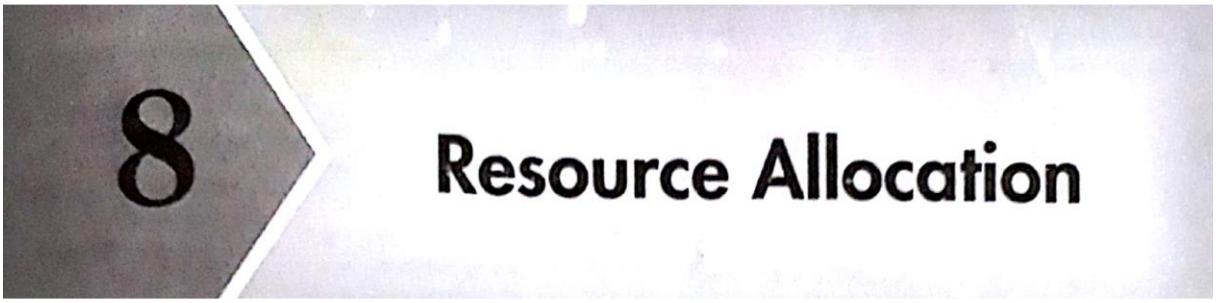
When the project is executed, the following principles are followed:

- No chain of tasks should be started earlier than scheduled, but once it has been started it should be finished as soon as possible – this invokes the *relay race principle*, where developers should be ready to start their tasks as soon as the previous, dependent, tasks are completed.
- Buffers are divided into three zones: green, amber and red, each of an even (33%) size:
  - *green*, where no action is required if the project completion date creeps into this zone;
  - *amber*, where an action plan is formulated if the project completion dates moves into this zone;
  - *red*, where the action plan above is executed if the project penetrates this zone.

Critical chain planning concepts have the support of a dedicated group of enthusiasts. However, the full application of the model has attracted controversy on various grounds. Our personal view is that the ideas of:

- requiring two estimates: the most likely duration/effort and the safety estimate which includes additional time to deal with problems that could arise with the task, and
- placing the contingency time, based on the 'comfort zone' which is the difference between the most likely and safety estimates, in common buffers rather than associating it with individual activities are sound ones that could usefully be absorbed into software project management practice.

See, for example, D. Trietsch (2005) 'Why a critical path by any other name would smell less sweet' *Project Management Journal* 36(1) 27–36 and T. Raz *et al.* (2003) 'A critical look at critical chain project management' *Project Management Journal* 34(4) 24–32.



Schedules

- ***Activity schedule*** - indicating start and completion dates for each activity
- ***Resource schedule*** - indicating dates when resources needed + level of resources
- ***Cost schedule*** showing accumulative expenditure

Resources

- These include
  - labour
  - equipment (e.g. workstations)
  - materials
  - space
  - services
- Time: elapsed time can often be reduced by adding more staff
- Money: used to buy the other resources

8.2 Nature of Resources

A resource is any item or person required for the execution of the project. This covers many things – from paper clips to key personnel – and it is unlikely that we would wish to itemize every resource required, let alone draw up a schedule for their use. Stationery and other standard office supplies, for example, need

not normally be the concern of the project manager – ensuring an adequate supply is the role of the office manager. The project manager must concentrate on those resources which, without planning, might not be available when required.

Some resources, such as a project manager, will be required for the duration of the project whereas others, such as a specific software developer, might be required for a single activity. The former, while vital to the success of the project, does not require the same level of scheduling as the latter. As we saw in Chapter 2, a project manager may not have unrestricted control over a developer who may be needed to work on a range of projects. The manager may have to request the use of a developer who belongs to a pool of resources controlled at programme level.

In general, resources will fall into one of seven categories.

- *Labour* The main items in this category will be members of the development project team such as the project manager, systems analysts and software developers. Equally important will be the quality assurance team and other support staff and any employees of the client organization who might be required to undertake or participate in specific activities.
- *Equipment* Obvious items will include workstations and other computing and office equipment. We must not forget that staff also need basic equipment such as desks and chairs.
- *Materials* Materials are items that are consumed, rather than equipment that is used. They are of little consequence in most software projects but can be important for some – software that is to be widely distributed might, for example, require supplies of disks to be specially obtained.
- *Space* For projects that are undertaken with existing staff, space is normally readily available. If any additional staff (recruited or contracted) should be needed then office space will need to be found.
- *Services* Some projects will require procurement of specialist services – development of a wide area distributed system, for example, requires scheduling of telecommunications services.
- *Time* Time is the resource that is being offset against the other primary resources – project timescales can sometimes be reduced by increasing other resources and will almost certainly be extended if they are unexpectedly reduced.
- *Money* Money is a secondary resource – it is used to buy other resources and will be consumed as other resources are used. It is similar to other resources in that it is available at a cost – in this case interest charges.

The cost of money as a resource is a factor taken into account in DCF evaluation.

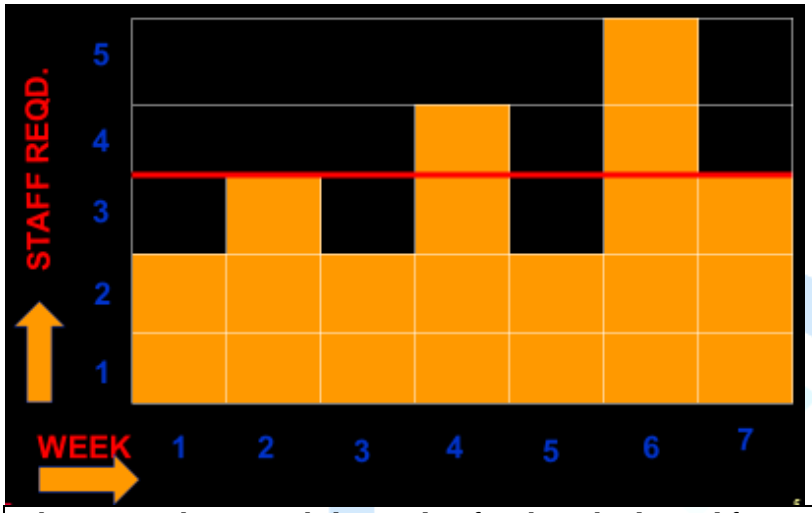
Resource allocation

- Identify the resources needed for each activity and create a ***resource requirement list***
- Identify ***resource types*** - individuals are interchangeable within the group (e.g. 'VB programmers' as opposed to 'software developers')
- Allocate resource types to activities and examine the ***resource histogram***

- This is covered in Section 8.3 of the text.
- Note that at this point we have to assume that we are dealing with, for example, 'standard' software developers who have an average productivity. When we allocate actual people we may find that we have a trainee or a super-expert and this will affect productivity. A shortcoming in productivity in an individual might be compensated for by a lower cost (as would be expected with trainees).

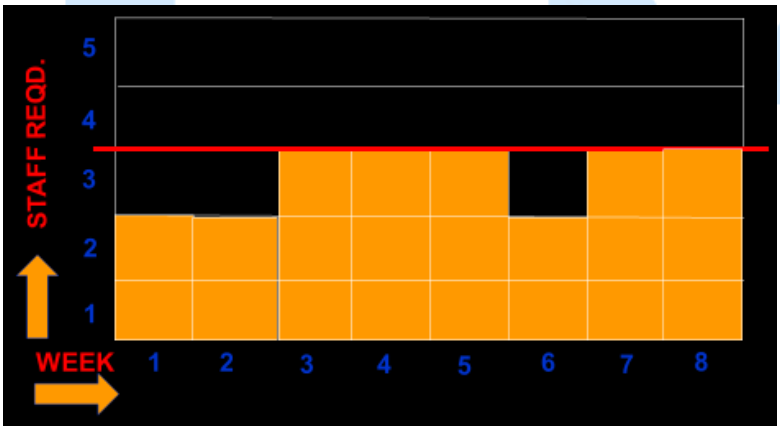
- In the example in the text we start by scheduling every activity to start at the earliest possible date. However in Chapter 7, in the section on the critical chain technique it was suggested that we plan to start activities as late as possible. Whatever the starting procedure, we then need to deal with resource clashes.

Resource histogram: systems analysts



The resource histogram helps us identify where the demand for a resource exceeds the supply.

If we use a tool such as MS Project, the tool will generate the resource histograms for us.



Resource smoothing

- It is usually difficult to get specialist staff who will work odd days to fill in gaps – need for staff to learn about application etc
- Staff often have to be employed for a continuous block of time
- Therefore desirable to employ a constant number of staff on a project – who as far as possible are fully employed
- Hence need for **resource smoothing**

Changing the level of resources on a project over time, particularly personnel, generally adds to the cost of a project. Recruiting staff has costs and, even where staff are transferred internally, time will be needed for familiarization with the new project environment.

The resource histogram in Figure 8.3 poses particular problems in that it calls for two analyst/designers to be idle for twelve days, one for seven days and one for two days between the specification and design stage. It is unlikely that IOE would have another project requiring their skills for exactly those periods of time. This raises the question whether this idle time should be charged to Amanda's project. The ideal resource histogram will be smooth with, perhaps, an initial build-up and a staged run-down.

An additional problem with an uneven resource histogram is that it is more likely to call for levels of resource beyond those available. Figure 8.4 illustrates how, by adjusting the start date of some activities and splitting others, a resource histogram can, subject to constraints such as precedence requirements, be smoothed to contain resource demand at available levels. The different letters represent staff working on a series of module testing tasks, that is, one person working on task A, two on tasks B and C etc.

In Figure 8.4, the original histogram was created by scheduling the activities at their earliest start dates. The resource histogram shows the typical peaked shape caused by earliest start date scheduling and calls for a total of nine staff where only five are available for the project.

By delaying the start of some of the activities, it has been possible to smooth the histogram and reduce the maximum level of demand for the resource. Notice that some activities, such as C and D, have been split. Where non-critical activities can be split they can provide a useful way of filling troughs in the demand for a resource, but in software projects it is difficult to split tasks without increasing the time they take.

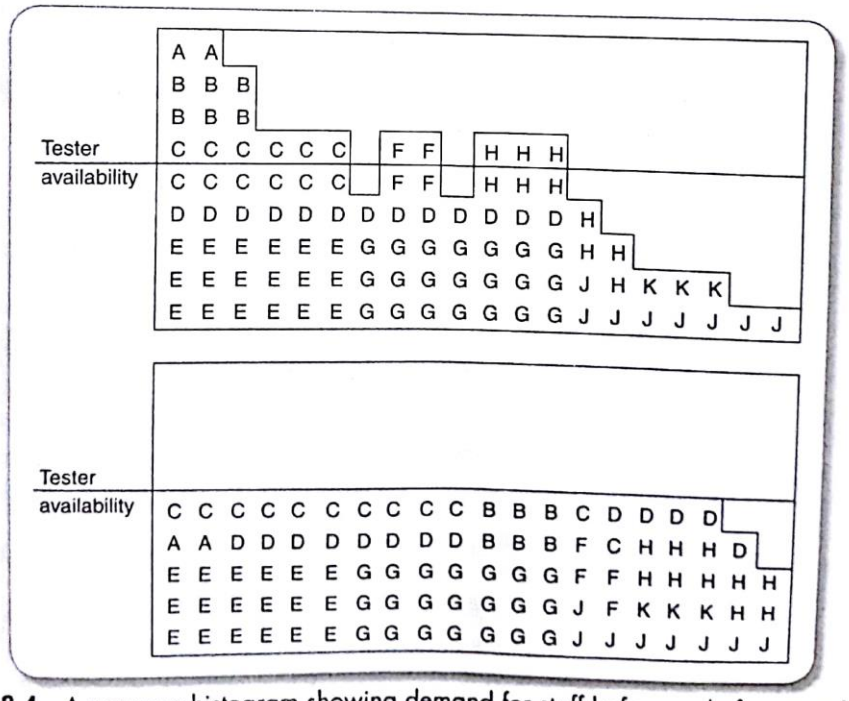


FIGURE 8.4 A resource histogram showing demand for staff before and after smoothing

Resource clashes

- Where same resource needed in more than one place at the same time
- can be resolved by:

- delaying one of the activities
    - taking advantage of float to change start date
    - delaying start of one activity until finish of the other activity that resource is being used on - *puts back project completion*
- moving resource from a non-critical activity
- bringing in additional resource - *increases costs*

Prioritizing activities

There are two main ways of doing this:

- ***Total float priority*** – those with the smallest float have the highest priority
- ***Ordered list priority*** – this takes account of the duration of the activity as well as the float – see next overhead

Where more than one activity is competing for the same limited resource at the same time then those activities need to be prioritized.

Burman's priority list

Give priority to:

- Shortest critical activities
- Other critical activities
- Shortest non-critical activities
- Non-critical activities with least float
- Non-critical activities

Resource usage

- Need to maximise %usage of resources i.e. reduce idle periods between tasks
- Need to balance costs against early completion date
- Need to allow for contingency

Critical path

- Scheduling resources can create new dependencies between activities – recall *critical chains*
- It is best not to add dependencies to the activity network to reflect resource constraints
  - Makes network very messy
  - A resource constraint may disappear during the project, but link remains on network
- Amend dates on **schedule** to reflect resource constraints

- In chapter 7 the concept of critical chains was introduced which took account of resource constraints.
- The point about not adding links to the network to deal with resource constraints is not in the text, but is based on practical experience. The notation for activity networks does not tell you why one activity might be dependent on the completion of another.

Allocating individuals to activities

The initial 'resource types' for a task have to be replaced by actual individuals.

Factors to be considered:

- Availability
- Criticality
- Risk
- Training
- Team building – and motivation

- **Availability** – who is free? Note that this will change during the course of the project as some tasks are completed earlier or later than planned
- **Criticality –** You would want to put your more experienced, 'safer', staff on the critical activities
- **Risk –** this is similar to the point above, but some activities could be off the critical path but still have risks e.g. to the quality of subsequent products
- **Training** – despite concerns about minimizing risk, it is healthy to take some risks in order to develop staff capabilities by allocating challenging tasks to relatively inexperienced stafff

- **Team-building** – identifying people who work well together can pay dividends; chopping and changing plans all the time may in theory optimize project performance, but can in practice be demotivating for staff

Cost schedules

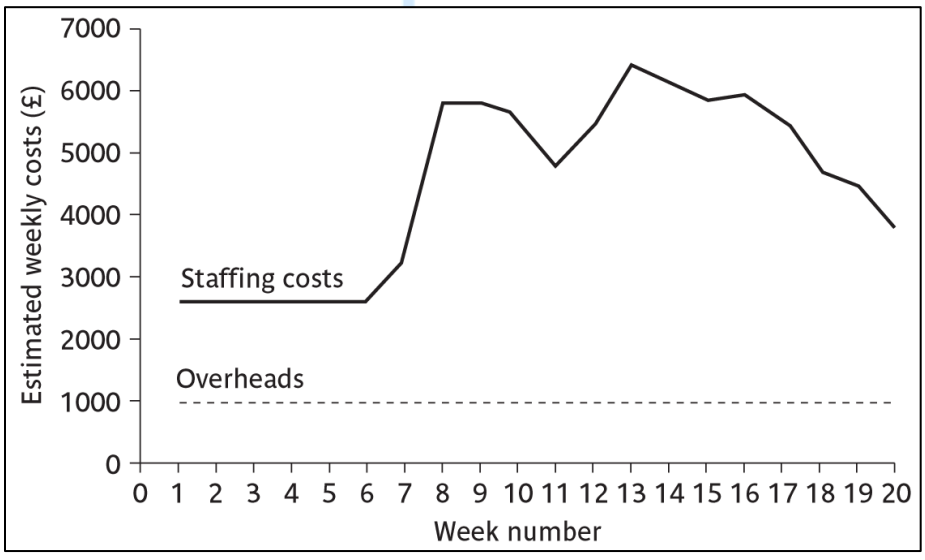
Cost schedules can now be produced:

Costs include:

- Staff costs
- Overheads
- Usage charges

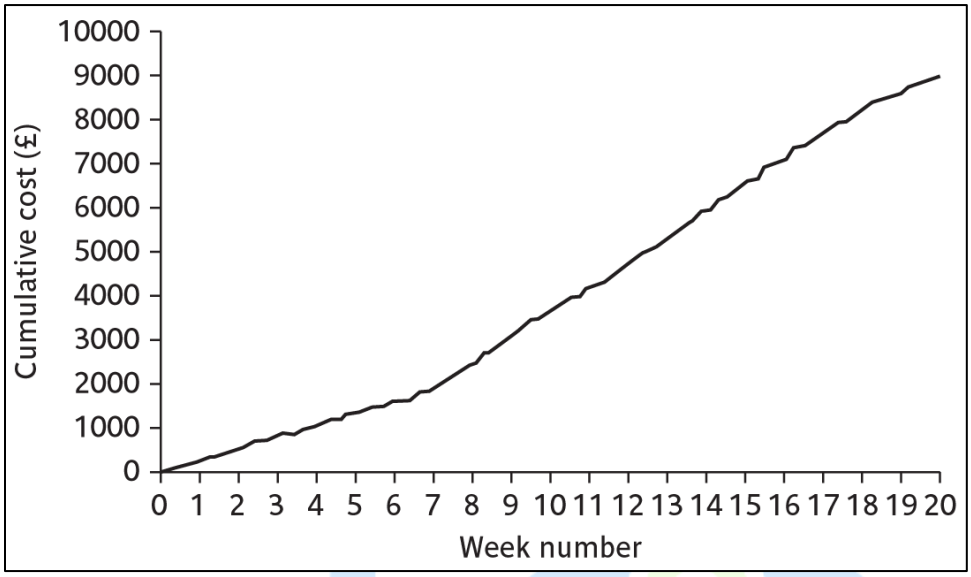
- Section 8.9 covers this in more depth.
- Staff costs – includes not just salary, but also social security contributions by the employer, holiday pay etc. Timesheets are often used to record actual hours spent on each project by an individual. One issue can be how time when a staff member is allocated and available to the project, but is not actually working on the project, is dealt with.
- Overheads e.g. space rental, service charges etc. Some overheads might be directly attributable to the project; in other cases a percentage of departmental overheads may be allocated to project costs.
- Usage charges – some charges can be on a 'pay as you go' basis e.g. telephone charges, postage, car mileage – at the planning stage an estimate of these may have to be made

Cost profile



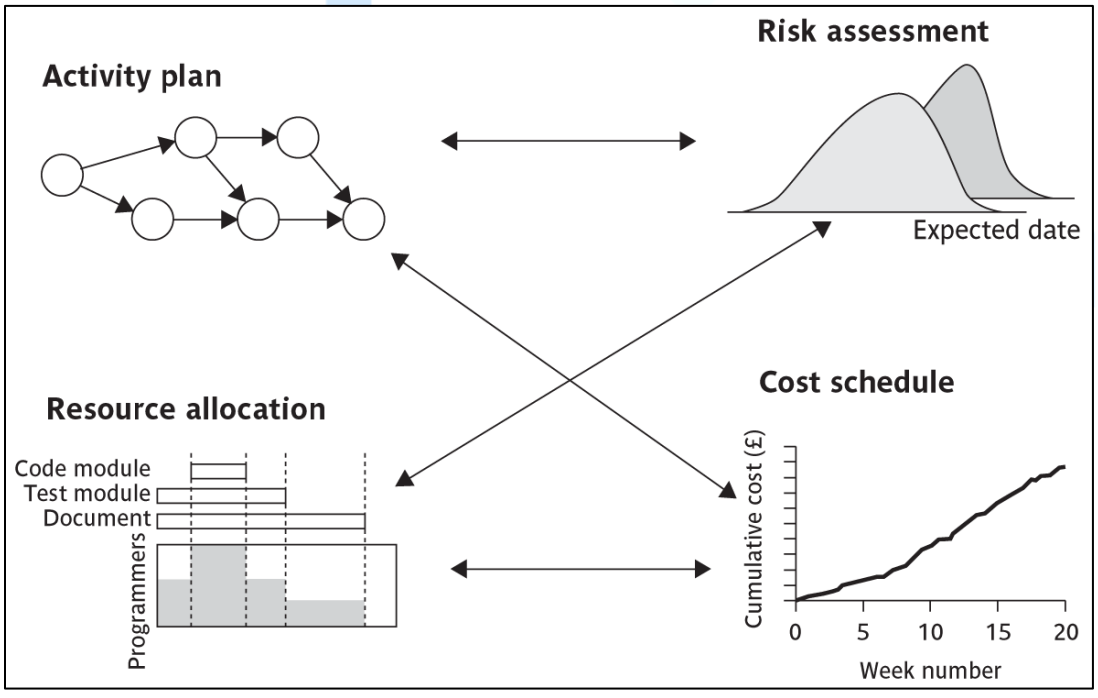
This shows how much is going to be spent in each week. This could be important where an organization allocates project budgets by financial year or quarter and the project straddles more than one of these financial periods

Accumulative costs



The project manager will also be concerned about planned accumulative costs. This chart can be compared to the actual accumulative costs when controlling the project to assess whether the project is likely to meet its cost targets.

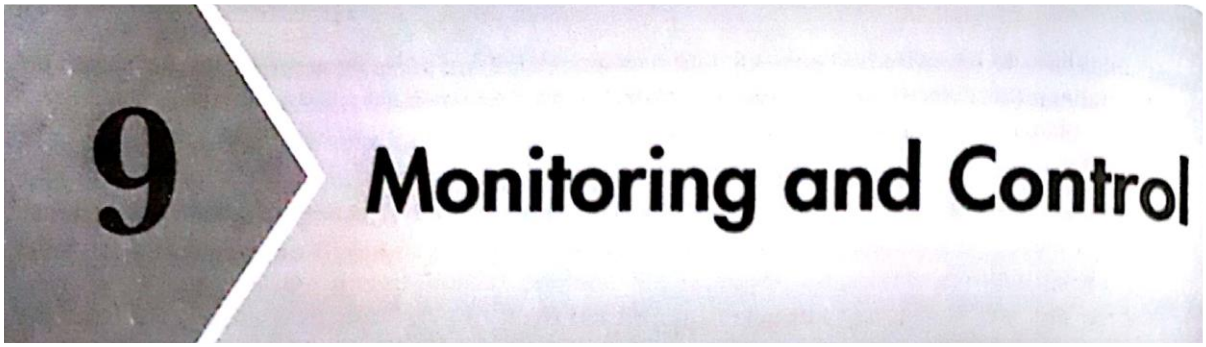
Balancing concerns



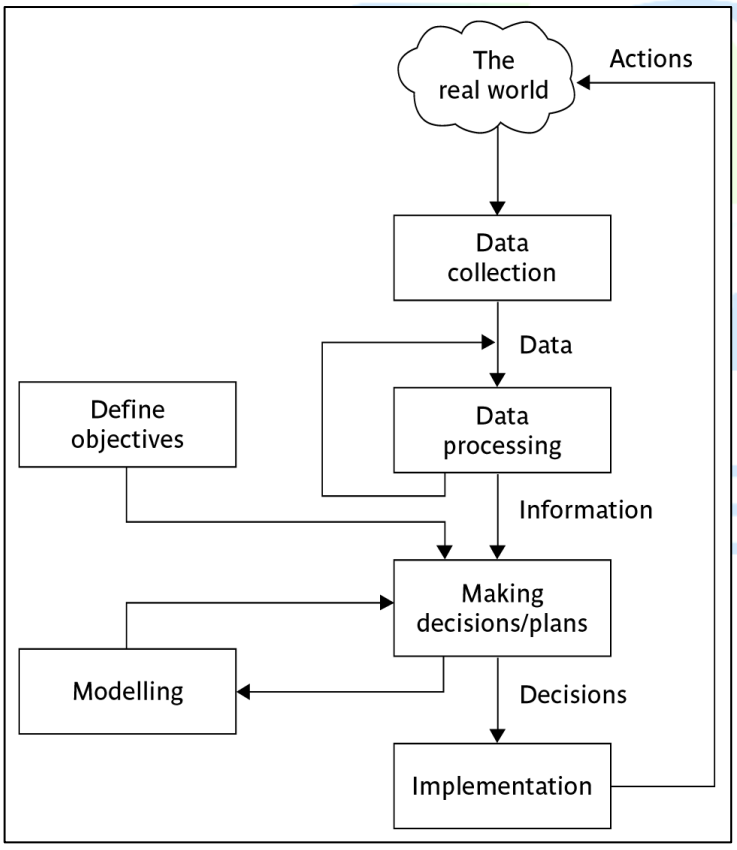
Successful project scheduling is not a simple sequence.

Because of the inter-linking of different concerns project planning will need to be iterative. The consequences of decisions will need to carefully assessed and plans adjusted accordingly.

Unit IV



The control cycle



Define objectives – at the beginning of the project we decide on what we want to achieve

Making decisions/plans – we decide how we are going to achieve the objectives i.e. we create a plan

Modelling – as part of the process of creating a plan we will consider different approaches and attempt to assess the consequences of each of these approaches in terms of how much it will cost and how long it will take, and so on.

Implementation – the plan is now carried out

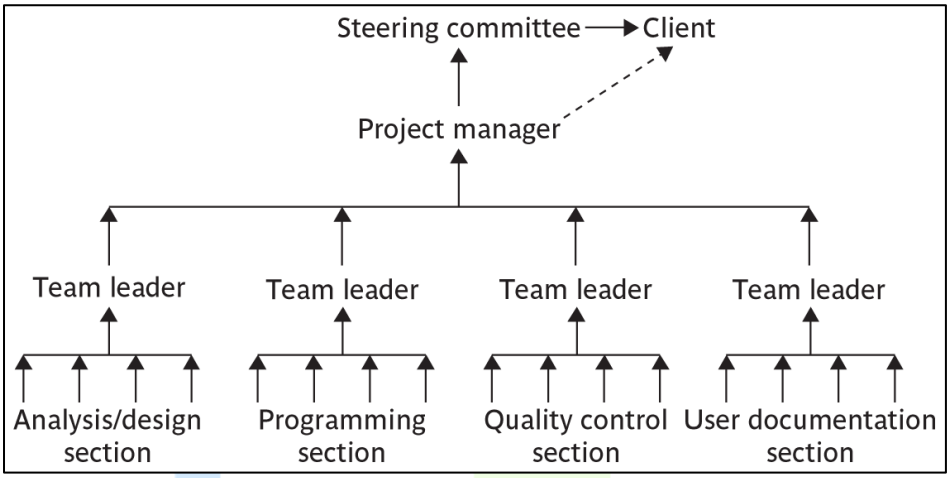
Data collection – we gather information at regular intervals about how the project is progressing. These raw details could be quite numerous and complex on a large project

Data processing – we process the progress data and convert it into 'information' which makes it easier for the project managers and others to understand the overall condition of the project

Making decisions/plans – in the light of the comparison of actual project progress with that planned, the plans are modified. This may require the modelling of the outcomes of different possible courses of action

…and so the cycle goes on.

Responsibilities



The concept of a reporting hierarchy was introduced in Chapter 1.

The main lesson here is that the details relating to project progress have to originate with the people actually doing the work and have then to be fed up through the management structure. At each management level there is going to be some summarising and commentary before information is passed up to the next level. This means that there is always a danger of 'information overload' as information passes from the many to the few.

Assessing progress

Checkpoints – predetermined times when progress is checked

- Event driven: check takes place when a particular event has been achieved
- Time driven: date of the check is pre-determined

Frequency of reporting

The higher the management level then generally the longer the gaps between checkpoints

Collecting progress details

Need to collect data about:

- Achievements
- Costs

A big problem: how to deal with *partial completions*

99% completion syndrome

Possible solutions:

- Control of products, not activities
- Subdivide into lots of sub-activities

- Projects have to be delivered on time and within budget, hence the concern with monitoring achievements and costs.
- Partial completion is where, for example, data is being collected at the end of Week 2 of an activity that should take four weeks. We want to know if it is about 50% completed.
- An example of the '99% completion syndrome' would be in the above case if the developer reported at the end of weeks 1,2 and 3 that the task was respectively 25%, 50% and 75% complete. However at the end of week 4 it is reported that the task is 99% complete. The same thing is reported at the end of week 5 and so on until the task is actually completed.
- Control on products implies that actual examination of intermediate allows us to verify independently and objectively that sub-tasks have been completed.

Red/Amber/Green reporting

- Identify key tasks
  - Break down into sub-tasks
- Assess subtasks as:

Green – 'on target'

Amber – 'not on target but recoverable'

Red – 'not on target and recoverable only with difficulty'

- Status of 'critical' tasks is particularly important

- RAG reporting highlights those activities which need particular attention. The status of a troubled activity might typically move from green to amber; if corrective action is possible it might go back to green, otherwise it could switch to red. If there are lots of instances where activities switch directly from green to red, this could indicate more management control.
- 'Critical tasks' would be those on the critical path and/or reliant on critical resources.

Review

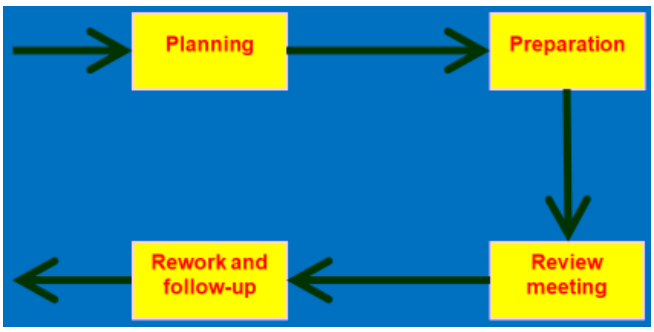
- Review of work products is an important mechanism for monitoring the progress of a project and ensuring the quality of the work products.
- Testing is an effective defect removal mechanism.
  - However, testing is applicable to only executable code.
  - Review is applicable to all work products.

Utility of Review

- A cost-effective defect removal mechanism.
- Review usually helps to identify any deviation from standards.
- Reviewers suggest ways to improve the work product
- a review meeting often provides learning opportunities to not only the author of a work product, but also the other participants of the review meeting.
- The review participants gain a good understanding of the work product under review, making it easier for them to interface or use the work product in their work.

Review Roles

- Moderator:
  - Schedules and convenes meetings, distributes review materials, leads and moderates review sessions.
- Recorder:
  - Records the defects found and the time and effort data.
- Reviewers.



Review Process

Project Termination Review

- Project termination reviews provide important opportunities to learn from past mistakes as well as successes.
- Project termination need not necessarily mean project failure or premature abandonment.
  - A project may be terminated on successful completion

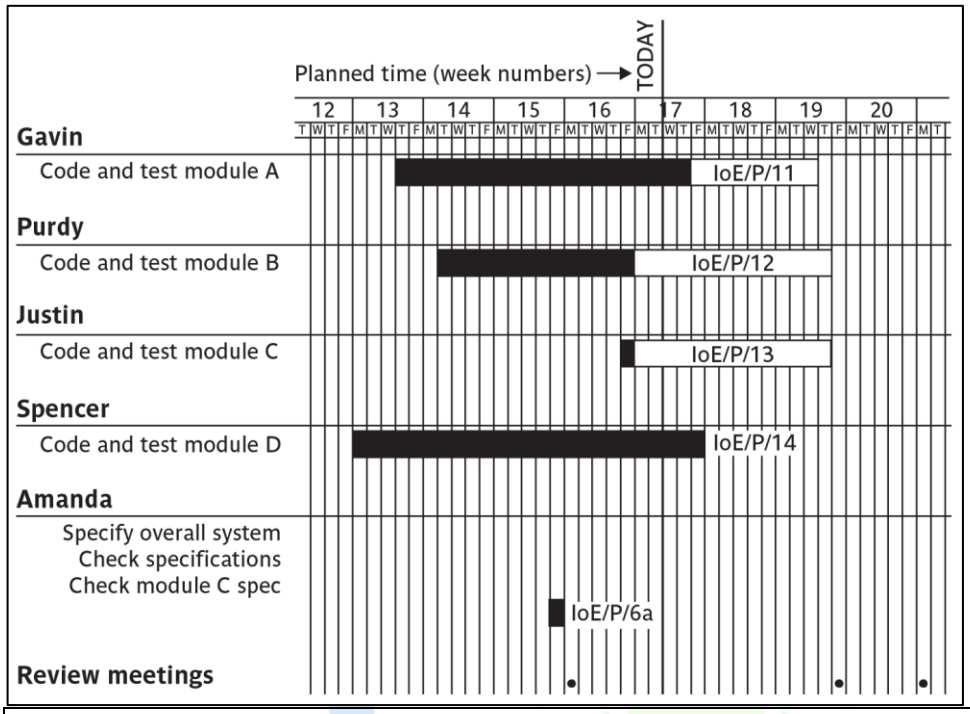
Reasons for Project Termination

- Project is completed successfully handed over to the customer.
- Incomplete requirements
- Lack of resources
- Some key technologies used in the project have become obsolete during project execution
- Economics of the project has changed, for example because many competing product may have become available in the market.

Project Termination Process

- Project survey
- Collection of objective information
- Debriefing meeting
- Final project review
- Result publication

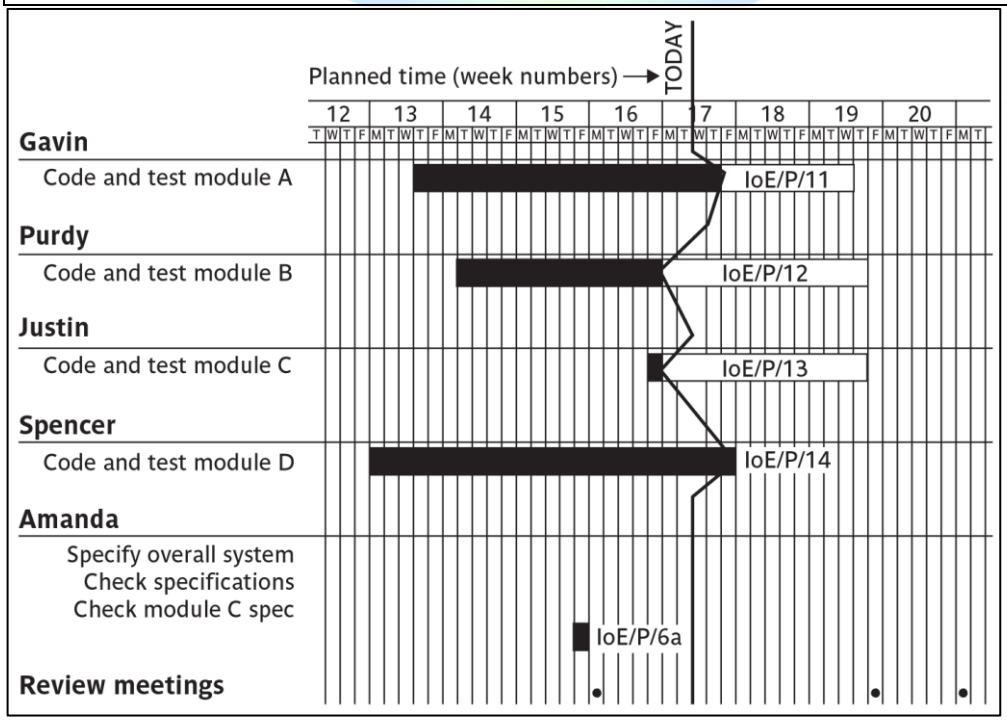
Gantt charts



Note that the Gantt chart is named after Henry Gantt (1861-1919) and so should not be written in capitals!

The format of the Gantt chart here differs from the format used in Microsoft project as the activities for each team member are grouped together. You could input the details so that they came out in this format, but it would not occur automatically.

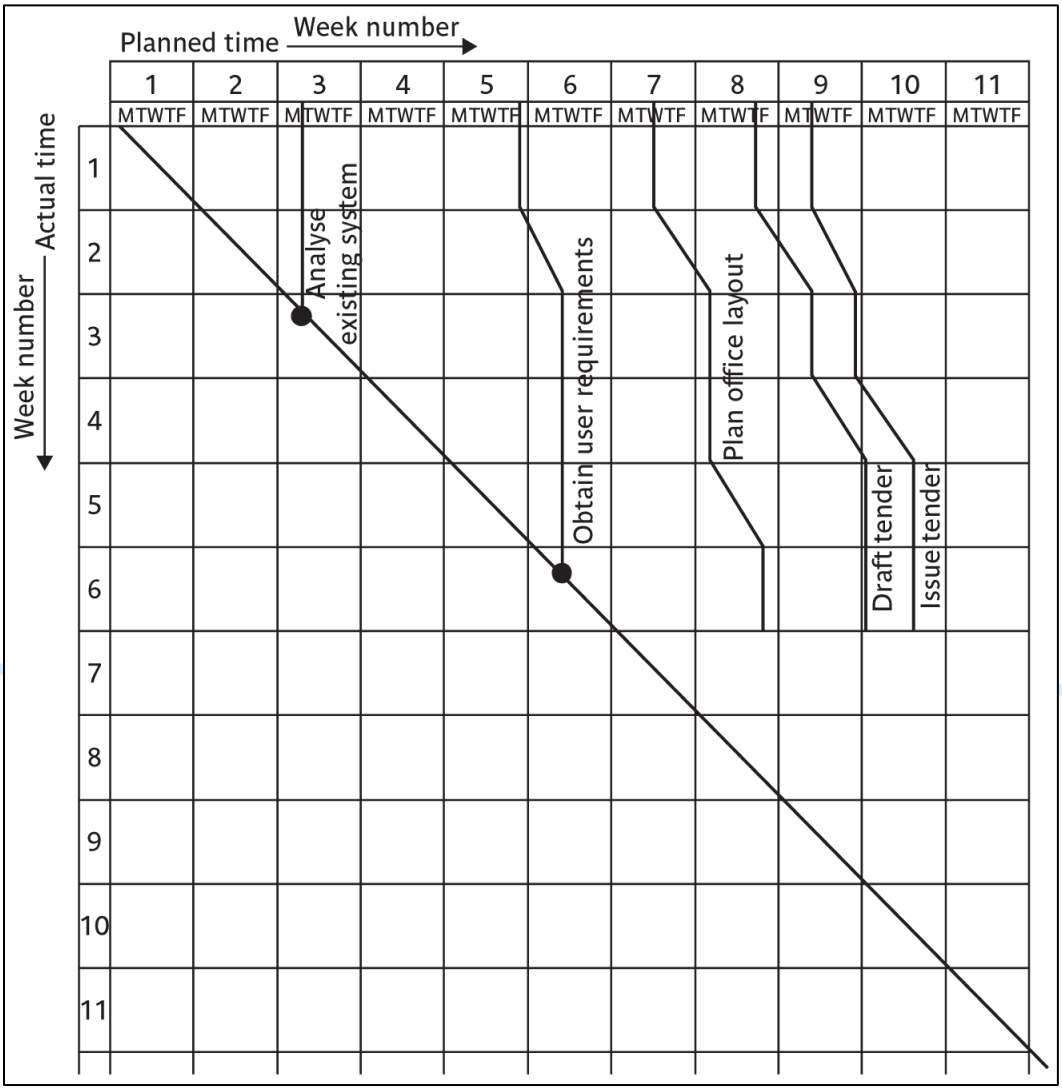
Slip charts



A slip chart is a version of the Gantt chart where a line is drawn from top to bottom. To the left of the line are all the completed activities and to the right those activities (or parts of activities) that have not been completed.

The more jagged the line, the more it means that that there are some activities that are lagging to various degrees and some that are ahead of themselves. A very jagged line means that there is scope for re-planning to move resources from those activities that are ahead to those that are behind.

The timeline



This records the way that targets have changed throughout the project.

Planned time is plotted on the horizontal axis, and actual time on the vertical axis. The bendy lines going from top to bottom represent the scheduled completion date for each activity e.g.

'analyse existing system' – at start this was due finish on the Monday of week 3 and it did finish then

'obtain user requirements' was originally planned to finish on the Thursday of week 5, but at the end of the first week it was rescheduled to finish on the Tuesday of week 6.

Cost monitoring

- A project could be late because the staff originally committed have not been deployed
- In this case the project will be *behind time* but *under budget*
- A project could be on time but only because additional resources have been added and so be *over budget*
- Need to monitor both achievements and costs

Earned value analysis

- *Planned value (PV)* or *Budgeted cost of work scheduled (BCWS)* – original estimate of the effort/cost to complete a task (compare with idea of a 'price')
- *Earned value (EV)* or *Budgeted cost of work performed* (BCWP) – total of PVs for the work completed at this time

Accounting conventions

- Work completed allocated on the basis
  - *50/50* half allocated at start, the other half on completion. These proportions can vary e.g. 0/100, 75/25 etc
  - *Milestone* current value depends on the milestones achieved
  - *Units processed*
- Can use money values, or staff effort as a surrogate

Earned value – an example

- Tasks
  - Specify module 5 days
  - Code module 8 days
  - Test module 6 days

- At the beginning of day 20, PV = 19 days
- If everything but testing completed EV = 13 days
- Schedule variance = EV-PV i.e. 13-19 = -6
- Schedule performance indicator (SPI) = 13/19 = 0.68
- SV negative or SPI <1.00, project behind schedule

Earned value analysis – actual cost

- Actual cost (AC) is also known as Actual cost of work performed (ACWP)
- In previous example, if
  - 'Specify module' actually took 3 days
  - 'Code module' actually took 4 days
- Actual cost = 7 days
- Cost variance (CV) = EV-AC i.e. 13-7 = 6 days
- Cost performance indicator = 13/7 = 1.86
- Positive CV or CPI > 1.00 means project within budget
- CPI can be used to produce new cost estimate
- Budget at completion (BAC) – current budget allocated to total costs of project
- Estimate at completion (EAC) – updated estimate = BAC/CPI
  - e.g. say budget at completion is £19,000 and CPI is 1.86
  - EAC = BAC/CPI = £10,215 (projected costs reduced because work being completed in less time)

Time variance

- Time variance (TV) – difference between time when specified EV should have been reached and time it actually did reach.
- For example say an EV of £19000 was supposed to have been reached on 1st April and it was actually reached on 1st July then TV = - 3 months

Earned value chart with revised forecasts

Activity Assessment Sheet

Staff Justin

Ref: IoE/P/13 Activity: Code and test module C

Week number	13	14	15	16	17	18	
Activity summary	G	A	A	R			
Component							Comments
Screen handling procedures	G	A	A	G			
File update procedures	G	G	R	A			
Housekeeping procedures	G	G	G	A			
Compilation	G	G	G	R			
Test data runs	G	G	G	A			
Program documentation	G	G	A	R			

This shows how the planned value (PV), earned value (EV) and actual cost (AC) can be tracked over the lifetime of a project.

It also shows how the graph can be used to show adjustments to the final estimated cost and duration. A revised assessment of the budget at completion (EAC estimate at completion) can be produced by dividing the original estimated budget at completion (BAC) by the current CPI.

Similarly a forecast of the actual duration of the project can be derived by dividing the original estimated duration by the SPI.

Prioritizing monitoring

We might focus more on monitoring certain types of activity e.g.

- Critical path activities
- Activities with no free float – if delayed later dependent activities are delayed
- Activities with less than a specified float
- High risk activities

- Activities using critical resources

- **Critical path activities** – by definition if these are late then the project as a whole will be delayed
- **Activities with no free float** – free float was defined in Chapter 6. A project with no free float will delay following dependent activities, although the project end date may not be directly threatened.
- **Activities with less than a specified float** – projects when being executed can be **very dynamic**: some activities will take longer than estimated others less; this could lead to the critical shifting. Activities with small floats are the most likely to find themselves turned into activities on the critical path if their floats get eroded.
- **High risk activities** – recall the calculation of activity standard deviations in Chapter 7. If the standard deviation for an activity is large, this indicates that there is a lot of uncertainty about how long it will actually take.
  Activities using critical resources – some resources may only be available for a limited period and if the activities that need the resource are delayed the resource could become unavailable.

Getting back on track: options

- Renegotiate the deadline – if not possible then
- Try to shorten critical path e.g.
  - Work overtime
  - Re-allocate staff from less pressing work
  - Buy in more staff
- Reconsider activity dependencies
  - Over-lap the activities so that the start of one activity does not have to wait for completion of another
  - Split activities

- **Renegotiating the deadline** – one way of doing this is to divide the deliverables into 'tranches' (see Chapter 3), delivering the ones most valuable to the client on or before the deadline, but delaying less valuable ones.
- **Shortening the critical path** – the idea is to try to get things done more quickly by adding more staff. Some activities lend themselves to this more readily than others – it is often quite difficult to do this with software development. It also increases costs
- **Reconsidering activity dependencies** – allowing activities to overlap often increases the risk of quality shortfalls

Exception planning

- Some changes could affect
  - Users
  - The business case (e.g. costs increase reducing the potential profits of delivered software product)
- These changes could be to
  - Delivery date
  - Scope
  - Cost
- In these cases an **exception report** is needed
- First stage
  - Write an **exception report** for sponsors (perhaps through project board)
    - Explaining problems
    - Setting out options for resolution
- Second stage
  - Sponsor selects an option (or identifies another option)
  - Project manager produces an **exception plan** implementing selected option
  - Exception plan is reviewed and accepted/rejected by sponsors/Project Board

Change control

The role of configuration librarian:

- Identifying items that need to be subject to change control
- Management of a central repository of the master copies of software and documentation
- Administering change procedures
- Maintenance of access records

- *Identifying items that need to be subject to change control* – it is unlikely, for example, that a feasibility report would be subject to change control once agreement has been obtained to start the project
- *Management of a central repository of the master copies of software and documentation*
- *Administering change procedures* It is important that someone ensures that there is adherence to change control procedures.
- *Maintenance of access records*. A situation to be avoided is where two different developers are making changes to the same software component.

Typical change control process

1. One or more users might perceive the need for a change
2. User management decide that the change is valid and worthwhile and pass it to development management
3. A developer is assigned to assess the practicality and cost of making the change
4. Development management report back to user management on the cost of the change; user management decide whether to go ahead
5. One or more developers are authorized to make copies of components to be modified
6. Copies modified. After initial testing, a test version might be released to users for acceptance testing
7. When users are satisfied then operational release authorized – master configuration items updated

- 1 and 2. The user community itself must come to a consensus about whether a proposal for a change should go forward. A change deemed desirable by one part of the user community could cause opposition with other users.
- 2 and 3. This part of the process often involves a multipart form, initially raised by a user representative and then completed with a response by the developers.
- 4. There could be a change control board with user and developer representatives that oversees this decision-making process
- 5. The configuration librarian would control this release
- 6. Note that it is a copy that is modified; the original would still exist as the current operational version
- 7. The previous version of the configuration items would be archived but preserved. If there are unforeseen problems with the new version when it is made operational then a fall-back to the previous version could be considered

Software Configuration Management (SCM)

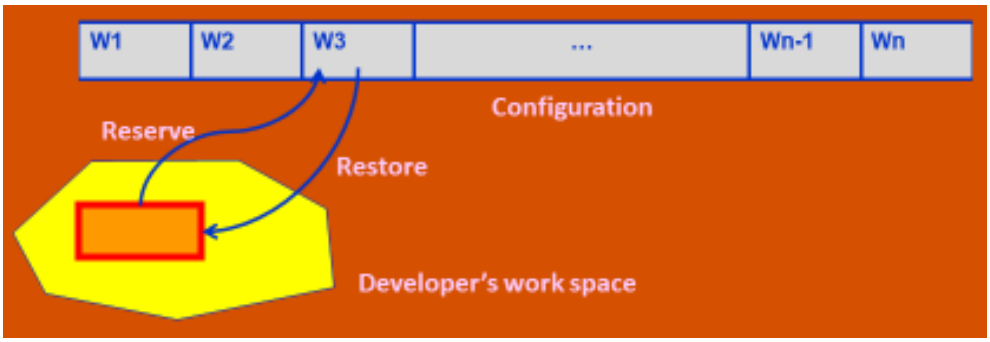
- SCM is concerned with tracking and controlling changes to a software.
- Development and maintenance environment:
  - Various work products associated with the software continually change.
  - Unless a proper configuration management system is deployed, several problems can appear.

Why Use SCM?

- Problems associated with concurrent access
- Undoing Changes
- System accounting
- Handling variants
- Accurate determination project status
- Preventing unauthorized access to the work products

Configuration Control

- Two main operations:
  - Reserve
  - Restore



Acquiring software from external supplier

This could be:

- a *bespoke system* - created specially for the customer
- *off-the-shelf* - bought 'as is'
- *customised off-the-shelf* (COTS) - a core system is customised to meet needs of a particular customer

Section 10.2

Types of contract

- fixed price contracts
- time and materials contracts
- fixed price per delivered unit

Note difference between goods and services

Often licence to use software is bought rather than the software itself

Section 10.4 of the textbook provides more detail about the types of contract.

Fixed price contracts

Advantages to customer

- known expenditure
- supplier motivated to be cost-effective

Disadvantages

- supplier will increase price to meet contingencies
- difficult to modify requirements
- cost of changes likely to be higher
- threat to system quality

Even though the supplier will have to add a margin to the price to deal with contingencies, the cost could still be less than doing the work in-house as the supplier may be able to exploit economies of scale and the expertise that the have from having done similar projects in the past.

When competing for work, there will be pressure on the suppliers to reduce prices. Once a contract has been won and signed, the contractor is in a stronger negotiating position when it come to negotiating the price of additional work as the customer is now locked in.

Time and materials

Advantages to customer

- easy to change requirements
- lack of price pressure can assist product quality

Disadvantages

- Customer liability - the customer absorbs all the risk associated with poorly defined or changing requirements
- Lack of incentive for supplier to be cost-effective

Because suppliers appear to be given a blank cheque, this approach does not normally find favour with customers. However, the employment of contract developers may involve this type of contract.

Fixed price per unit delivered

FP count	Design cost/FP	implement-ation cost/FP	total cost/FP
to 2,000	$242	$725	$967
2,001-2,500	$255	$764	$1,019
2,501-3,000	$265	$793	$1,058
3,001-3,500	$274	$820	$1,094
3,501-4,000	$284	$850	$1,134

This is Table 10.1

These figures do come from a real source (RDI Technologies in the USA). These are now several year old. The bigger the project, the higher the cost per function point. Recall that function points were covered in Chapter 5 on software effort estimation.

Fixed price/unit example

- Estimated system size 2,600 FPs
- Price
  - 2000 FPs x $967 *plus*
  - 500 FPs x $1,019 *plus*
  - 100 FPs x $1,058
  - i.e. $2,549,300
- What would be charge for 3,200 FPs?

2000 FPs at	$967 =	$1,934,000
500 FPs at	$1019 =	$509,500
500 FPs at	$1058 =	$529,000
200 FPs at	$1094	$218,800
total		**$3,191,300**

Advantages for customer

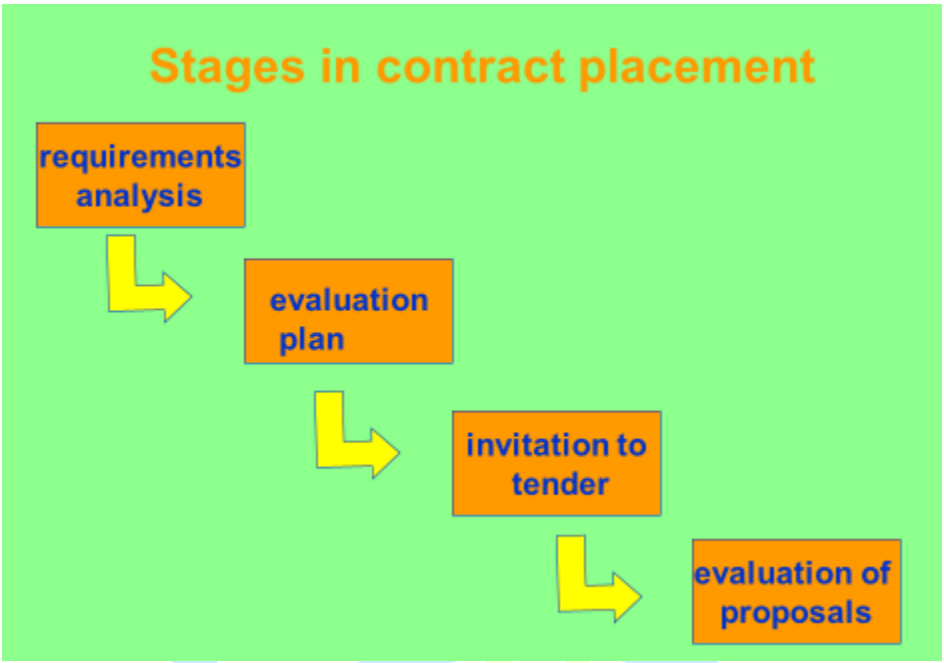
- customer understanding of how price is calculated
- comparability between different pricing schedules
- emerging functionality can be accounted for
- supplier incentive to be cost-effective

Disadvantages

- difficulties with software size measurement - may need independent FP counter
- changing (as opposed to new) requirements: how do you charge?

The tendering process

- **Open tendering**
  - any supplier can bid in response to the *invitation to tender*
  - all tenders must be evaluated in the same way
  - government bodies may have to do this by local/international law (including EU and WTO, World Trade Organization, requirements
- **Restricted tendering process**
  - bids only from those specifically invited
  - can reduce suppliers being considered at any stage
- **Negotiated procedure**
  - negotiate with one supplier e.g. for extensions to software already supplied



Requirements document: sections

- introduction
- description of existing system and current environment
- future strategy or plans
- system requirements -
  - mandatory/desirable features
- deadlines
- additional information required from bidders

- The requirements document is sometimes referred to as the operational requirement or OR.
- If a mandatory requirement cannot be met the proposed application would have to be rejected regardless of how good it might be in other ways.
- A shortfall in one desirable requirement might be compensated for by other qualities or features.

Requirements

- These should include
  - functions in software, with necessary inputs and outputs

- standards to be adhered to
- other applications with which software is to be compatible
- quality requirements e.g. response times

Evaluation plan

- How are proposals to be evaluated?
- Methods could include:
  - reading proposals
  - interviews
  - demonstrations
  - site visits
  - practical tests

Off the shelf software clearly has an advantage here as there is actually product that can be evaluated in existence.

- Need to assess value for money (VFM) for each desirable feature
- VFM approach an improvement on previous emphasis on accepting lowest bid
- Example:
  - feeder file saves data input
  - 4 hours work a month saved at £20 an hour
  - system to be used for 4 years
  - if cost of feature £1000, would it be worth it?

£(4 x 10 x 12 x 4) would be saved i.e. £3,840. The payback period would be just over a year and so this feature would be worth the additional cost.

Invitation to tender (ITT)

- Note that bidder is making an *offer* in response to ITT
- *acceptance* of offer creates a *contract*
- Customer may need further information
- Problem of different technical solutions to the same problem

ISO 12207 refers to an ITT as a Request for Proposal or RFP.

Memoranda of agreement (MoA)

- Customer asks for technical proposals
- Technical proposals are examined and discussed
- Agreed technical solution in MoA
- Tenders are then requested from suppliers based in MoA
- Tenders judged on price
- Fee could be paid for technical proposals by customer

Contracts

- A project manager cannot be expected to be a legal expert – needs advice
- BUT must ensure contract reflect true requirements and expectations of supplier and client

Contract checklist

- Definitions – what words mean precisely e.g. 'supplier', 'user', 'application'
- Form of agreement. For example, is this a contract for a sale or a lease, or a license to use a software application? Can the license be transferred?
- Goods and services to be supplied – this could include lengthy specifications
- Timetable of activities
- Payment arrangements – payments may be tied to completion of specific tasks
- **Ownership of software**

- Can client sell software to others?
- Can supplier sell software to others? Could specify that customer has 'exclusive use'
- Does supplier retain the copyright?
- Where supplier retains source code, may be a problem if supplier goes out of business; to circumvent a copy of code could be deposited with an **escrow** service

- Environment – for example, where equipment is to be installed, who is responsible for various aspects of site preparation e.g. electricity supply?
- Customer commitments – for example providing access, supplying information
- Standards to be met

Contract management

Some terms of contract will relate to management of contract, for example,

- Progress reporting
- Decision points – could be linked to release of payments to the contractor
- Variations to the contract, i.e. how are changes to requirements dealt with?
- Acceptance criteria

How would you evaluate the following?

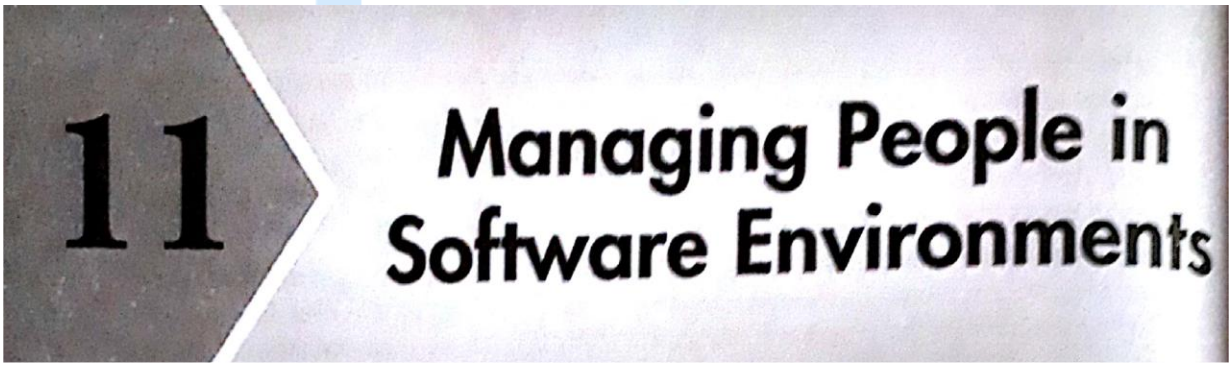
- usability of an existing package
- usability of an application yet to be built
- maintenance costs of hardware
- time taken to respond to requests for software support
- training

- Usability of existing package – you could try out a demo or ask existing users
- Usability of application to be built – here you would have to make stipulation about the process e.g. on the development of interface prototypes; you could also specify performance requirements

- Maintenance costs of hardware – this could be incorporated in a maintenance agreement and you could compare the terms offered by different potential suppliers; another approach is ask current users of the hardware about their experience of it
- Time taken to respond to support requests – this could once again be made a contractual matter and the terms offered by different suppliers could be compared; suppliers could be asked to supply evidence of their past performance (but they might refuse, or not tell the truth); you could ask for references from current customers of the supplier;
- Training – once again references could be taken up; you could ask for the CV of the trainer; you could even get them to give a short presentation

Contract management

- Contracts should include agreement about how customer/supplier relationship is to be managed e.g.
  - *decision points* - could be linked to payment
  - *quality reviews*
  - *changes to requirements*



Main topics

- What is organizational behaviour?
- Staff selection and induction
- Models of motivation – focus on the individual
- The dark side of motivation - stress
- The broader issues of health and safety
- Some ethical and professional concerns

Before organizational behaviour

- Frederick Taylor (1856-1915) 'the father of scientific management'
- Focus:
  - To select the best people for the job;
  - To instruct them in the best methods;
  - To give financial incentives in the form of piece work
- One problem: 'group norms'

- Much of the work of Taylor was in factories and mines, working with manual workers. The 'instruction in best methods' involved breaking down a manual task into its component activities, identifying the best way of carrying out those activities and then teaching the workers to copy the approved method. This can be seen as treating the workers as little better than automatons – but it is also the way the sporting coaches often work!
- The individual workers were encouraged to maximize output by paying them piece-rates e.g. by the units processed.
- One difficulty with this is that workers learn that increasing output can in fact lead to the piece-rate being adjusted in a downward direction. Maximizing output can also be physically and mentally exhausting. Groups of workers therefore tend to converge on an agreed output rate which does not require a constant 100% effort.
- See Sections 11.2 and 11.3.

Hawthorne effect

- 1920's – series of experiments at the Hawthorne Plant of Western Electric, Chicago
- Found that simply showing an interest in a group increased productivity
- Theory X: there is a need for coercion, direction, and control of people at work
- Theory Y: work is as natural as rest or play

- The Hawthorne experiments investigated the effect of various factors such as improved lightning on productivity. It was found that the productivity of the control group (whose working conditions such as lighting were not changed) increased – the fact that someone singled them out for observation improved their motivation.
- Donald McGregor Theory X and Theory Y management approaches.
- See Section 11.3

Selecting the best people

- Belbin distinguishes between **eligible** (having the right qualifications) and **suitable** candidates (can do the job).
- The danger is employ someone who is eligible but not suitable
- The best situation is to employ someone who is suitable but not eligible! For example, these are likely to be cheaper and to stay in the job.

Do good software developers have innate characteristics?

- 1968 study – difference of 1:25 in time taken by different programmers to code program
- Other research found experience better than maths skills as a guide to software skills
- Some research suggested software developers less sociable than other workers
- Later surveys have found no significant social differences between IT workers and others – this could be result of broader role of IT in organizations

- There is some evidence that there is a very wide variation in software development skills – going back many years.
- Some research found that computer people had fewer social needs than other professionals. Later research has not found any significant difference – this may be because the 'ICT profession' has become broader in scope.

A selection process

1. **Create a job specification**.

Content includes types of task to be carried out.

2. **Create a job holder profile**

Describes the characteristics of the person who could do the job

3. **Obtain applicants**

Identify the media that potential job holders are likely to consult. Elicit CVs

4. **Select potential candidates from CVs.**

Do not waste everybody's time interviewing people whose CV clearly indicates are unsuitable.

5. **Further selection, including interview**

Selection processes could include aptitude tests, examination of work portfolios. Make sure selection processes map to the job holder profile

6. **Other procedures**.

e.g. taking up references, medicals etc

Instruction in the best methods

- The induction of new staff should be carefully planned – worst case where new recruit is simply ignored and not given any tasks
- Good induction leads to new recruit becoming productive more quickly
- Need to review staff progress frequently and provide feedback
- Need to identify training that could enhance staff effectiveness.

Section 11.5

Motivation

- Motivation and application can often make up for shortfalls in innate skills
- Taylor's approach - financial incentives
- Abraham Maslow (1908-1970)
  - motivations vary from individual to individual
  - hierarchy of needs – as lower ones fulfilled, higher ones emerge
  - Lowest level – food, shelter
  - Highest level – self-actualization

- Maslow's model implies that people will be motivated by different things at different times. Also that people always feel dissatisfied, but the focus of the dissatisfaction changes over time.
- This and other aspects of motivation are discussed in Section 11.6 of the text
- Exercises 11.2 and 11.3 are designed to get students to think a little about the implications of the Taylor and Maslow models of motivation.

Maslow's Hierarchy of Needs

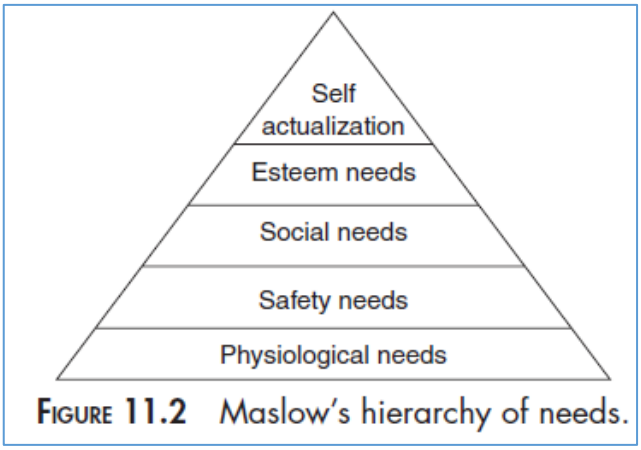


FIGURE 11.2 Maslow's hierarchy of needs.

As a lower level of needs is satisfied then gradually a higher level of need emerges.

Herzberg

Herzberg suggested two sets of factors affected job satisfaction

1. *Hygiene* or *maintenance factors* – make you dissatisfied if they are not right e.g. pay, working conditions
2. *Motivators* – make you feel the job is worthwhile e.g. a sense of achievement

Exercise 11.4 illustrates the approach that Herzberg and associates used to gather data about motivational factors.

Vroom

Vroom and colleagues identified three influences on motivation

1. *Expectancy* – the belief that working harder leads to better performance
2. *Instrumentality* – the belief that better performance will be rewarded
3. *Perceived value* of the reward

- Note: if any of the factors has a zero value, then motivation will be zero.
- Example from the text book: expectancy – trying to use a compiler to compile software code; the code has a bug which causes a compilation error regardless of what you do. In this case motivation will collapse.
- Instrumentality – you are working on removing a fault from a software tool used by a client; you find that the client has given up using the tool and has acquired a different one to do the job. Low perceived value of reward: a reward that everyone gets is less highly regarded than one which only outstanding people get. Getting a first is more valuable if only 5% of students get a first compared to where 90% get a first!

Oldham-Hackman job characteristics

Identified the following characteristics of a job which make it more 'meaningful'

- Skill variety
- Task identity
- Task significance

Two other factors contributed to satisfaction:

- Autonomy
- Feedback

- Skill variety – number of different skills the job holder has the opportunity to exercise
- Task identity – the degree to which your work and its results are associated with you
- Task significance – the degree to which your job has an influence on others
- Two other factors contributed to satisfaction:
- Autonomy – the freedom that you have about the way that you do the job;
- Feedback – the information you get back about the results of your work.
- Software developers will tend to be associated with their code – task identity; analyst programmers will have a to use a wider range of skills than lower level programmers – more skill variety. If you have direct contact with the end-users of your software you are likely to be more aware of the results of your work – task significance, and more likely to get feedback on it.

Methods to improve job satisfaction

- Set specific goals
- Provide feedback on the progress towards meeting those goals
- Consider job redesign
  - Job enlargement
  - Job enrichment

- Job enlargement – widening the range of tasks carried out by a worker
- Job enrichment – delegating some management roles to the worker e.g for re-ordering raw materials.

Stress

- Edward Yourdon quotes a project manager: '*Once a project gets rolling, you should be expecting members to be putting in at least 60 hours a week….The project manager must expect to put in as many hours as possible.'*
- 1960 study in US: people under 45 who worked more than 48 hours a week twice the risk of death from coronary heart disease.
- XP practice – maximum 40 hour working week

Stress can be reduced by good project management

Good project management should lead to:

- Reasonable estimates of effort
- Good project control leading fewer unexpected crises
- Making clear what is expected of each team member – reduces **role ambiguity**
- Reduced **role conflict** where a person is torn between conflicting responsibilities

Bullying tactics are a symptom of incompetent project management.

Stress Management

- Imagery, relaxation, and meditation
  - An example of a simple relaxation technique can be rolling the head from side to side
- Cognitive behaviorial approaches
  - Include self-monitoring of stress intensity, thought record-keeping and rewriting, time management, assertiveness training and increased social interactions.
- Systemic approach
  - Altering the factors which contribute to stress

Health and safety

- Apart from stress, health and safety less likely to be an issue compared to other engineering projects.

- …but sometimes IT infrastructure may be set up as other building work is going on
- UK law lays down that organizations employing over 5 staff should have a **written safety policy**
- Management of safety should be embedded in project management.
- Top management must be committed to health and safety (H&S) policy
- Delegation of responsibilities relating to H&S should be clear
- Job descriptions should include H&S related responsibilities
- Need to ensure those given H&S responsibilities should understand and accept them
- There should be a designated safety officer
- Staff, particularly knowledgeable technical specialists, should consulted about safety
- There should be an adequate H&S budget

Ethical and professional concerns

Ethics relates to the moral obligation to respect the rights and interests of others – goes beyond strictly legal responsibilities

Three groups of responsibilities:

- Responsibilities that everyone has
- Responsibilities that people in organizations have
- Responsibilities relating to your profession or calling

Organizational ethics

There are some who argue that ethical organizational ethics are limited:

Stockholder theory (e.g. Milton Friedman). An employee's duty is to the owners of the business (which often means the stakeholders) above all others – although legal requirements must be met.

Competitive relationships between businesses. Competition may cause you to do things that could have a negative impact on the owners or employees of competitive businesses

Exercise

Identify some of the possible objections and criticisms that can be made of the stockholder business ethics model described above.

Giving this exercise to students can be an interesting (and sometimes scary) experience. The argument against stockholder theory is that the work of an organization is not purely that of money-making. Banks, for example, have a social and economic role facilitating day to day commerce and business. If the owners of the banks decided to opt out of this role the community at large (through the government) would have to step in to ensure that that role is still carried out. Freidman's arguments unwittingly support a greater role for state ownership of important organizations.

Professional ethics

- Professionals have knowledge about the technical domain that the general public does not
- Ethical duty of the expert to warn lay people of the risks involved in a particular course of action
- Many professions, or would be professions, have codes of conduct for their members e.g.

<http://www.bcs.org/upload/pdf/cop.pdf>

<http://www.ieee.org/web/aboutus/ethics>

<http://www.apm.org/about/se_code>

Unit V

Becoming a team

Five basic stages of development:

- Forming
- Storming
- Norming
- Performing
- Adjourning

Classification associated with Tuckman and Jensen

1. The members of the group get to know one another and try to set up some ground rules about behaviour
2. Storming – conflicts arise as various members of the group try to exert leadership and the group's methods of working are established
3. Norming – conflicts are largely settled and a feeling of group identity emerges
4. Performing – the emphasis is now on the tasks at hand
5. Adjourning – the group disbands

One way of attempting to accelerate this process is through team-building exercises.

This is covered in Section 12.2 of the text book.

Balanced teams

- Meredith Belbin studied the performance of top executives carrying out group work at the Hendon Management Centre
- Tried putting the 'best' people together in 'Apollo' teams – almost invariably did badly
- Identified the need for a balance of skills and management roles in a successful team

Management team roles

- The co-ordinator – good at chairing meetings
- The 'plant' – an idea generator
- The monitor-evaluator – good at evaluating ideas
- The shaper – helps direct team's efforts
- The team worker – skilled at creating a good working environment

Belbin management roles – continued

- The resource investigator – adept at finding resources, including information
- The completer-finisher – concerned with getting tasks completed
- The implementer – a good team player who is willing to undertake less attractive tasks if they are needed for team success
- The specialist – the 'techie' who likes to acquire knowledge for its own sake

- The 'specialist' was added by Belbin in 1996.
- A person can have elements of more than one of the types, but usually one or two predominate. According to Belbin about 30% cannot be classified under any heading at all.
- Problems can occur if, for example, you have two or more shapers and no effective co-ordinator – there are likely to be clashes that are difficult to resolve.

Group performance

Some tasks are better carried out collectively while other tasks are better delegated to individuals

- *Additive tasks* – the effort of each participant is summed
- *Compensatory tasks* – the judgements of individual group members are summed – errors of some compensated for by judgements of others

An example of an additive task is clearing snow – in practice there might be some diseconomies of scale if there were too many people involved

Compensatory tasks – using groups of developers to estimate the development effort needed to develop new software components

- *Disjunctive tasks* – there is only one correct answer – someone must:
  - Come up with right answer
  - Persuade the other that they are right
- *Conjunctive* – the task is only finished when all components have been completed

Software development would tend to be conjunctive – all the components have to be completed before the system as a whole is deemed to be complete

'Social loafing'

- Tendency for some team participants to 'coast' and let others do the work
- Also tendency not to assist other team members who have problems
- Suggested counter-measures:
  - Make individual contributions identifiable
  - Consciously involve group members ('loafer' could in fact just be shy!)
  - Reward 'team players'

Barriers to good team decisions

- Inter-personal conflicts – see earlier section on team formation
- Conflicts tend to be a dampened by emergence of *group norms* – shared group opinions and attitudes
- *Risky shift* – people in groups are more likely to make risky decisions than they would as individuals

Delphi approach

To avoid dominant personalities intruding the

following approach is adopted

1. Enlist co-operation of experts
2. Moderator presents experts with problem
3. Experts send in their recommendations to the moderator
4. Recommendations are collated and circulated to all experts
5. Experts comment on ideas of others and modify their own recommendation if so moved
6. If moderator detects consensus, stop; else back to 4

The Delphi approach was originally developed by the RAND Corporation in the late 1960s. The development of email clearly makes the application of the method much easier to organize.

Team 'heedfulness'

- Where group members are aware of the activities of other members that contribute to overall group success
- Impression of a 'collective mind'
- Some attempts to promote this:
  - Egoless programming
  - Chief programmer teams
  - XP
  - Scrum

Egoless programming

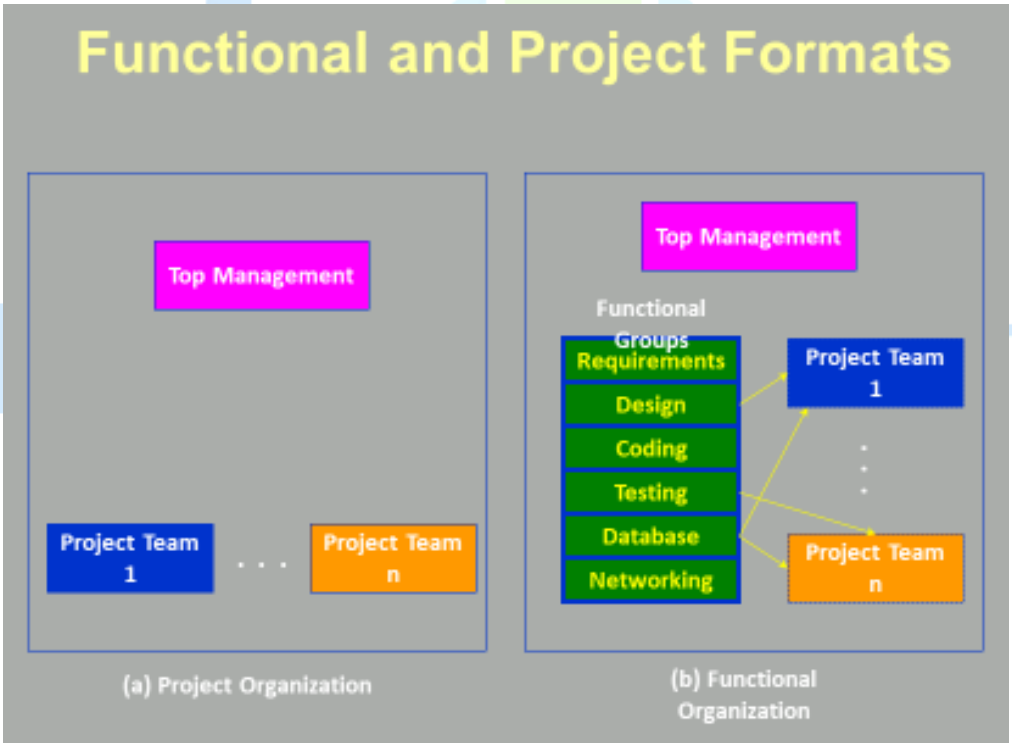
- Gerry Weinberg noted a tendency for programmers to be protective of their code and to resist perceived criticisms by others of the code
- Encouraged programmers to read each others code
- Argued that software should become communal, not personal – hence 'egoless programming'

Organization and Team Structures

- Two important issues that are critical to the effective functioning of every organization are:
  - **Department structure:** How is a department organized into teams?
  - **Team structure:** How are the individual project teams structured?

Department Structure

- **Functional format:**
  - Each functional group comprises of developers having expertise in some specific task or functional area.
- **Project format:**
  - The same team carries out all the project activities.



Functional versus project formats

- Ease of staffing
- Production of good quality documents

- Job specialization
- Efficient handling of the problems associated with manpower turnover
- Career planning

Matrix Format

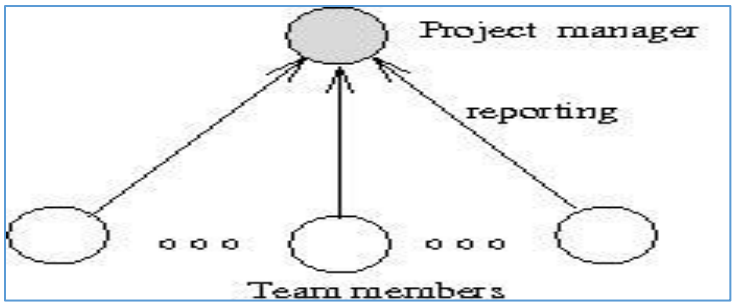
- The pool of functional specialists are assigned to different projects as needed.

	Project			
Functional group	#1	#2	#3	
#1	2	0	3	Functional manager 1
#2	0	5	3	Functional manager 2
#3	0	4	2	Functional manager 3
#4	1	4	0	Functional manager 4
#5	0	4	6	Functional manager 5
	Project manager 1	Project manager 2	Project manager 3	

Team Structure

- We consider only three team structures:
  - Democratic,
  - Chief programmer,
  - Mixed team

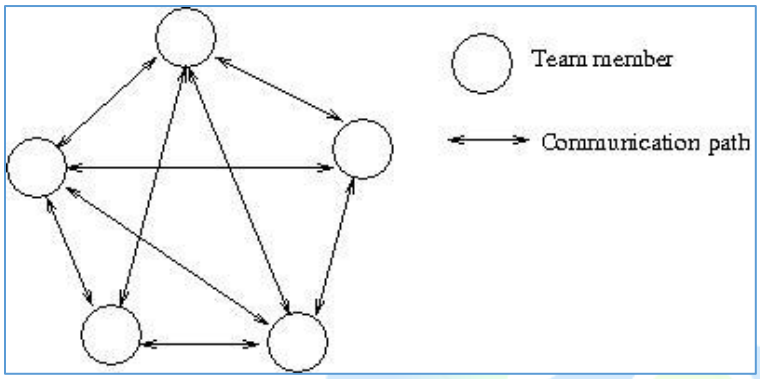
Chief programmer teams



- Fred Brooks was concerned about the need to maintain 'design consistency' in large software systems
- Appointment of key programmers, **Chief Programmers**, with responsibilities for defining requirements, designing, writing and test software code

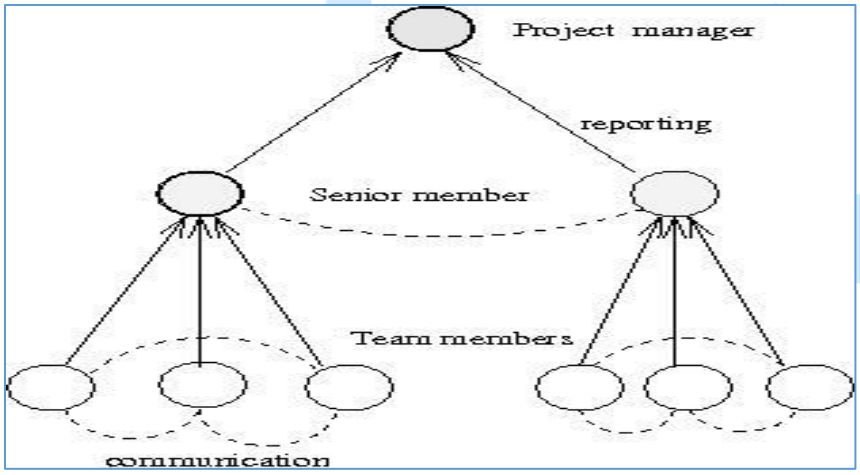
- Assisted by a support team: **co-pilot** – shared coding, **editor** who made typed in new or changed code, **program clerk** who wrote and maintained documentation and **tester**
- Problem – finding staff capable of the chief programmer role

Democratic Team



- Does not enforce any formal team hierarchy.
- Decisions are taken based on discussions,
  - any member is free to discuss with any other member
- Since a lot of debate and discussions among the team members takes place,
  - for large team sizes significant overhead is incurred

Mixed Control Team Structure



- Incorporates both hierarchical reporting and democratic set up.

Extreme programming

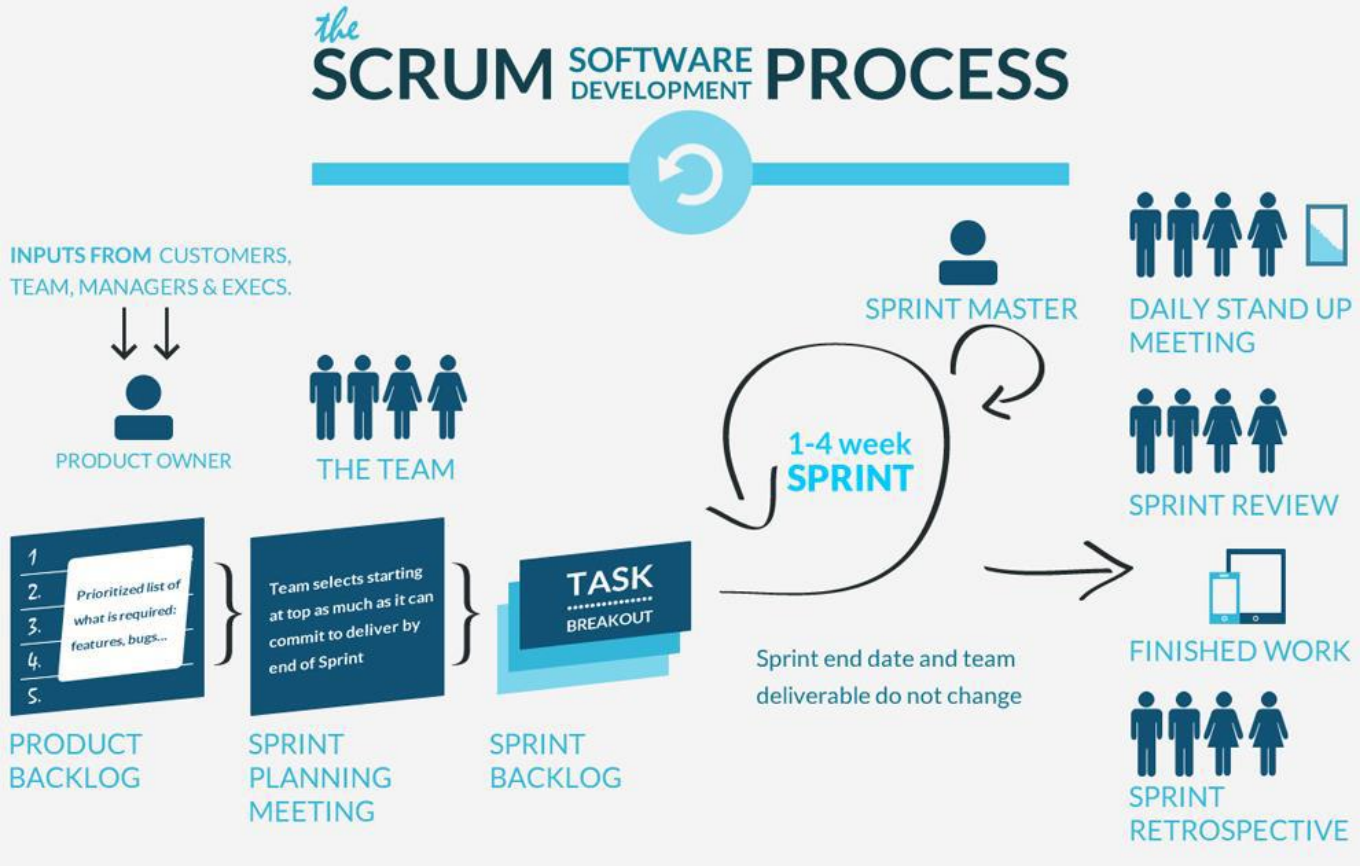
XP can be seen as an attempt to improve team heedfulness and reduce the length of communication paths (the time between something being recorded and it being used)

- Software code enhanced to be self-documenting
- Software regularly refactored to clarify its structure

- Test cases/expected results created *before* coding – acts as a supplementary specification
- Pair programming – a development of the co-pilot concept

Scrum

- Named as an analogy to a rugby scrum – all pushing together
- Originally designed for new product development where 'time-to-market' is important
- 'Sprints' increments of typically one to four weeks
- Daily 'scrums' – daily stand-up meetings of about 15 minutes
- Unlike XP, requirements are frozen during a sprint
- At the beginning of the sprint there is a sprint planning meeting where requirements are prioritized
- At end of sprint, a review meeting where work is reviewed and requirements may be changed or added to



Co-ordination of dependencies

- The previous discussion on team heedfulness focused (mainly) in communication inside the team
- What sort of communications are needed between teams and other units
- Co-ordination theory has identified the following types of coordination:
  - *Shared resources. e.g.* where several projects need the services of scarce technical experts for certain parts of the project.
  - *Producer-customer ('right time') relationships.* A project activity may depend on a product being delivered first.
  - *Task-subtask dependencies.* In order to complete a task a sequence of subtasks have to be carried out.
  - *Accessibility ('right place') dependencies.* This type of dependency is of more relevance to activities that require movement over a large geographical area, but arranging the delivery and installation of IT equipment might be identified as such.
  - *Usability ('right thing') dependencies.* Broader concern than the design of user interfaces: relates to the general question of *fitness for purpose*, e.g. the satisfaction of business requirements.
  - *Fit requirements.* This is ensuring that different system components work together effectively.

Why 'virtual projects'?

The physical needs of software developers (according to an IBM report):

- 100 square feet of floor space
- 30 square feet of work surface
- Dividers at least 6 feet high to muffle noise
- Demarco and Lister found clear statistical links between noise and coding error rates
- One answer: send the developers home!

In practice, most organizations (in the UK at least) pay little attention to creating the optimal software development environment; often they are constrained by the existing structure of buildings.

One solution is to encourage working from home. Taken a step further people at a distance, even in other continents, can be employed.

Possible advantages

- Can use staff from developing countries – lower costs
- Can use short term contracts:
  - Reduction in overheads related to use of premises
  - Reduction in staff costs, training, holidays, pensions etc.
- Can use specialist staff for specific jobs

Most of the advantages are related to cost reduction.

Further advantages

- Productivity of home workers can be higher – fewer distractions
- Can take advantage of time zone differences e.g. overnight system testing

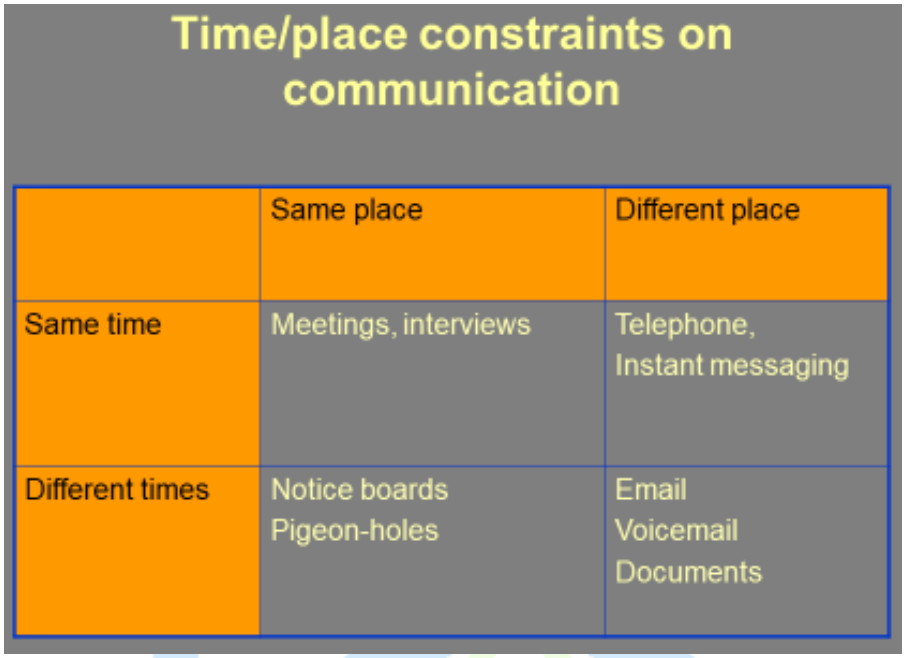
Some challenges

- Work requirements have to be carefully specified
- Procedures need to be formally documented
- Co-ordination can be difficult
- Payment methods need to be modified – piece-rates or fixed price, rather then day-rates

Most of the challenges relate to the organization of staff work. Things have to be spelled out in advance.

More challenges

- Possible lack of trust when there is no face-to-face contact
- Assessment of quality of delivered products needs to be rigorous
- Different time zones can cause communication and co-ordination problems



Other factors influencing communication genres

- Size and complexity of information – favours documents
- Familiarity of context e.g. terminology – where low, two-way communication favoured
- Personally sensitive – it has to be face-to-face communication here

Other factors that would influence the choice of communication methods.

Best method of communication depends on stage of project

- Early stages
  - Need to build trust
  - Establishing context
  - Making important 'global' decisions
  - *Favours same time/ same place*
- Intermediate stages
  - Often involves the paralled detailed design of components
  - Need for clarification of interfaces etc
  - *Favours same time/different place*

Different stages of a project would favour different modes of communication

- Implementation stages
  - Design is relatively clear
  - Domain and context familiar
  - Small amounts of operational data need to be exchanged
  - Favours different time/different place communications e.g. e-mail
- Face to face co-ordination meetings – the 'heartbeat' of the project

Communications plans

- As we have seen choosing the right communication methods is crucial in a project
- Therefore, a good idea to create a **communication plan**
- **Stages** of creating a communication plan
  - Identify all the major stakeholders for the project – see chapter 1
  - Create a plan for the project – see chapter 3
  - Identify stakeholder and communication needs for each stage of the project
  - Document in a communication plan

Content of a communication plan

For each communication event and channel, identify:

- *What*. This contains the name of a particular communication event, e.g, 'kick-off meeting', or channel, e.g. 'project intranet site'.
- *Who/target*. The target audience for the communication.
- *Purpose*. What the communication is to achieve.
- *When/frequency*. If the communication is by means of a single event, then a date can be supplied. If the event is a recurring one, such as a progress meeting then the frequency should be indicated.
- *Type/method*. The nature of the communication, e.g., a meeting or a distributed document.

- *Responsibility*. The person who initiates the communication.

Leadership: types of authority

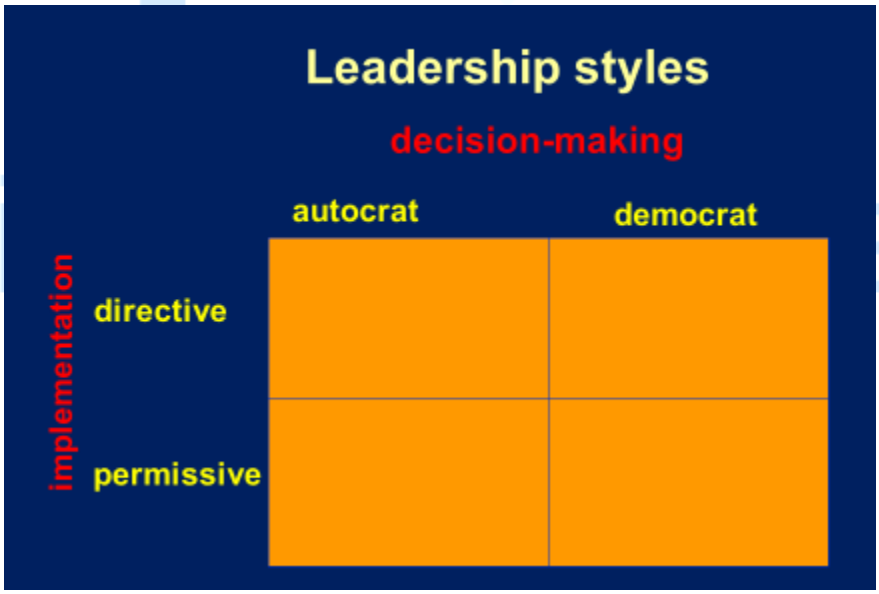
Position power

- Coercive power – able to threaten punishment
- Connection power – have access to those who do have power
- Legitimate power – based on a person's title conferring a special status
- Reward power – able to reward those who comply

Leadership: types of power

Personal power

- *Expert power:* holder can carry out specialist tasks that are in demand
- *Information power*: holder has access to needed information
- *Referent power*: based on personal attractiveness or charisma

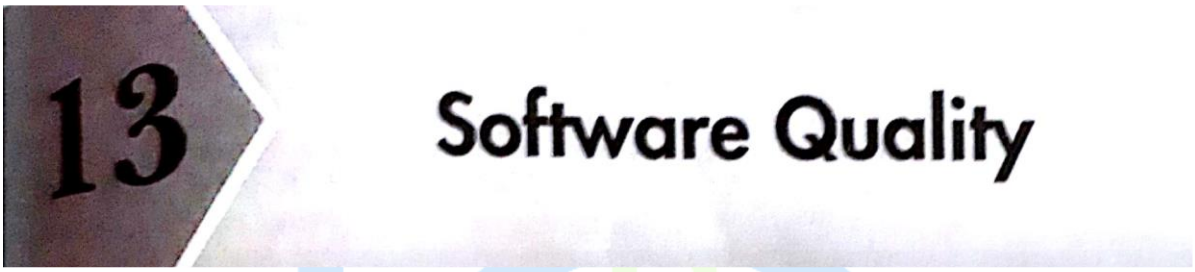


Leadership styles

- Task orientation – focus on the work in hand
- People orientation – focus on relationships

- Where there is uncertainty about the way job is to be done or staff are inexperienced they welcome task oriented supervision
- Uncertainty is reduced – people orientation more important
- Risk that with reduction of uncertainty, managers have time on their hands and become more task oriented (interfering)

Essentially staff want hands-on management when they need guidance. Once the know the job they want to be left to get on with it!



Software product quality

The importance of software quality

- Increasing criticality of software
- The intangibility of software
- Project control concerns:
  - errors accumulate with each stage
  - errors become more expensive to remove the later they are found
  - it is difficult to control the error removal process (e.g. testing)

- Increasing criticality of software – e.g .software is increasingly being used in systems that can threaten or support human life and well-being
- The intangibility of software – it is difficult for observers to judge the quality of software development, especially during its early stages
- Project control concerns:
- The products of one sub-process in the development process are the inputs to subsequent sub-processes, thus
- errors accumulate with each stage e.g. at the design stage, the specification errors are incorporated into the design, and at the coding stage specification and design errors are incorporated into the software

- errors become more expensive to remove the later they are found
- it is difficult to control the error removal process (e.g. testing)
- See Section 13.3

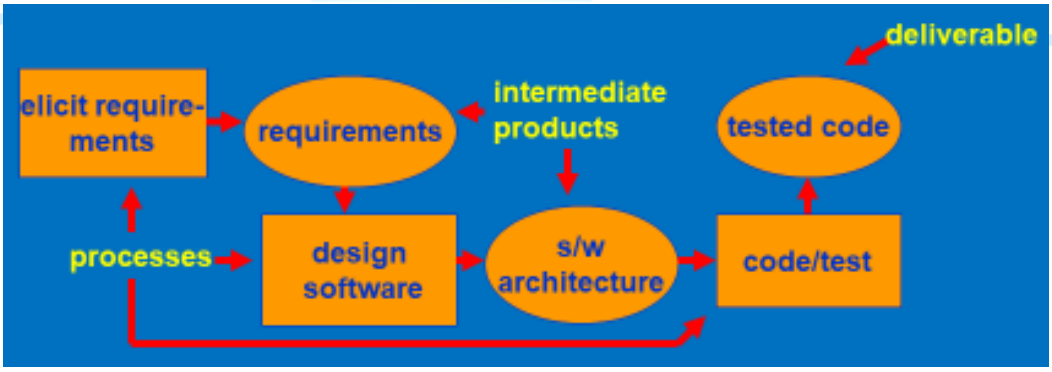
Quality specifications

Where there is a specific need for a quality, produce a quality specification

- Definition/description of the quality
- Scale: the unit of measurement
- Test: practical test of extent of quality
- Minimally acceptable: lowest acceptable value, if compensated for by higher quality level elsewhere
- Target range: desirable value
- Now: value that currently applies

ISO standards: development life cycles

A development life cycle (like ISO 12207) indicates the sequence of *processes* that will produce the software *deliverable* and the *intermediate products* that will pass between the processes.



The *deliverables* are the products that are handed over to the client at the end of the project, typically the executable code.

Intermediate products are things that are produced during the project, but which are not (usually) handed to the client at the end. Typically they are things that are produced by one sub-process (e.g. a requirements document created by the requirements elicitation and analysis processes) and used by others (e.g. a design process which produces a design that fulfils the requirements).

These sub-processes will fit into the overall framework of a *development cycle*.

Some software quality models focus on evaluating the quality of software products, others on the processes by which the products are created.

ISO standards

ISO 9126 Software product quality

Attributes of software product quality

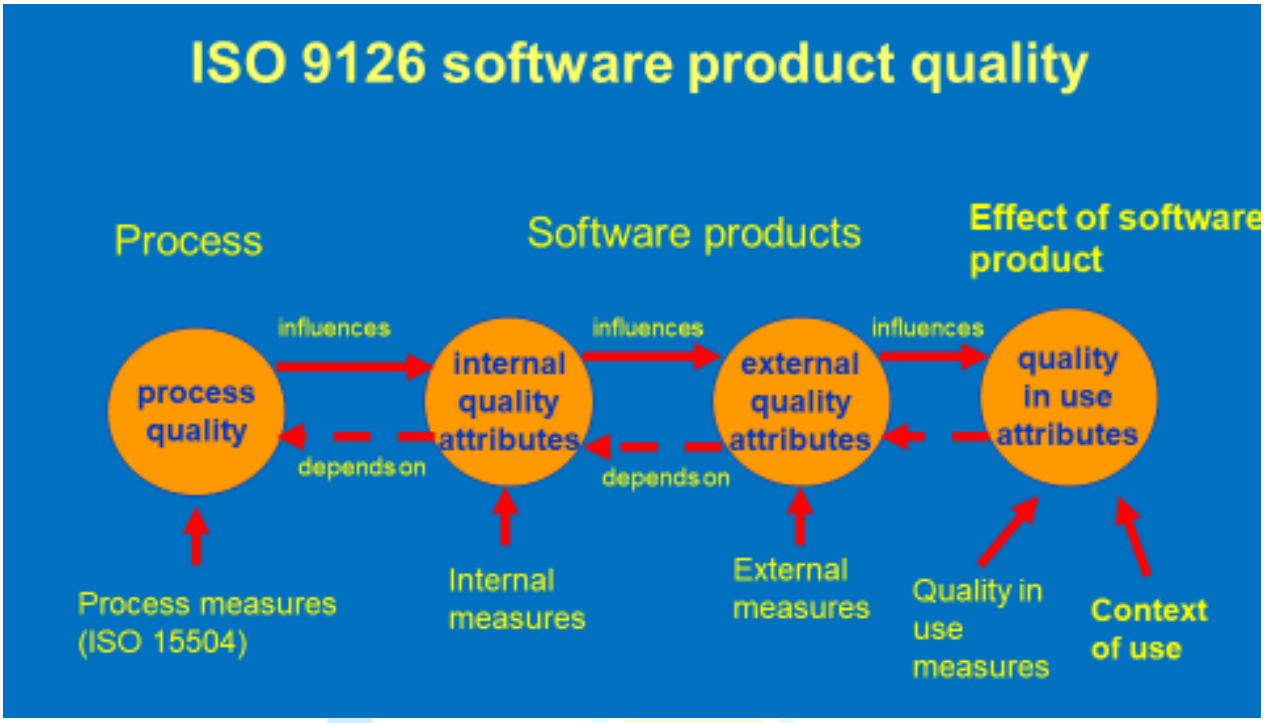
- External qualities i.e. apparent to the user of the deliverable
- Internal qualities i.e. apparent to the developers of the deliverables and the intermediate products

ISO 14598 Procedures to carry out the assessment of the product qualities defined in ISO 9126

ISO 9126 focuses on the definition of software quality, while ISO 14598 focuses on the way that the quality, once defined, is assessed. The ISO 9000 series of standards establishes requirements for quality management systems for the creation/supply of all types of goods and services while the ones mentioned here relate specifically to software.

Types of quality assessment

- During software development:
  - To assist developers to build software with the required qualities
- During software acquisition:
  - To allow a customer to compare and select the best quality product
- For a particular community of users:
  - Independent evaluation by assessors rating a software product



Internal quality attributes are the things that developers would be aware of during the project. They could be qualities in the intermediate products that are created. External quality attributes are the qualities of the final products that are delivered to the users. Thus users would be very aware of these. A key task is mapping these two types of quality.

Acceptable quality in software depends on the use to which the software is put. For example, a higher standard of reliability would be expected of a software component that was very heavily used and the continued functioning of which was essential to the organization, than of a rarely used software tool for which there were many alternatives.

Quality in use

- Effectiveness – ability to achieve user goals with accuracy and completeness
- Productivity – avoids excessive use of resources in achieving user goals
- Safety – within reasonable levels of risk of harm to people, business, software, property, environment etc,
- Satisfaction – happy users!

'users' include those maintain software as well as those who operate it.

Quality Models

- It is a model of the software:
  - Constructed with the objective to describe, assess and/or predict quality.
- Popular models:
  - Garvin's model
  - Boehm's model
  - ISO 9126
  - Dromey's Model

ISO 9126 software qualities

functionality	does it satisfy user needs?
reliability	can the software maintain its level of performance?
usability	how easy is it to use?
efficiency	relates to the physical resources used during execution
maintainability	relates to the effort needed to make changes to the software
portability	how easy can it be moved to a new environment?

These are the six top-level qualities identified in ISO9126.

Note that 'functionality' is listed as a quality. Sometimes 'functional' requirements are distinguished from 'non-functional' (i.e. quality) requirements.

Sub-characteristics of Functionality

- Suitability
- Accuracy

- Interoperability
  - ability of software to interact with other software components
- Functionality compliance
  - degree to which software adheres to application-related standards or legal requirements e.g audit
- Security
  - control of access to the system

In the case of interoperability, a good example is the ability to copy and paste content between different Microsoft products such as Excel and Word.

Sub-characteristics of Reliability

- Maturity
  - frequency of failure due to faults - the more the software has been used, the more faults will have been removed
- Fault-tolerance
- Recoverability
  - note that this is distinguished from 'security' - see above
- Reliability compliance

  – complies with standards relating to reliability

Maturity – in general you would expect software that has been in operation for a relatively long time and had many users to be more reliable than brand-new software which has only a few users. In the first case, one would hope that most of the bugs had been found by the users and had been fixed by the developers.

Sub-characteristics of Usability

- Understandability
  - easy to understand?
- Learnability
  - easy to learn?

- Operability
  - easy to use?
- Attractiveness – this is a recent addition
- Usability compliance
  - compliance with relevant standards

Note that there may need to be a trade-off between 'learnability' and 'operability' – an analogy would be in learning shorthand writing: it takes a long time to do, but once you have done it you can write very quickly indeed.

'Attractiveness' is a new addition – it is important where the use of software is optional – if users don't like the software then they won't use it.

Sub-characteristics of Efficiency

- Time behaviour
  - e.g. response time
- Resource utilization
  - e.g. memory usage
- Efficiency compliance
  - compliance with relevant standards

Sub-characteristics of Maintainability

- "Analysability"
  - ease with which the cause of a failure can be found
- Changeability
  - how easy is software to change?
- Stability
  - low risk of modification having unexpected effects
- "Testability"
- Maintainability conformance

Testability can be particularly important with embedded software, such as that used to control aircraft. Software that simulates the actions of the aircraft in terms of inputs to the control system etc would need to be built.

Sub-characteristics of portability

- Adaptability
- "Installability"
- Co-existence
  - Capability of co-existing with other independent software products
- "Replaceability"
  - factors giving 'upwards' compatibility - 'downwards' compatibility is excluded
- Portability conformance
  - Adherence to standards that support portability

Replaceability - A new version of a software product should be able to deal correctly with all the inputs that the previous versions could deal with. However there could be new features that need new inputs that the old system would not be able to deal with.

Using ISO 9126 quality standards (development mode)

- Judge the importance of each quality factor for the application
  - for example, safety critical systems - *reliability* very important
  - real-time systems - *efficiency* important
- Select relevant external measurements within ISO 9126 framework for these qualities, for example
  - mean-time between failures for reliability
  - response-time for efficiency

Using ISO 9126 quality standards

- Map measurement onto ratings scale to show degree of user satisfaction – for example response time

response (secs)	rating
<2	Exceeds requirement
2-5	Target range
6-10	Minimally acceptable
>10	Unacceptable

- Identify the relevant internal measurements and the intermediate products in which they would appear
  - For example, at software design stage the estimated execution time for a transaction could be calculated

Using ISO9126 approach for application software selection

- Rather than map engineering measurement to qualitative rating, map it to a score
- Rate the importance of each quality in the range 1-5
- Multiply quality and importance scores

Response (secs)	Quality score
<2	5
2-3	4
4-5	3
6-7	2
8-9	1
>9	0

Weighted quality scores

		Product A		Product B	
Product quality	Importance rating (a)	Quality score (b)	Weighted score (a x b)	Quality score (c)	Weighted score (a x c)
usability	3	1	3	3	9
efficiency	4	2	8	2	8
maintain-ability	2	3	6	1	2
Overall totals			17		19

This suggests that Product B is slightly more likely to meet the users' quality needs.

How do we achieve product quality?

- The problem: quality attributes tend to *retrospectively* measurable
- Need to be able to examine processes by which product is created beforehand
- The production process is a network of sub-processes
  - Output from one process forms the input to the next
  - Errors can enter the process at any stage

Correction of errors

- Errors are more expensive to correct at later stages
  - need to rework more stages
  - later stages are more detailed and less able to absorb change
- Barry Boehm
  - Error typically 10 times more expensive to correct at coding stage than at requirements stage
  - 100 times more expensive at maintenance stage

The additional time needed to change products later is partly because more products in the project life cycle have to change e.g. the specification, design and code, rather than just the specification. Also the later products of the project tend to be more detailed and therefore more complicated.

For each activity, *define*:

- Entry requirements
  - these have to be in place before an activity can be started
  - example: 'a comprehensive set of test data and expected results be prepared and independently reviewed against the system requirement before program testing can commence'

We are moving into process quality now. However, the entry requirements could well relate to the quality of the products of other processes that this process will need to use.

- Implementation requirements
  - these define how the process is to be conducted
  - example 'whenever an error is found and corrected, *all* test runs must be completed, including those previously successfully passed'
- Exit requirements
  - an activity will not be completed until these requirements have been met
  - example: 'the testing phase is finished only when all tests have been run in succession with no outstanding errors'

The exit requirements, however, could well relate to characteristics of the products created by the process in question.

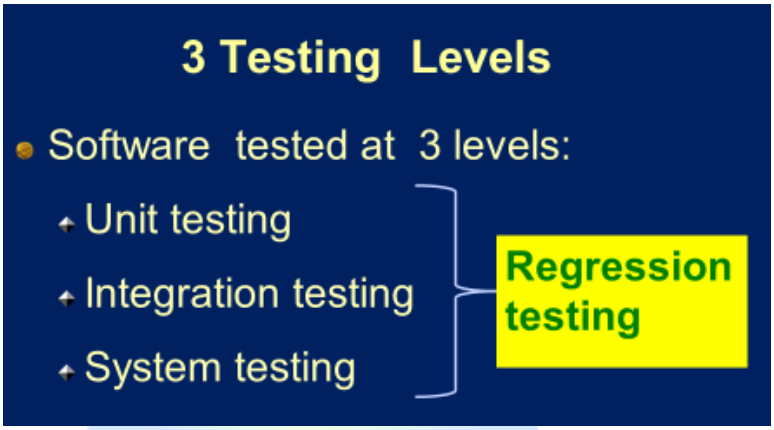
Techniques to Enhance Product Quality

- Increasing visibility
  - 'egoless programming'. Weinberg encouraged the simple practice of programmers looking at each other's code.
- Procedural structure
  - Over the years there has been the growth of
  - Methodologies

- Checking intermediate stages
  - checking the correctness of work at various stages of development

Testing

- Objective of testing is to expose as many bugs as possible.
- How is testing done?
  - Input test data to the program.
  - Observe the output: Check if the program behaves as expected



Test Levels

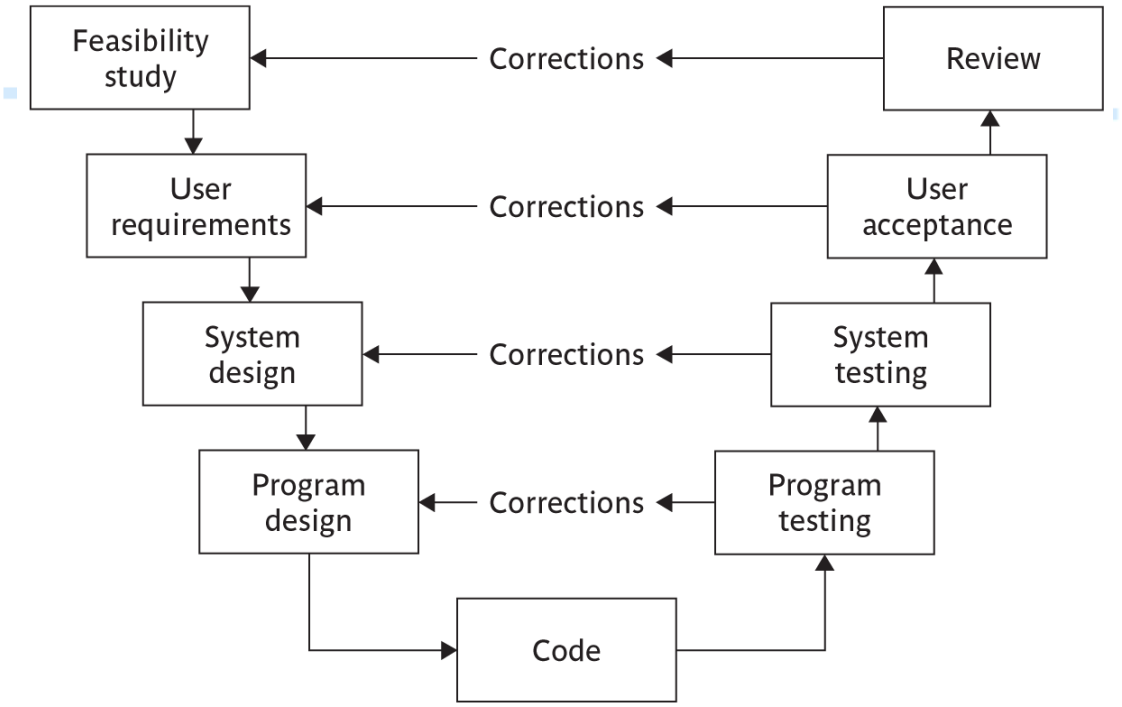
- Unit testing
  - Test each module (unit, or component) independently
  - Mostly done by developers of the modules
- Integration and system testing
  - Test the system as a whole
  - Often done by separate testing or QA team
- Acceptance testing
  - Validation of system functions by the customer

Verification versus Validation

- Verification is the process of determining:
  - Whether output of one phase of development conforms to its previous phase.
- Validation is the process of determining:
  - Whether a fully developed system conforms to its SRS document.
- Verification is concerned with phase containment of errors:
  - Whereas the aim of validation is that the final product be error free.

Testing: the V-process model

- This shown diagrammatically in the next slide
- It is an extension of the waterfall approach
- For each development stage there is a testing stage
- The testing associated with different stages serves different purposes e.g. system testing tests that components work together correctly, user acceptance testing that users can use system to carry out their work



Black box versus glass box test

- **Glass box testing**
  - The tester is aware of the internal structure of the code; can test each path; can assess percentage test coverage of the tests e.g. proportion of code that has been executed
- **Black box testing**
  - The tester is not aware of internal structure; concerned with degree to which it meets user requirements

Levels of testing

- Unit testing
- Integration testing
- System testing

Testing activities

- *Test planning*
- *Test suite design*
- *Test case execution and result checking*
- *Test reporting:*
- *Debugging:*
- *Error correction:*
- *Defect retesting*
- *Regression testing*
- *Test closure:*

Test plans

- Specify test environment
  - In many cases, especially with software that controls equipment, a special test system will need to be set up

- Usage profile
  - failures in operational system more likely in the more heavily used components
  - Faults in less used parts can lie hidden for a long time
  - Testing heavily used components more thoroughly tends to reduce number of operational failures

Management of testing

The tester executes test cases and may as a result find discrepancies between actual results and expected results – **issues**

Issue resolution – could be:

- a mistake by tester
- a fault – needs correction
- a fault – may decide not to correct: **off-specification**
- a change – software works as specified, but specification wrong: submit to change control

Decision to stop testing

The problem: impossible to know there are no more errors in code

Need to estimate how many errors are likely to be left

Bug seeding – insert (or leave) known bugs in code

Estimate of bugs left =

(total errors found)/(seeded errors found) x (total seeded errors)

Alternative method of error estimation

- Have two independent testers, A and B
- N_1 = valid errors found by A
- N_2 = valid errors found by B
- N_{12} = number of cases where same error found by A and B
- Estimate = $(N_1 \times N_2)/ N_{12}$

- Example: A finds 30 errors, B finds 20 errors. 15 are common to A and B. How many errors are there likely to be?

Test automation

- Other than reducing human effort and time in this otherwise time and effort-intensive work,
  - Test automation also significantly improves the thoroughness of testing.
- A large number of tools are at present available both in the public domain as well as from commercial sources.

Types of Testing Tools

- Capture and playback
- Automated test script
- Random input test
- Model-based test

Software reliability

- The reliability of a software product essentially denotes its *trustworthiness* or *dependability*.
  - Usually keeps on improving with time during the testing and operational phases as defects are identified and repaired.
- A reliability growth model (RGM) models how the reliability of a software product improves as failures are reported and bugs are corrected.
  - An RGM can be used to determine when during the testing phase a given reliability level will be attained, so that testing can be stopped

Software process quality

Product and Process Quality

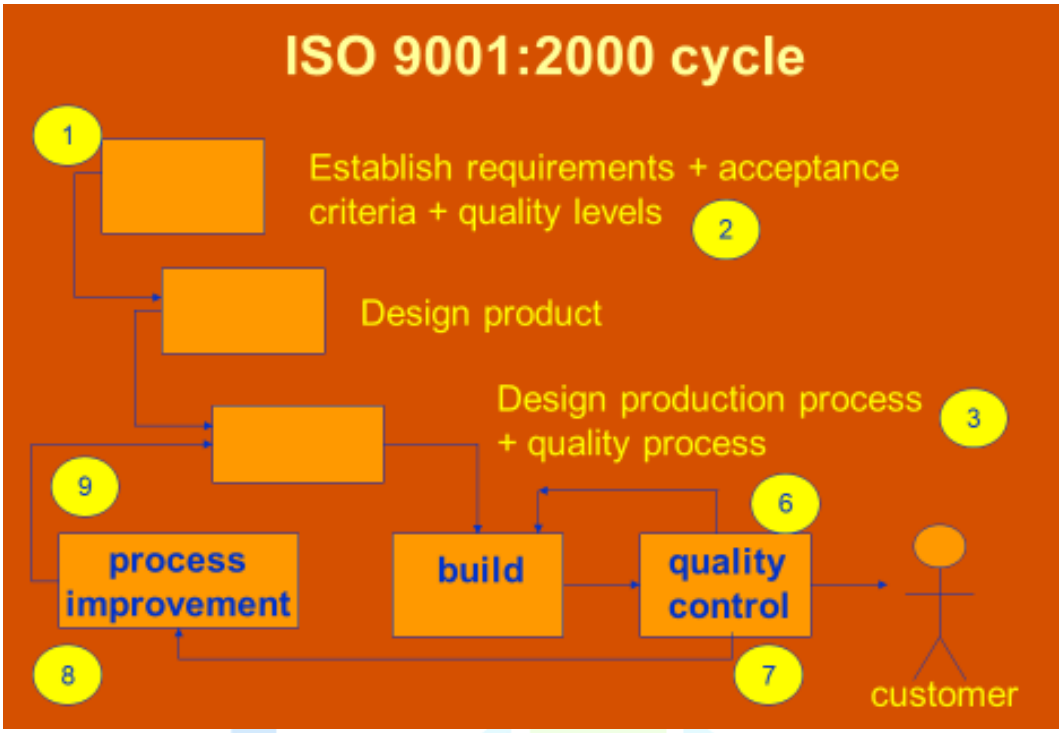
- A good process is usually required to produce a good product.
- For manufactured goods, process is the principal quality determinant.
- For design-based activity, other factors are also involved:
  - For example, the capabilities of the designers.

BS EN ISO 9001:2000 and quality management systems

- ISO 9001 is one of a family of standards that specify the characteristics of a good quality management system (QMS)
- Can be applied to the creation of any type of product or service, not just IT and software
- Does NOT set universal product/service standards
- DOES specify the way in which standards are established and monitored

ISO 9001:2000 principles

1. Understanding the requirements of the customer
2. Leadership to provide unity of purpose and direction to achieve quality
3. Involvement of staff at all levels
4. Focus on the individual which create intermediate and deliverable products and services
5. Focus on interrelation of processes that deliver products and services
6. Continuous process improvement
7. Decision-making based on factual evidence
8. Mutually beneficial relationships with suppliers



These principles are applied through cycles which involve the following activities (numbers in yellow circles map to numbered items below):

1. determining the needs and expectations of the customer;

2. establishing a *quality policy*, that is, a framework which allows the organization's objectives in relation to quality to be defined;

3. design of the *processes* which will create the products (or deliver the services) which will have the qualities implied in the organization's quality objectives;

4. allocation of the responsibilities for meeting these requirements for each element of each process;

5. ensuring resources are available to execute these processes properly;

6. design of methods for measuring the effectiveness and efficiency of each process in contributing to the organization's quality objectives;

7. gathering of measurements;

8. . identification of any discrepancies between the actual measurements and the target values;

9. analysis and elimination of the causes of discrepancies.

The need to improve

- can everything be improved at one?

  no, must tackle the most important things first

- 'poor companies are poor at changing'

  some later improvements build on earlier ones

- but there are problems
  - *improvement takes up time and money*
  - *'improvement' may simply be more bureaucracy!*

Capability maturity model (CMM)

- Created by Software Engineering Institute, Carnegie Mellon University
- CMM developed by SEI for US government to help procurement
- The rationale was:
  - Include likely contractor performance as a factor in contract awards.
- Watts S. Humphrey 'Managing the software process' Addison Wesley
- Assessment is by questionnaire and interview

Over the years different version of CMM were devised for different environments, but more recently there has been an attempt to reintegrate these disparate models into one generic, yet flexible, model called CMM Integration (CMMI)

Capability maturity model 2

- Different versions have been developed for different environments e.g. software engineering
- New version CMMI tries to set up a generic model which can be used for different environments

SEI Capability Maturity Model

- Can be used in two ways:
  - Capability evaluation
  - Software process assessment.

Capability Evaluation

- Provides a way to assess the software process capability of an organization:
  - Helps in selecting a contractor
  - Indicates the likely contractor performance.

Software Process Assessment

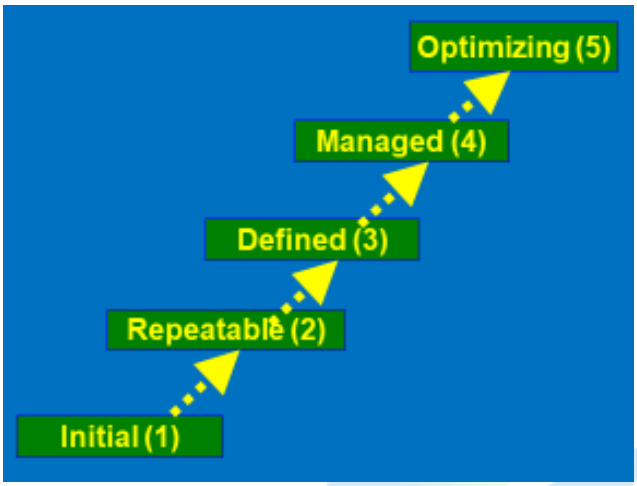
- Used by an organization to assess its current process:
  - Suggests ways to improve the process capability.
- This type of assessment is for purely internal use.

What is CMM?

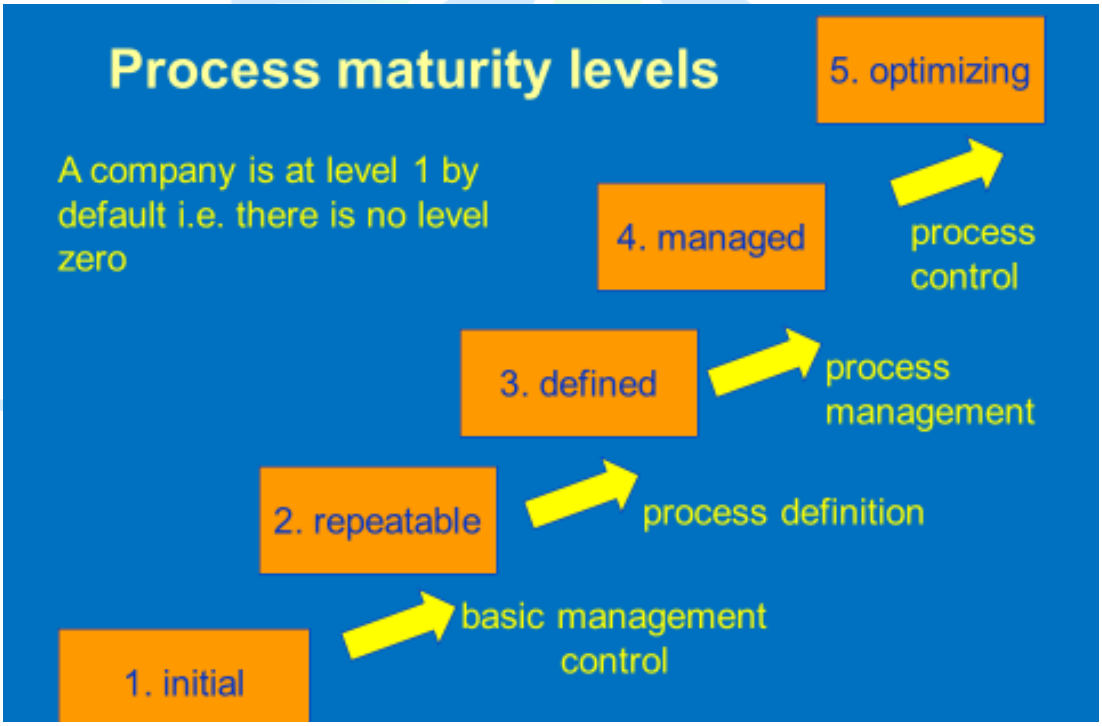
- Describes an evolutionary improvement path for software organizations from an ad hoc immature process :
  - To a mature, disciplined one.
- Provides guidance on:
  - How to control the process

- How to evolve the process

CMM Maturity Levels



- Five maturity levels:
- Stages are ordered so that improvements at one stage provide foundations for the next.
- Based on the pioneering work of Philip Crosby.



CMM Level 1 (Initial)

- Organization operates Without any formalized process or project plans
- An organization at this level is characterized by Ad hoc and chaotic activities.
- Software development processes are not defined,
- Different developers follow their own process

- The success of projects depend on individual efforts and heroics.

Level 2 (Repeatable)

- Basic project management practices are followed
  - Size and cost estimation techniques:
  - Function point analysis, COCOMO, etc.
  - Tracking cost, schedule, and functionality.
- Development process is ad hoc:
  - Not formally defined
  - Also not documented.

Level 3 (Defined)

- All management and development activities:
  - Defined and documented.
  - Common organization-wide understanding of activities, roles, and responsibilities.
- The process though defined:
  - Process and product qualities are not measured.

Level 4 (Managed)

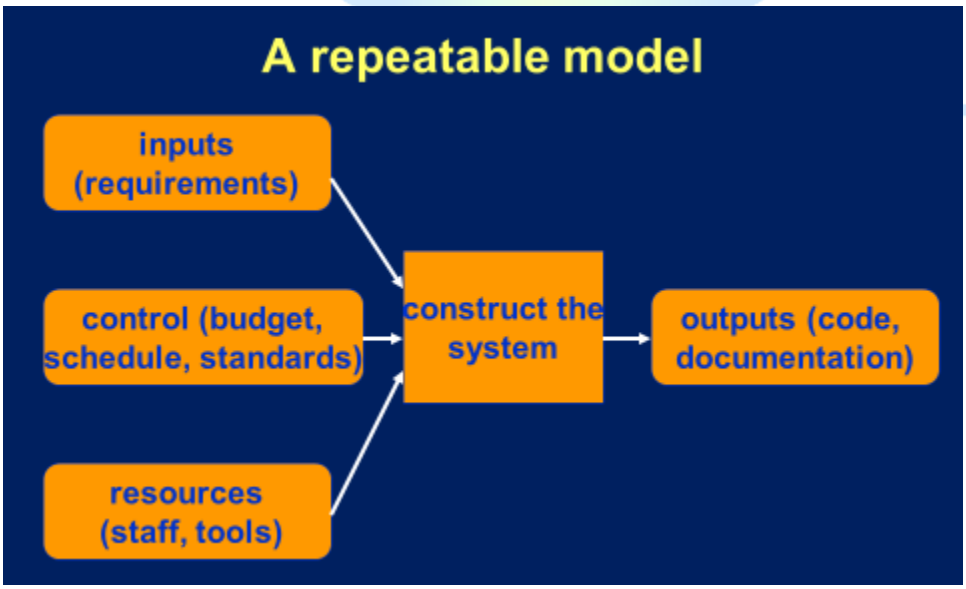
- Quantitative quality goals for products are set.
- Software process and product quality are measured:
  - The measured values are used to control the product quality.
- Results of measurement used to evaluate project performance:
  - Rather than improve process.
- Detailed measures of the software process and product quality are collected.
- Both the software process and products are quantitatively understood and controlled.

Level 5 (Optimizing)

- Statistics collected from process and product measurements are analyzed:
  - Continuous process improvement based on the measurements.
- Known types of defects are prevented from recurring by tuning the process
- Lessons learned from specific projects incorporated into the process

Key process areas

- The KPAs of a level indicate the areas that an organization at the lower maturity level needs to focus to reach this level.
- KPAs provide a way for an organization to gradually improve its quality of over several stages.
- KPAs for each stage has been carefully designed such that one stage enhances the capability already built up.
  - Trying to focus on some higher level KPAs without achieving the lower level KPAs would be counterproductive.



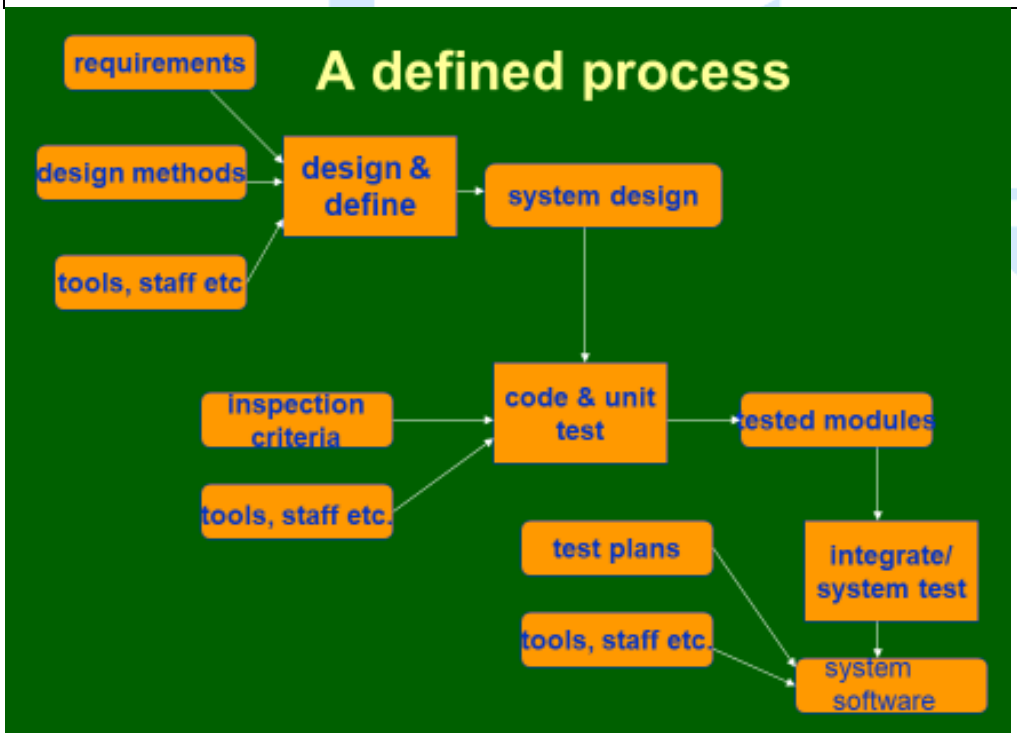
The idea is that the process is under 'statistical control'. Essentially this means that you are collecting data about the performance of the project. This allows you to plan future projects more accurately because, for example, you know approximately how long it will take you to do a new project because you have data on the productivity rates experienced by project teams doing similar work in the past.

Repeatable model KPAs

To move to this level concentrate on:

- Configuration management
- Quality assurance
- Sub-contract management
- Project planning
- Project tracking and oversight
- Measurement and analysis

- KPAs are Key Process areas.
- **Configuration management** – this is the creation and maintenance of a database of the products being created by the project. This enables the project team, among other things, to ensure that when a change is made to one product, e.g. a design document, other related documents, e.g. test cases and expected results are updated.
- **Quality assurance –** this involves setting up a group whose job it is to check that people are following the laid-down quality procedures



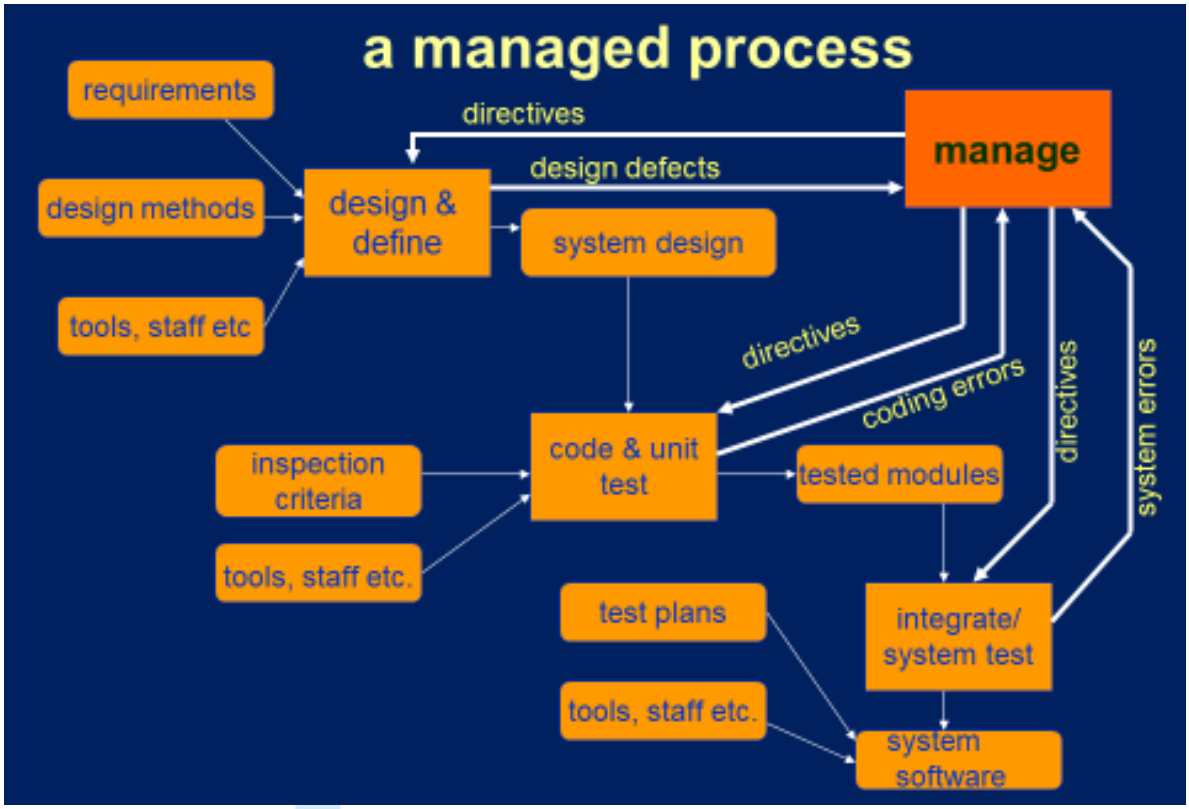
Management at the previous level tends to see the project as a whole. At this level an attempt is made to breakdown the project into a set of component sub-activities and to make the progress and effectiveness of these sub-activities visible to the wider world. Sub-activities are broken down into even lower level activities until you get to activities carried out by individuals or small teams.

Repeatable to defined KPAs

Concentrate on

- Requirements development and technical solution
- Verification and validation
- Product integration
- Risk management
- Organizational training
- Organizational process focus (function)
- Decision analysis and resolution
- Process definition
- Integrated project management

1. **Requirements analysis -** multi-stakeholder requirements evolution
2. **Technical solution –** evolutionary design and quality engineering
3. **Product integration –** continuous integration, interface control, change management
4. **Verification:** assessment to ensure that the product is produced correctly
5. **Validation:** assessment to ensure that the right product is created
6. **Risk management**
7. **Organizational training**
8. **Organizational process focus –** establishing an organizational framework for process definition
9. **Decision analysis and resolution –** systematic alternative assessment
10. **Process definition** treatment of process as a persistent, evolving asset of the organization
11. **Integrated project management** Methods for unifying the various teams and stakeholders within a project.

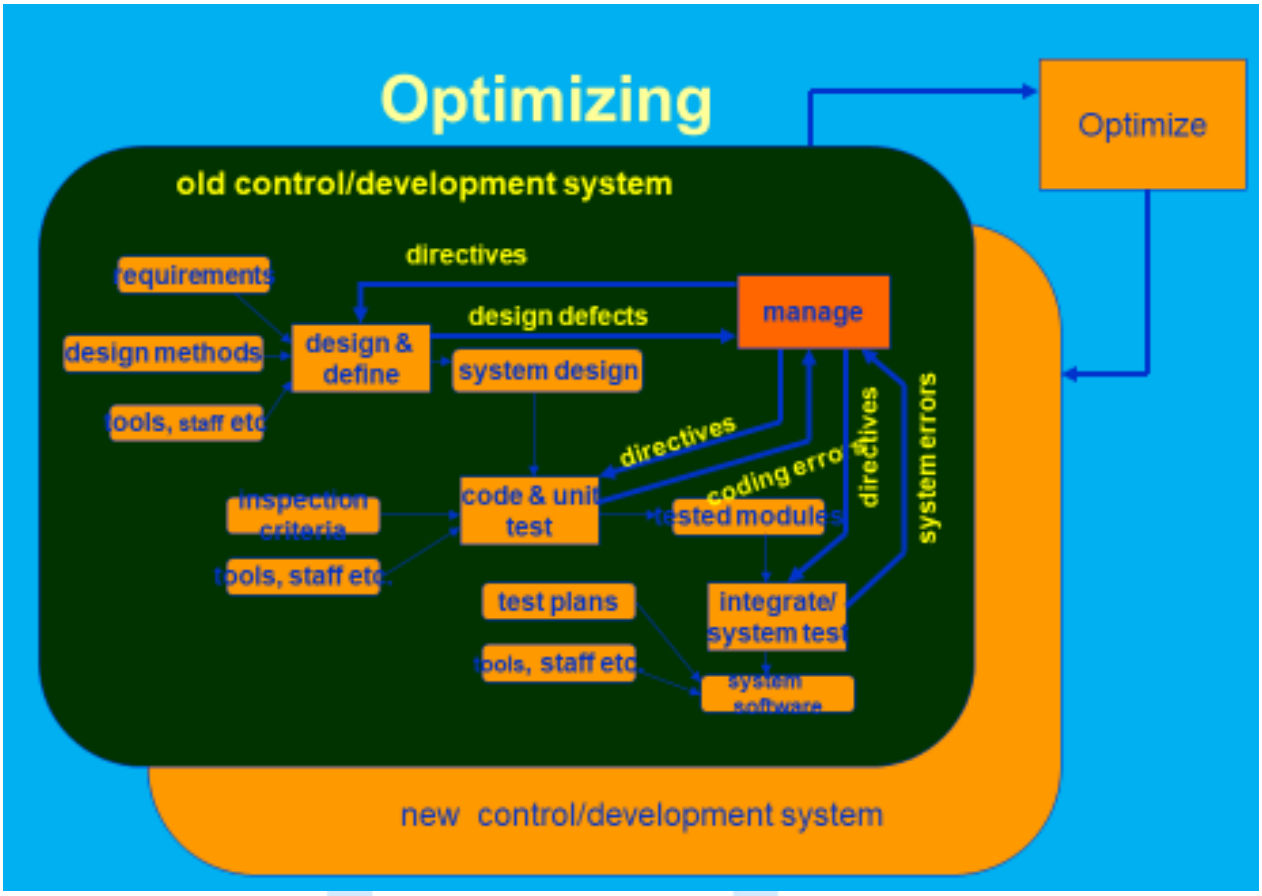


Now that you have identified individual sub-processes and are collecting data about them, you can carry management interventions as problems start to emerge and thus stop the problems evolving into major set-backs.

Defined to managed KPAs

Concentrate on:

- Organizational process performance
- Quantitative project management



You are now in a position not only to deal with problems and quality short-falls as they emerge in a project; you can look at the processes you have designed and identify defects in them that lead to deficient products. Actions can be taken to remove the source of deficient products, not just the deficient products themselves.

Managing to optimizing: KPAs

Concentrate on:

- Causal analysis and resolution
- Organizational innovation and deployment

CMMI (Capability Maturity Model Integration)

- CMMI is the successor of the Capability Maturity Model (CMM).
- After CMMI was first released in 1990:
  - It became popular in many other domains
  - Human Resource Management (HRM).: people management (PCMM)

- software acquisition (SA-CMM)
- systems engineering (SE-CMM)

Some questions about CMMI

- Suitable only for large organizations?
  - e.g. need for special quality assurance and process improvement groups
- Defining processes may not be easy with new technology
  - how can we plan when we've not used the development method before?
- Higher CMM levels easier with maintenance environments?
- Can you jump levels?

ISO/IEC 15504 IT process assessment

- To provide guidance on the assessment of software development processes
- **Process Reference Model**: Needs a defined set of processes that represent good practice to be the benchmark
- **ISO 12207** is the default reference model
- Could use others in specific environments

It can be argued that CMMI can be seen as a particular implementation of ISO 15504. There is however some controversy over this.

ISO 15504 performance attributes

CMMI level	ISO 15504
	0. incomplete
initial	1.1.process performance – achieves defined outcome
repeatable	2.1 process management – it is planned and monitored
	2.2 work product management – control of work products

CMMI	ISO 15504
Defined	3.1. Process definition
	3.2. Process deployment
Managed	4.1. Process measurement
	4.2. Process control
Optimizing	5.1. Process innovation
	5.2. Process optimization

At Level 3 ISO 15504 distinguishes between the definition of all the processes in the development life cycle and the process or educating and managing staff to adhere to the 'best practice' processes.

Similarly at Level 4 it implies that full control of the process cannot be in place until the process is being properly and consistently measured

Level 5 Process innovation – as a result of the data collected in 4.1, opportunities for process improvements are identified Process optimization – the opportunities for process improvement are properly evaluated and where appropriate are effectively implemented.

Process Reference Model

- A defined standard approach to development
- Reflects recognized good practice
- A benchmark against which the processes to be assessed can be judged
- ISO 12207 is the default model

The Process Reference model is the benchmark against which a set of processes is to be assessed. For example, ISO 12207 specifies the features of an acceptable software requirements document. Among the things that this must produce are included '*Functional and capability specifications, including performance, physical characteristics, and environmental conditions under which the software item is to perform*'. My interpretation here is that this includes ensuring the software will execute correctly when given any of the inputs that might be expected in the environment in which it will operate.

ISO 15504 performance indicators

This is just an example of how indicators for each level *might* be identified

1.Performance

Descriptions of maximum and minimum expected input values exist

2.1 Performance management

A plan of how expected input variable ranges are to be obtained exists which is up to date

2.2 Work product management

There are minutes of a meeting where the input requirements document was reviewed and corrections were mandated

3.1 Process definition

A written procedure for input requirements gathering exists

3.2 Process deployment

A control document exists that is signed as each part of the procedure is completed

4.1. Process measurement

Collected measurement data can be collected e.g. number of changes resulting from review

4.2. Process control

Memos relating to management actions taken in the light of the above

5.1 Process innovation

Existence of some kind of 'lessons learnt' report at the end of project

5.2. Process optimization

Existence of documents assessing the feasibility of suggested process improvements and which show consultation with relevant stakeholders

Techniques to improve quality –Inspections

- When a piece of work is completed, copies are distributed to co-workers
- Time is spent individually going through the work noting defects
- A meeting is held where the work is then discussed
- A list of defects requiring re-work is produced

Inspections are a simple but very effective methods of removing errors from documents. Software code could be seen as just another type of document in this context.

Inspections - advantages of approach

- An effective way of removing superficial errors from a piece of software
- Motivates the software developer to produce better structured and self-descriptive code
- Spreads good programming practice
- enhances team-spirit
- *The main problem* maintaining the commitment of participants

'Clean-room' software development

Ideas associated with Harlan Mills at IBM

- Three separate teams:

1. Specification team – documents user requirements and usage profiles (how much use each function will have)
2. Development team – develops code but does not test it. Uses mathematical verification techniques
3. Certification team – tests code. Statistical model used to decide when to stop

- The key idea here is to avoid developers building perhaps crudely coded software components which they turn into working system by a process of throwing tests at it and making changes. Claimed that it is impossible to produce fault-free code this way – rather they should used structured approaches and formal mathematical notations and demonstrate software will work by logical verification.
- Note that usage profiles mentioned on the slide are explored further in the section on testing.

Formal methods

- Use of mathematical notations such as VDM and Z to produce unambiguous specifications
- Can prove correctness of software mathematically (cf. geometric proofs of Pythagoras' theorem)
- Newer approach use Object Constraint Language (OCL) to add detail to UML models
- Aspiration is to be able to generate applications directly from UML+OCL without manual coding – Model Driven Architectures (MDA)

Model Driven Architecture is the complete opposite to extreme programming!

Quality plans

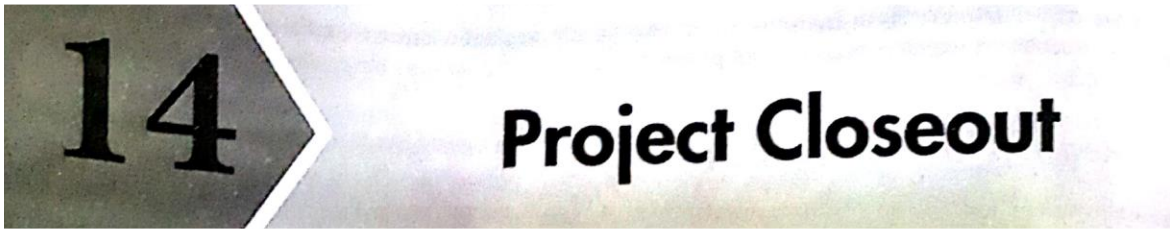
- quality standards and procedures should be documented in an organization's *quality manual*
- for each separate project, the quality needs should be assessed
- select the level of quality assurance needed for the project and document in a *quality plan*

Typical contents of a quality plan

- scope of plan
- references to other documents
- quality management, including organization, tasks, and responsibilities
- documentation to be produced
- standards, practices and conventions
- reviews and audits

more contents of a quality plan

- testing
- problem reporting and corrective action
- tools, techniques, and methodologies
- code, media and supplier control
- records collection, maintenance and retention
- training
- risk management



Project Closeout

- Every project must come to an end sometime or other
  - It is the responsibility of the project manager to decide the appropriate time to close a project.
- Project closeout activities:
  - administrative closure
  - Contract closure

- Increasing criticality of software – e.g .software is increasingly being used in systems that can threaten or support human life and well-being
- The intangibility of software – it is difficult for observers to judge the quality of software development, especially during its early stages
- Project control concerns:
- The products of one sub-process in the development process are the inputs to subsequent sub-processes, thus
- errors accumulate with each stage e.g. at the design stage, the specification errors are incorporated into the design, and at the coding stage specification and design errors are incorporated into the software
- errors become more expensive to remove the later they are found
- it is difficult to control the error removal process (e.g. testing)
- See Section 13.3

Types of Project Closure

- Two main types:
  - Normal termination: All the project goals have been successfully accomplished.
  - Premature termination: the project is unlikely to achieve its stated objectives --- This is the case for about a third of all projects

Premature Termination

- There are many reasons as to why a project may have to be prematurely terminated:
  - *Lack of resources*
  - *Changed business need of the customer*
  - *perceived benefits accruing from the project no longer remain valid*
  - *Changes to the regulatory policies*
  - *Key technologies used in the project becoming obsolete during project execution*
  - *Risks have become unacceptably high:*

Why are projects not properly closed?

- Often projects are not properly closed:
  - *Lack of interest by the project team*
  - *Underestimation of how fast know-how can get lost and how much implicit knowledge exists with the team members*
  - *Emotional factors*
  - *Indecision regarding project closure:*

Problems of Improper project closure

- *Time and cost overrun:*
  - project as a cost centre runs up expenditure
- *Locks up valuable human and other resources*
  - Redeployment of project personnel and other resources gets delayed
- *Stress on the project personnel:*

Issues associated with project termination

- Two fold:
  - Emotional

- Intellectual

- Team members may pay more attention to issues such as getting reassigned to a project of their choice and the project work can take a back seat
- Terms of contract and the list of deliverables need to be renegotiated
- Closure decision has to be effectively communicated to all stakeholders.

Project Closure Process

- Getting client acceptance
- Archiving project deliverables
- Preserving project know-how
- Performing a financial closure
- Performing postImplementation project review
- Preparing postimplementation review report
- Releasing staff

Post-implementation project review

- Conduct project survey.
- Collect objective information.
- Hold a debriefing meeting.
- Prepare post-implementation review report.
- Publish the report.

Project survey

- *Project performance*
- *Administrative performance*
- *Organizational structure*
- *Team performance*
- *Techniques of project management*

- *Risk management:*

Project Closeout Report

- Project closeout report is intended to provide a concise evaluation of the project.
- Project description: Information about the project, to give context
- What worked well
- The factors that impeded the performance of the project
- A prescription for other projects to follow