Prof.Tirup Parmar

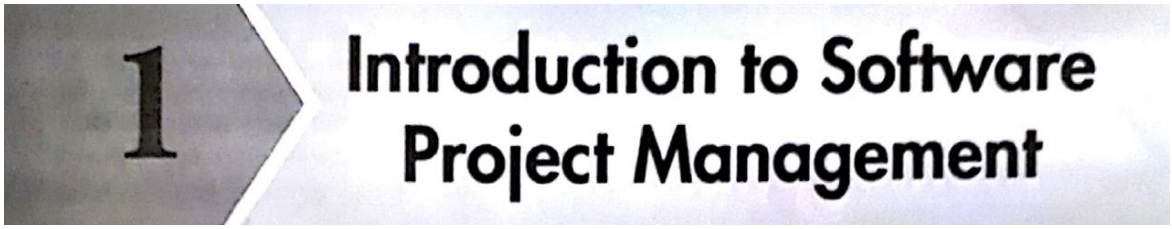
[Document subtitle]

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

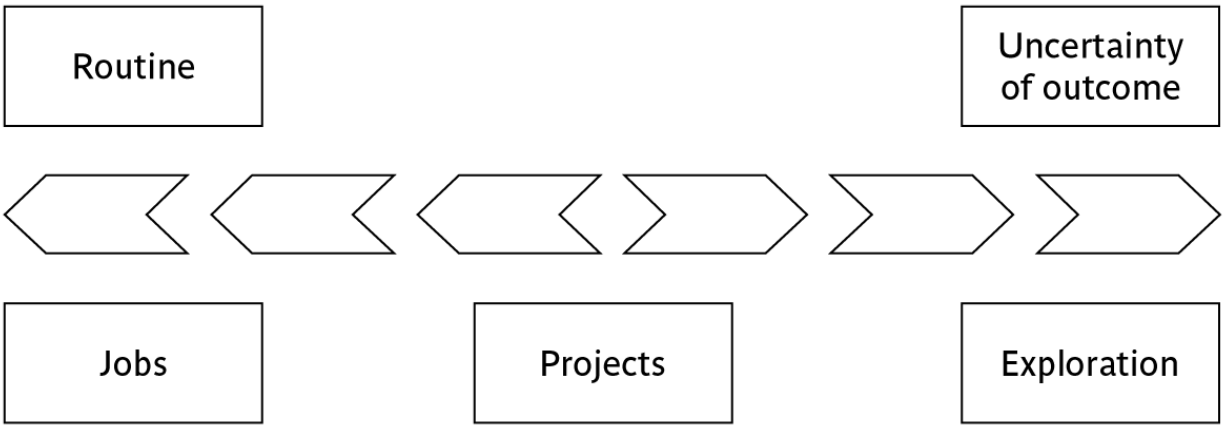
What is a Task?

- 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

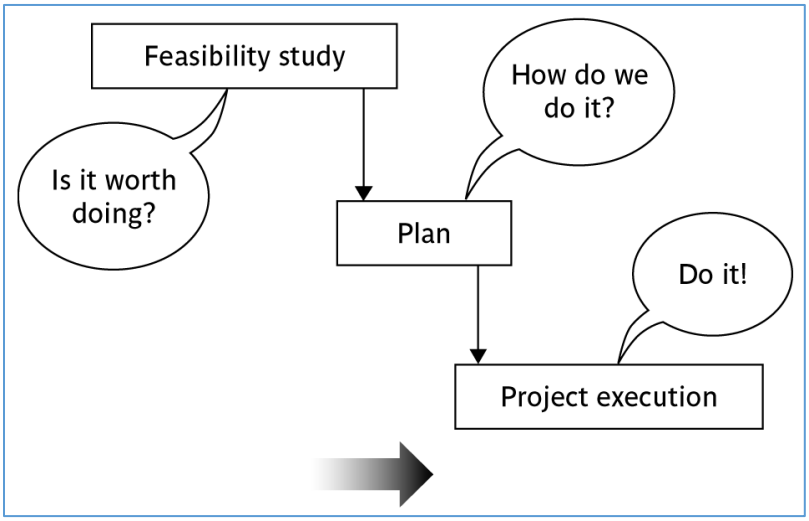
Contract management versus technical project management

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

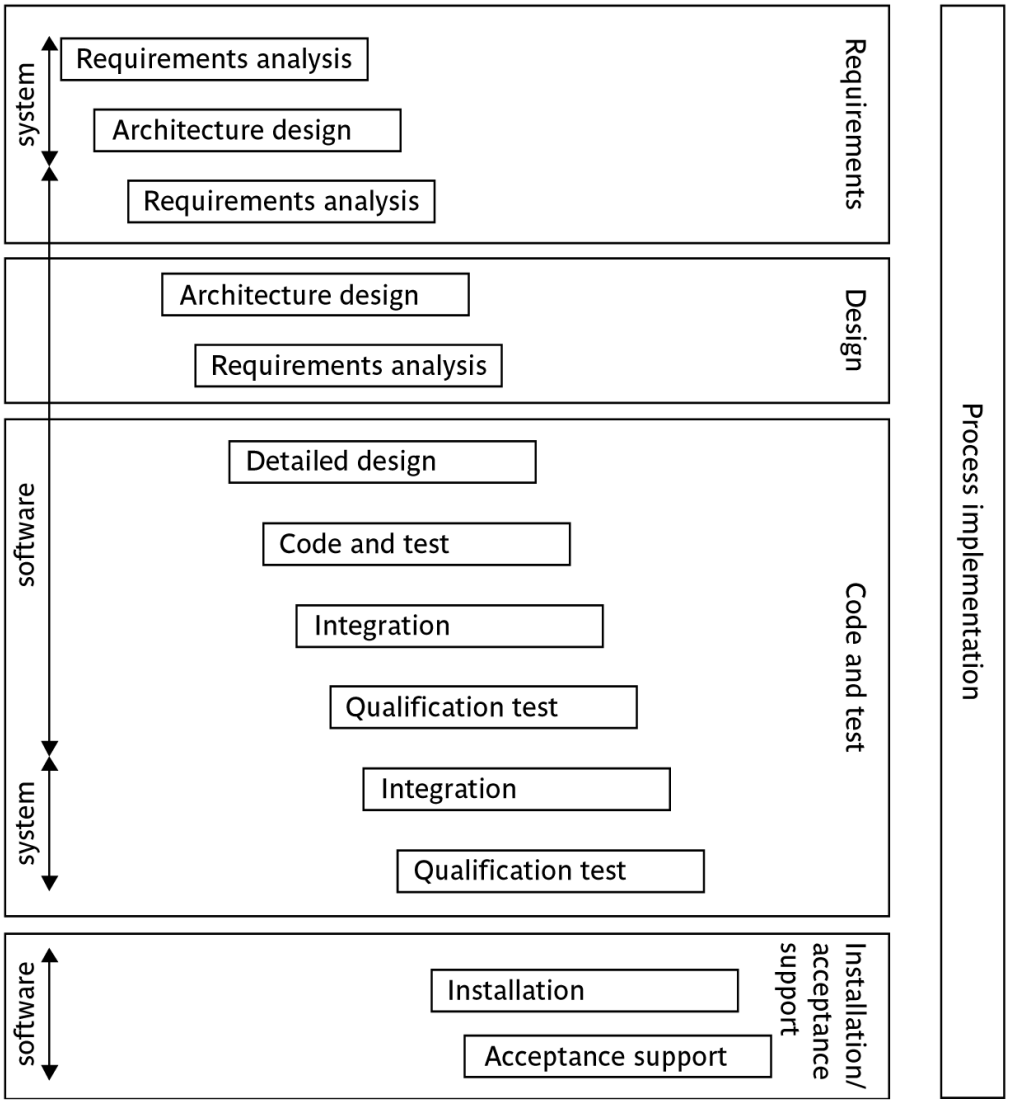
Execution:

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

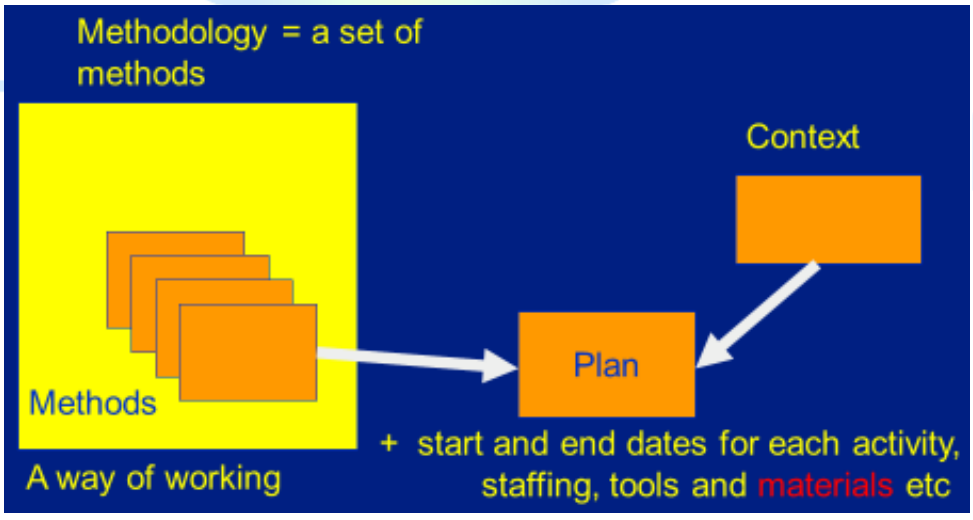
- *The software project will almost certainly be part of a larger project which has non-software elements. In a software engineering environment it could be the software will be embedded in hardware product of some kind. Thus there are system requirements for the product as a whole and software requirements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

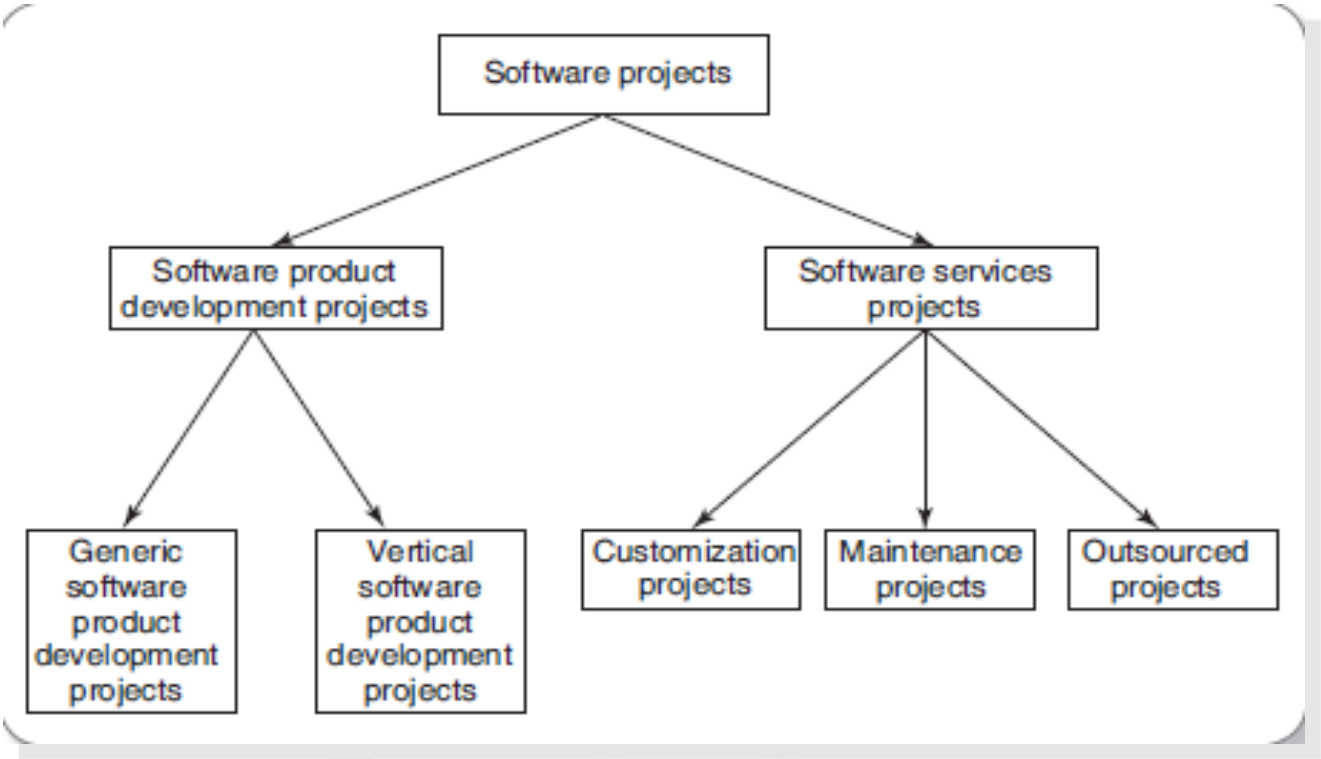


Fig. A Categorization of Software Projects

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

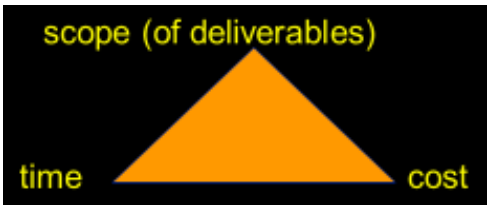
Quantifiable

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

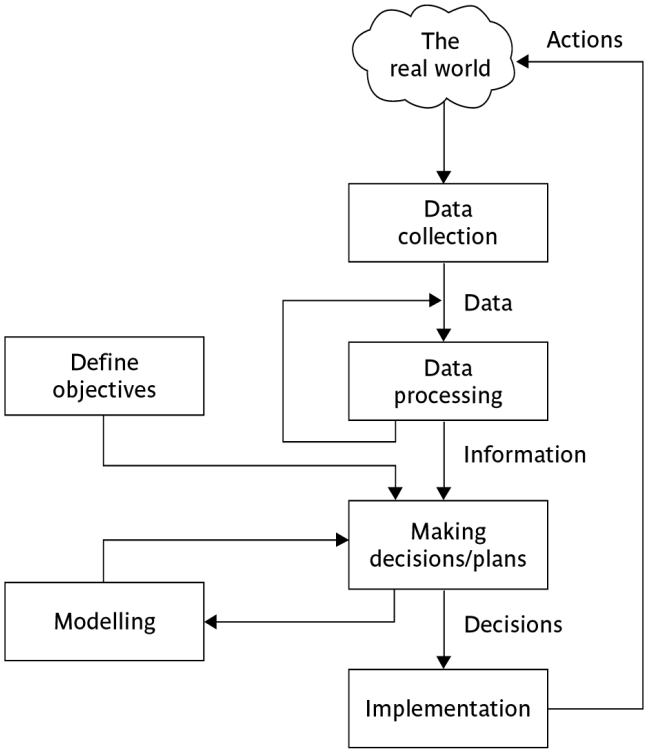
In the project initiation stage, an initial plan is made. As the project start, the project is monitored and controlled to proceed as per the plan. But, the initial plan is refined from time to time to factor in additional details and constraints about the project become available.

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

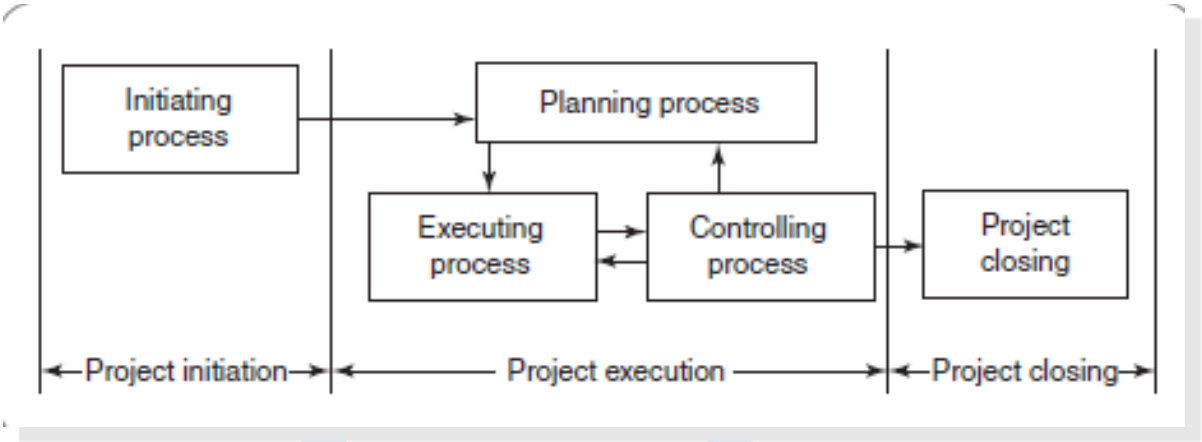
Modelling – working out the probable outcomes of various decisions

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

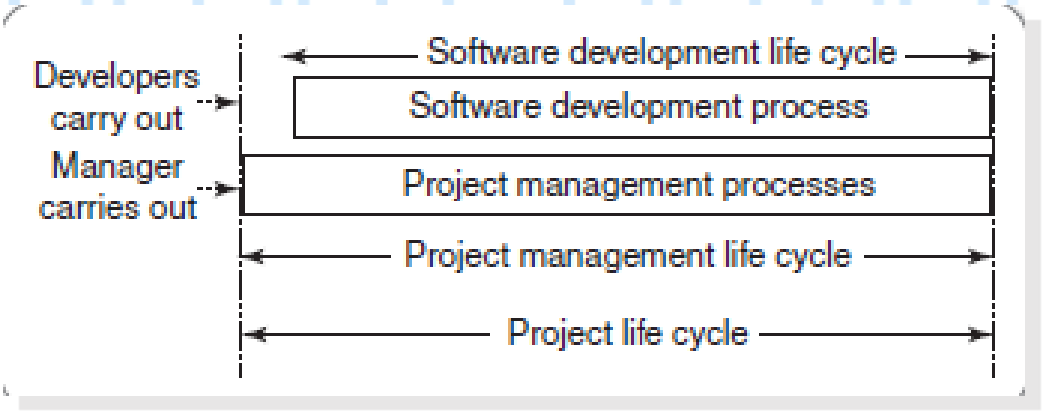
> *The authors' view is that an initially defective plan can often be remedied by good project control and management.*

Project Management Processes



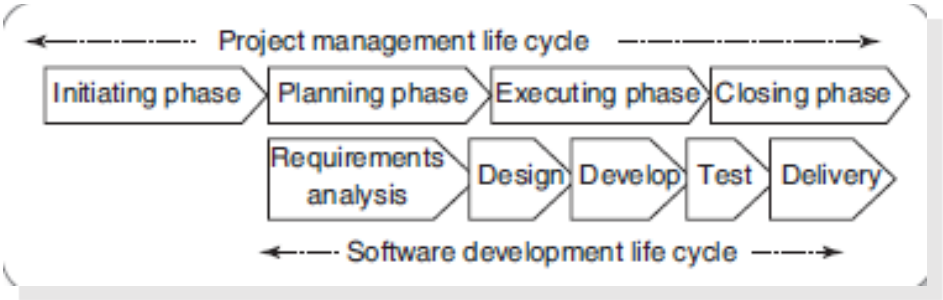
In the project initiation stage, an initial plan is made. As the project starts, the project is executed and controlled to proceed as planned. Finally, the project is closed.

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:

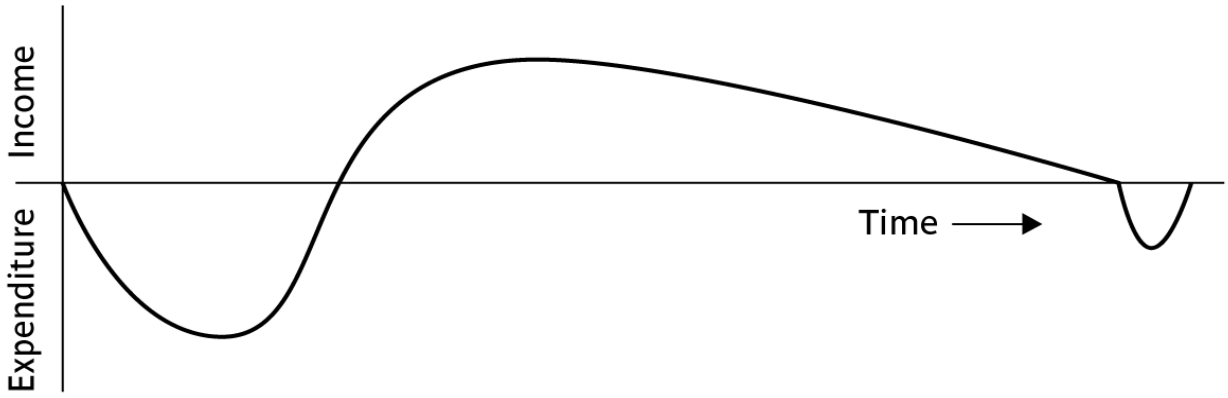
- Development costs
- Set-up
- Operational costs

- 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?

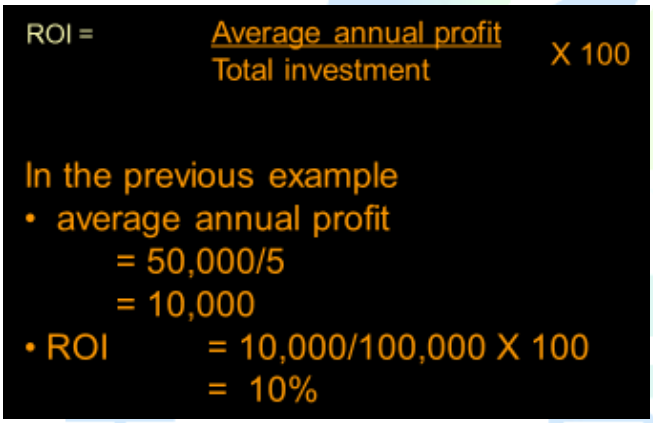
Year	Cash-flow	Accumulated
0	**-100,000**	**-100,000**
1	**10,000**	**-90,000**
2	**10,000**	**-80,000**

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

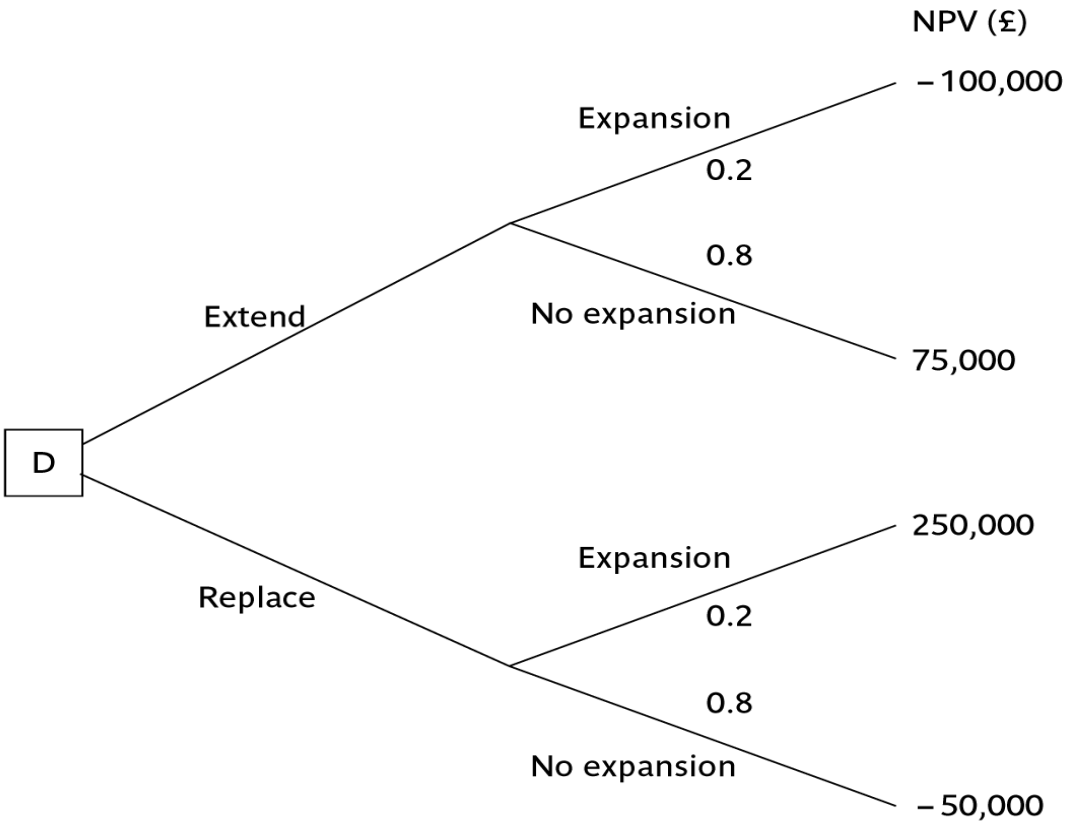
Example of a project risk matrix

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

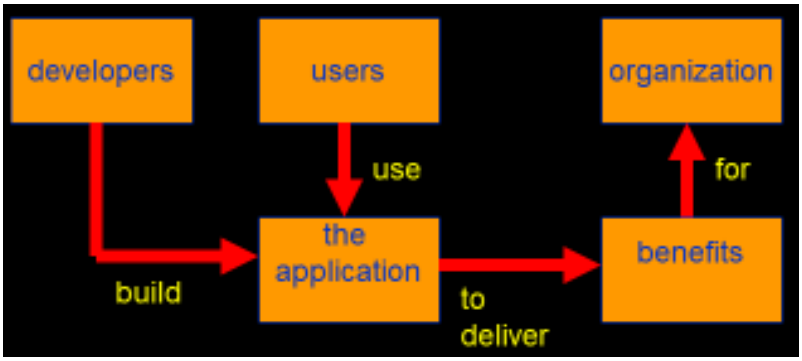
- *The material here is based on the UK government's Office of Government Commerce (OGC) approach which is described in detail in their publication Managing successful programmes.*
- *The programme director should be someone who is in a prominent position in the organization so that the seriousness and 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.*

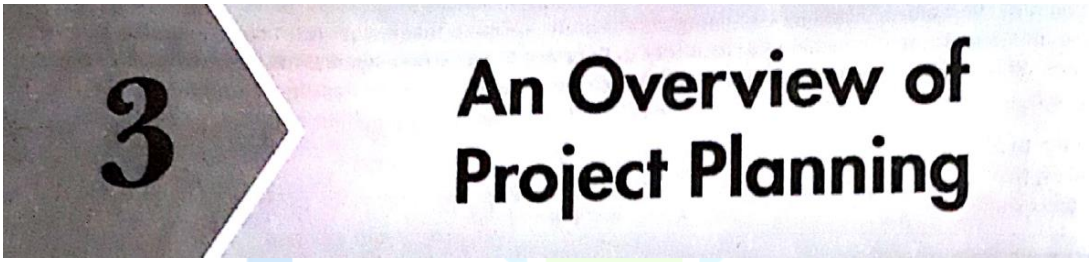
Quantifying benefits

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.

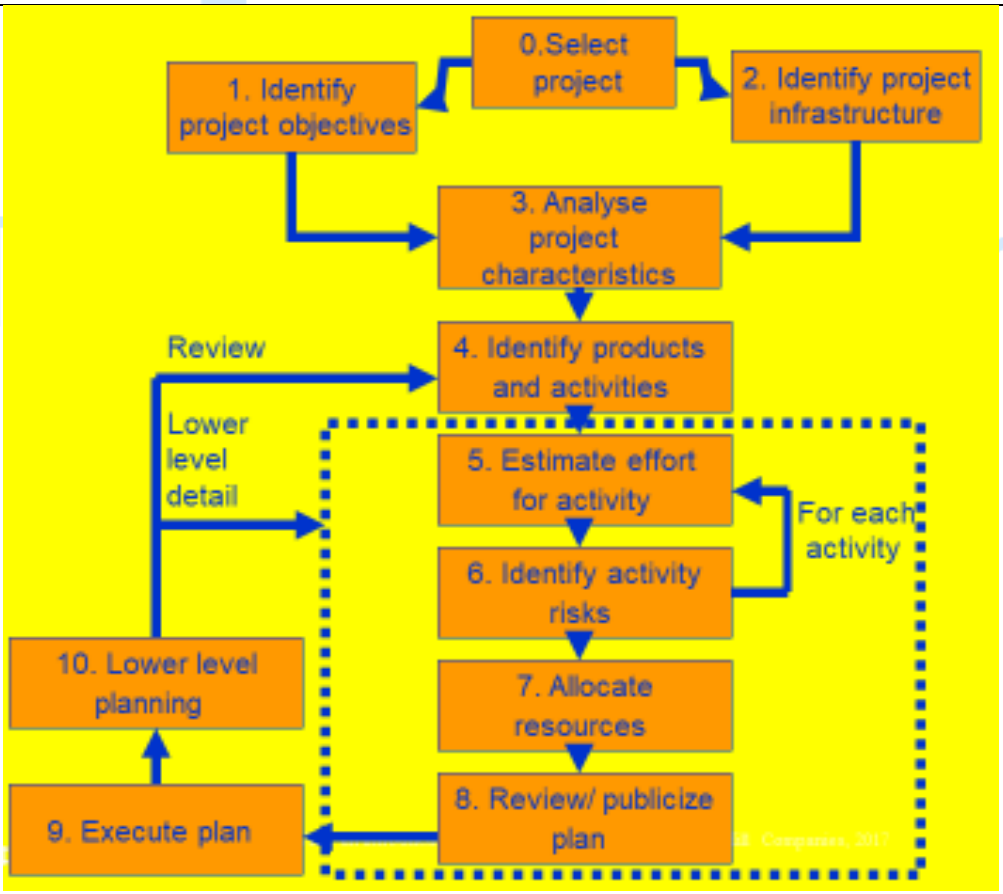
Students are perhaps most likely to come into contact with some kind of project planning (i) when they are involved in student projects, especially those carried in groups and (ii) if they are members of a project team when they are undertaking a placement year in industry. The first situation is probably

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.*

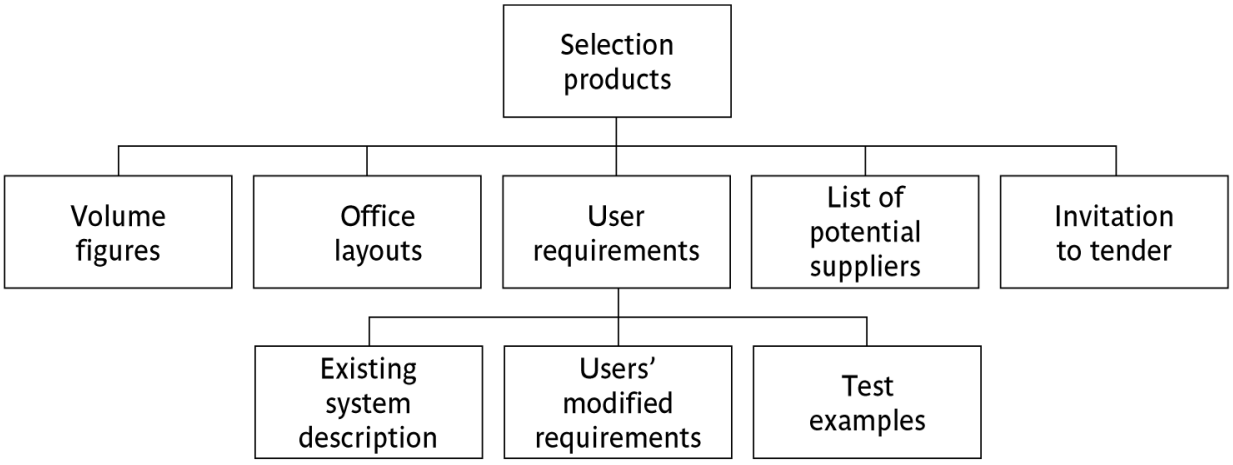
Back to the scenario

- 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

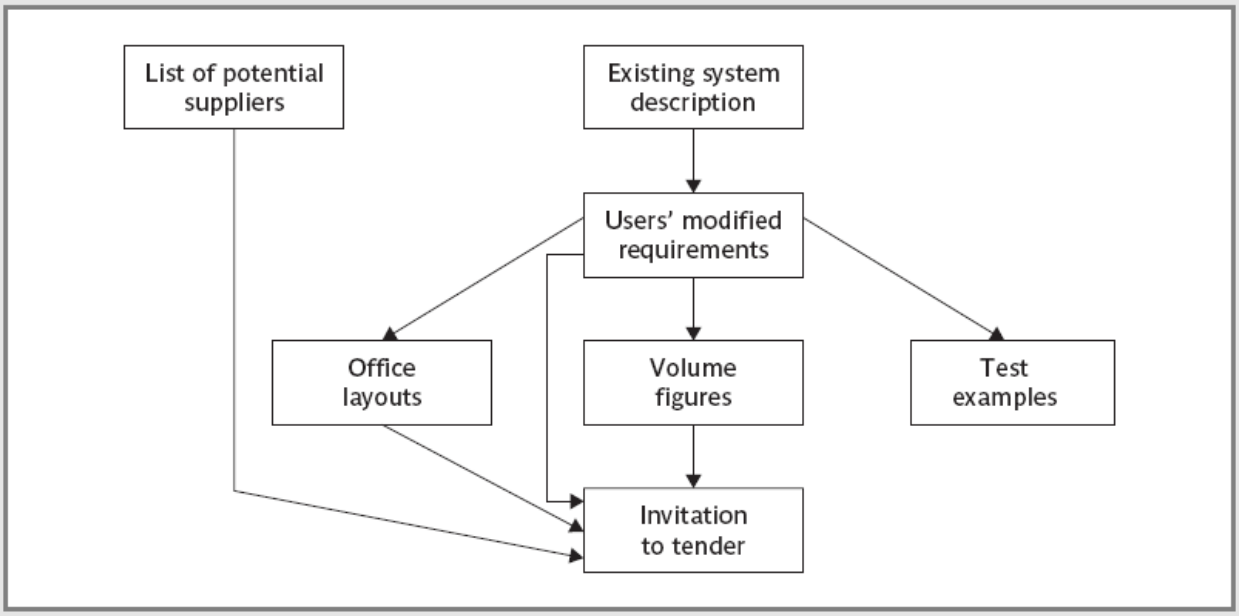


FIGURE B.1 Product Flow Diagram for the creation of an 'invitation to tender'

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

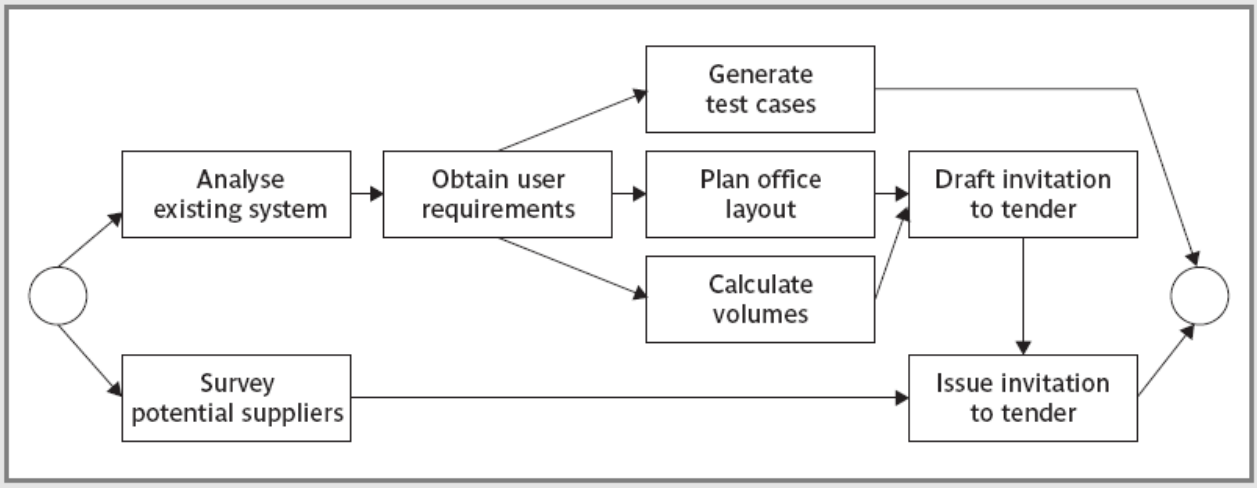
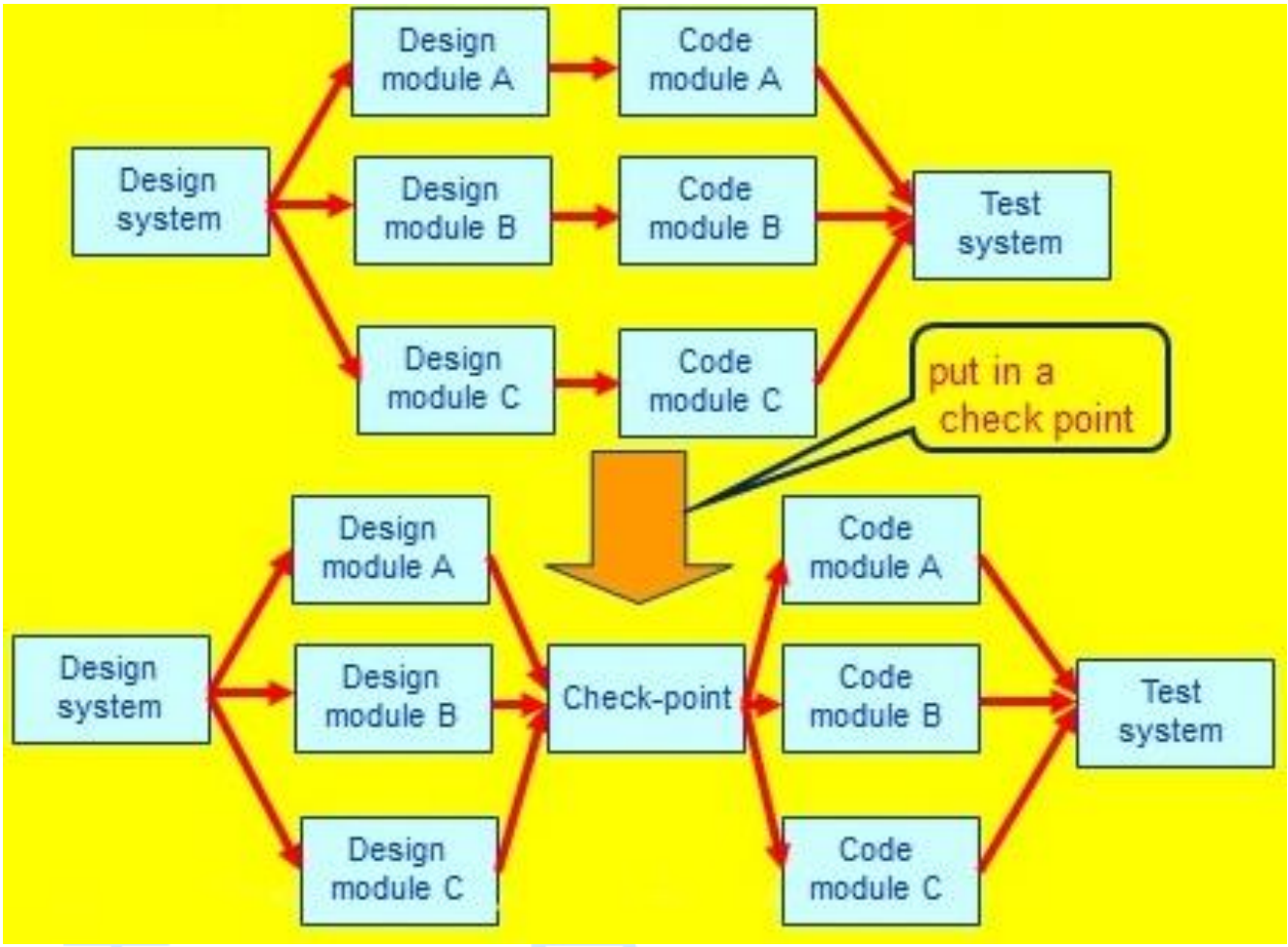


FIGURE B.2 Brightmouth College payroll project activity network fragment

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

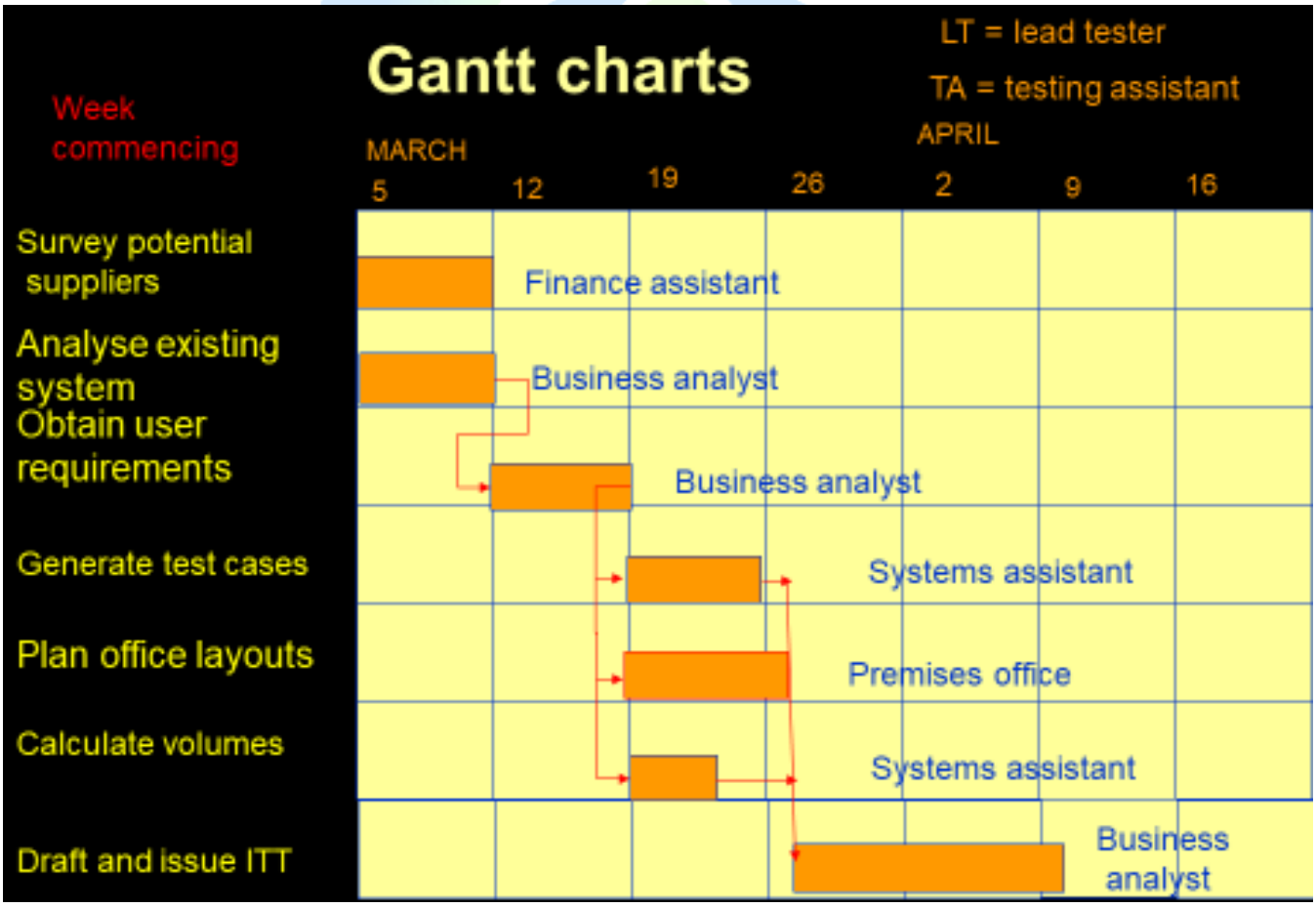
Note that we have already considered some high level risks that could affect the project as a whole at Step 3. Estimates of the effort and duration of activities can be quite tricky. When you produce an estimate for an activity it is worth reflecting about the events which could cause the assumptions 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.

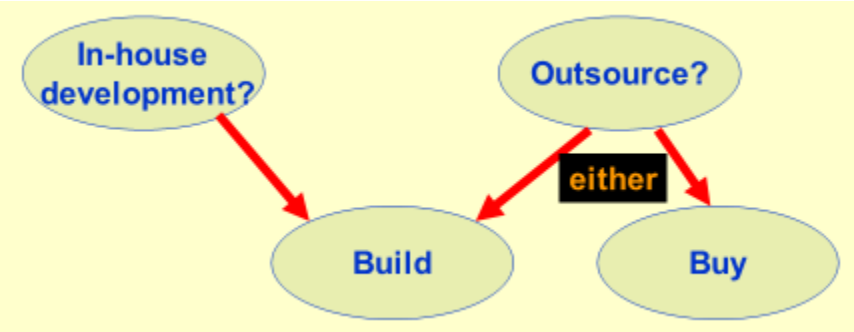
Selection of project approaches

- 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.

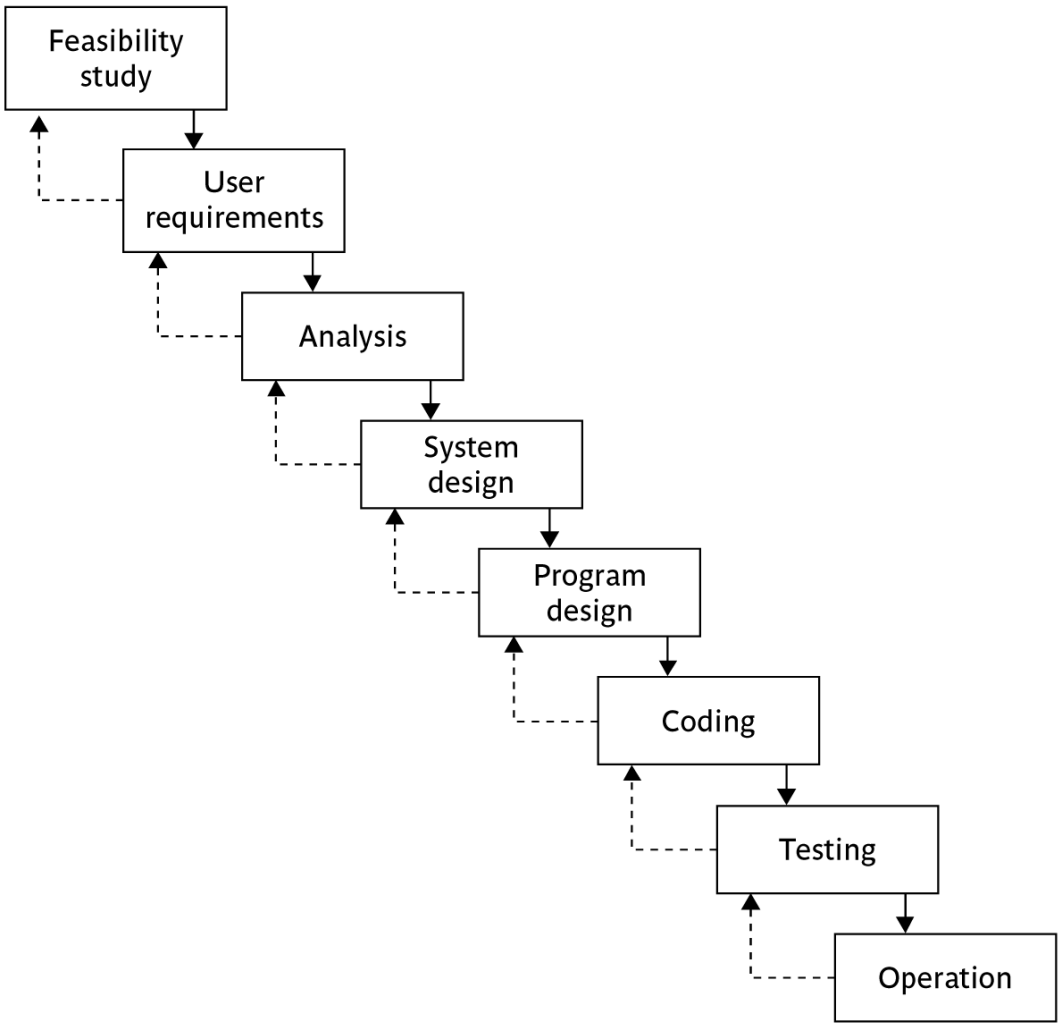
Choice of process models

- '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

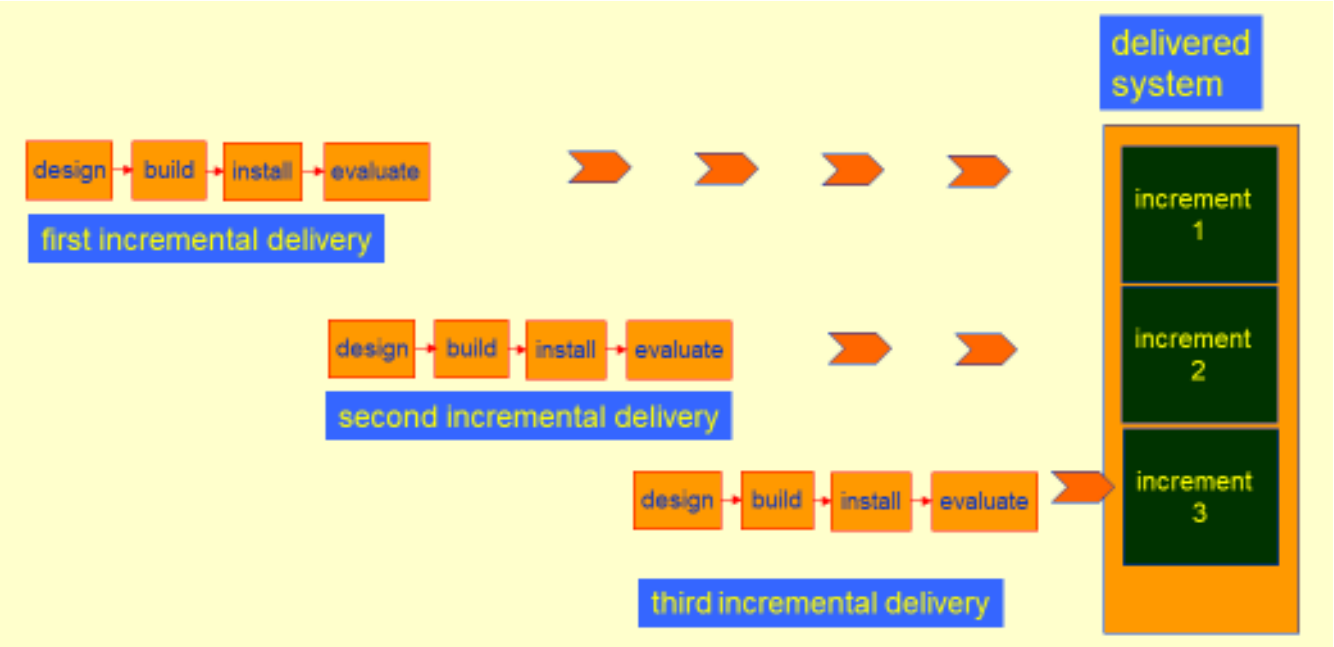
other ways of categorizing prototyping

- 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

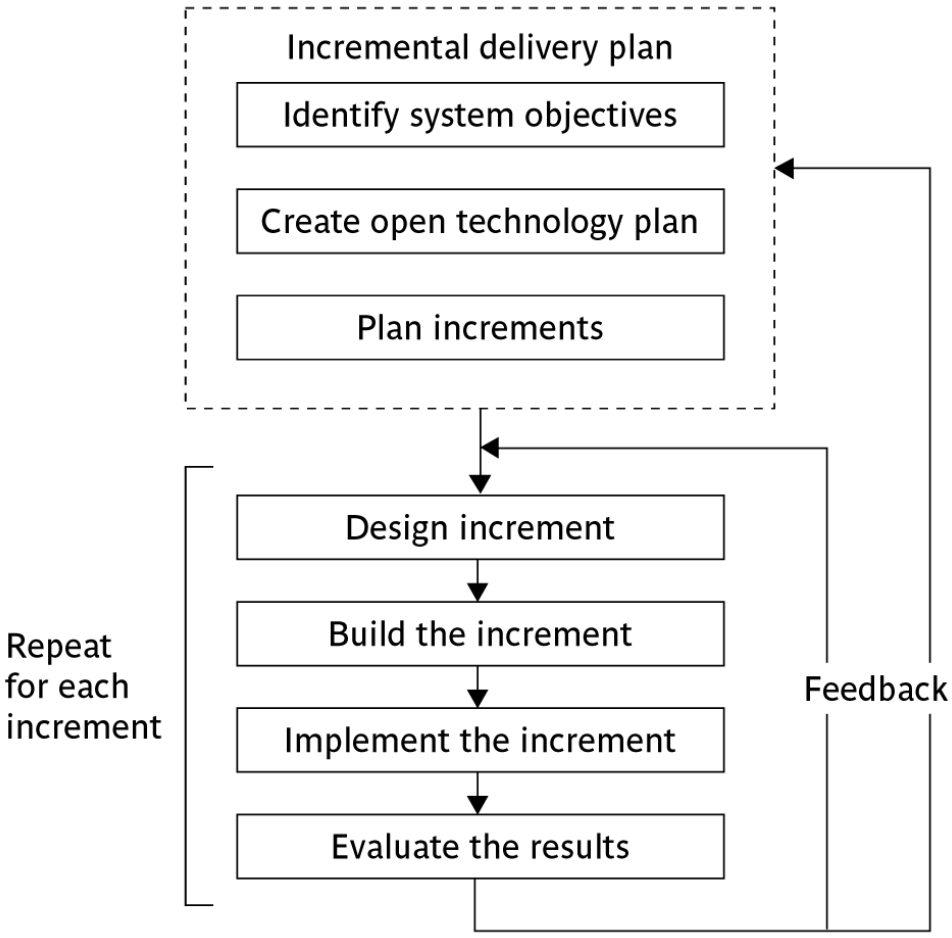
what has been learnt through the development and exercising of the prototype should be documented during the project.

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

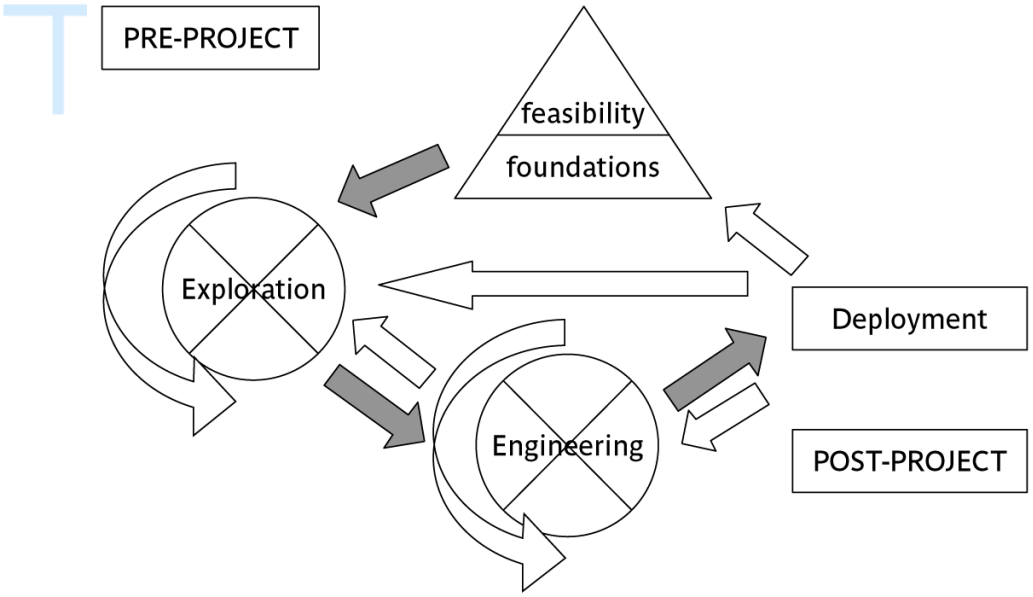


Fig. Atern/DSDM framework

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?

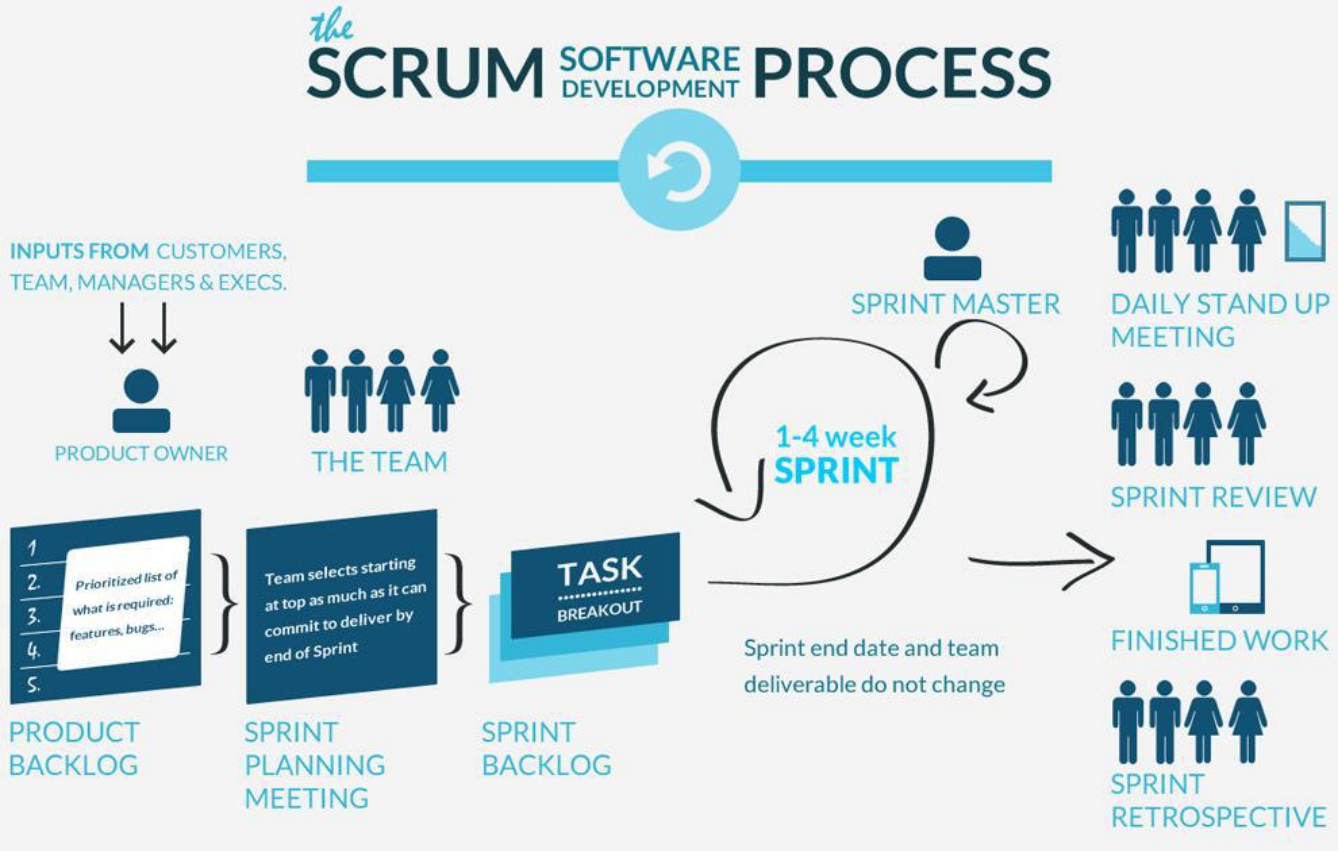
Project Characteristics that Suggest Suitability of Extreme Programming

- 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

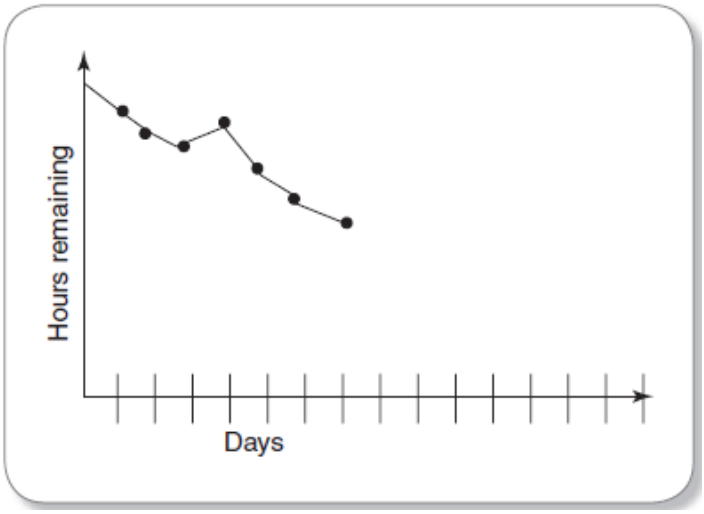
Product Backlog

- 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

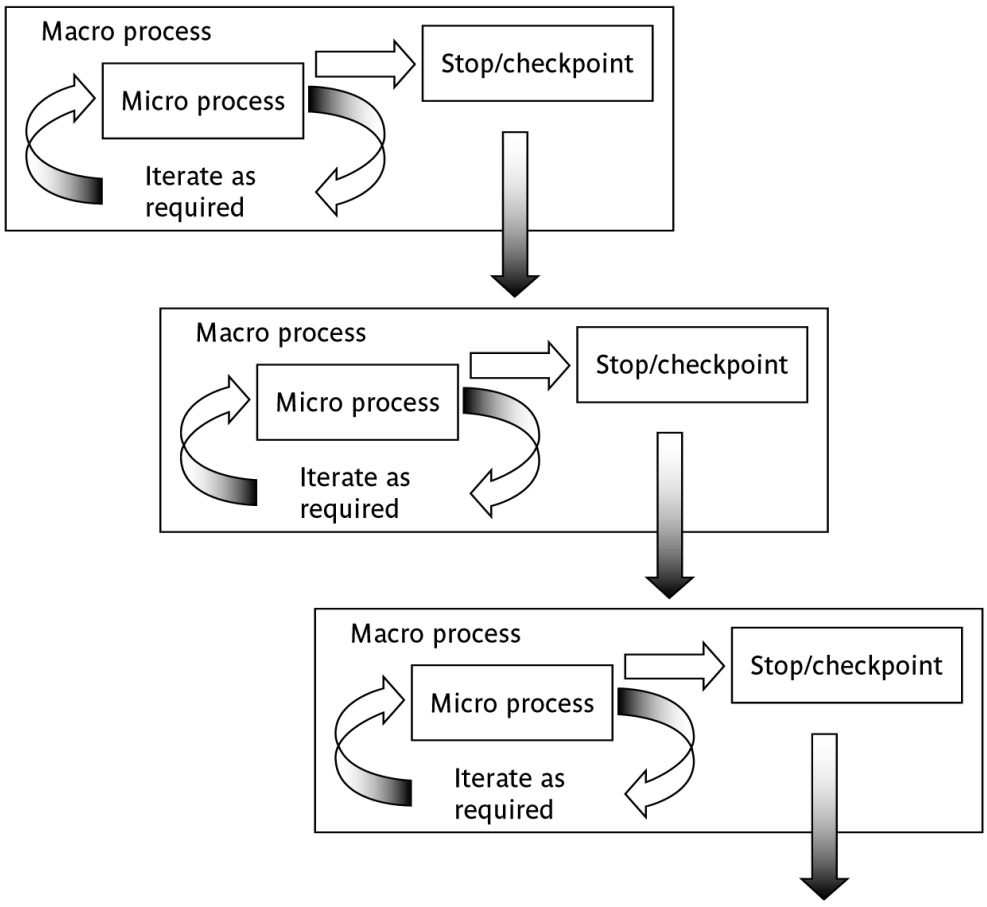
- A subset of Product Backlog Items, which define the work for a Sprint
  - Created by Team members
  - Each Item has it’s own status
  - Updated daily

Sprint Burndown Chart



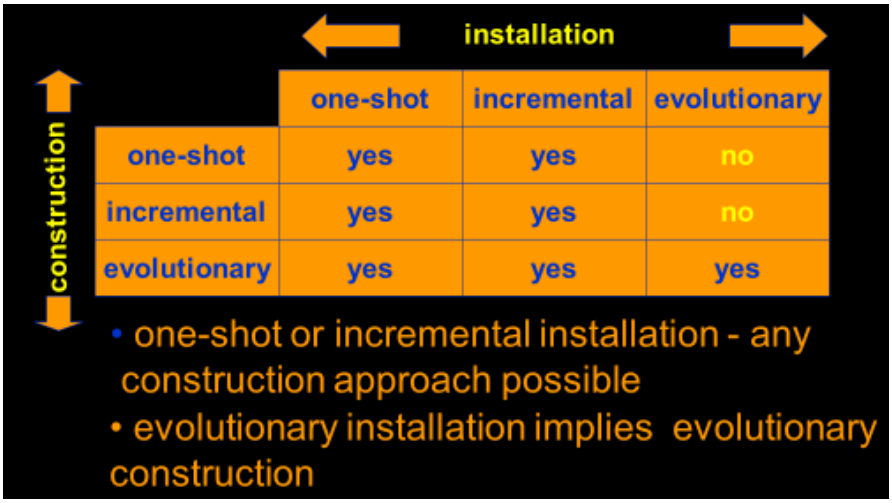
- Day-wise depicts the total Sprint Backlog hours remaining
- Ideally should burn down to zero to the end of the Sprint

macro and micro processes



Section 4.14 discusses these ideas in a little more detail

combinations of approach



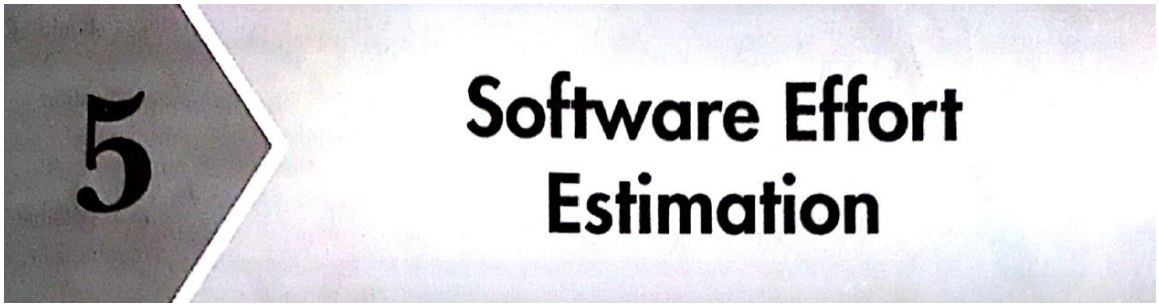
'rules of thumb' about approach to be used

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"

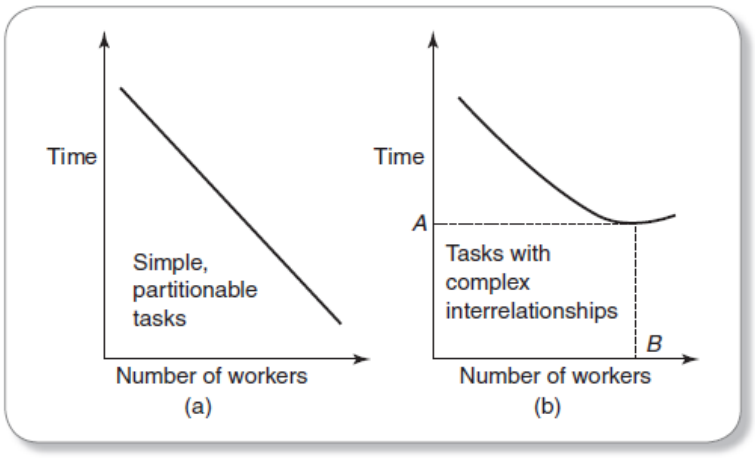


FIGURE 5.2 Impact of addition of workers on the completion time for various types of projects

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

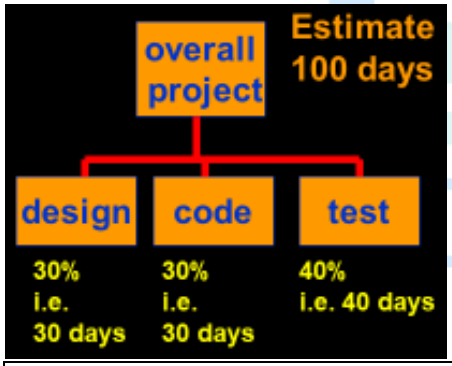
3. Estimate costs for the lowest level activities

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

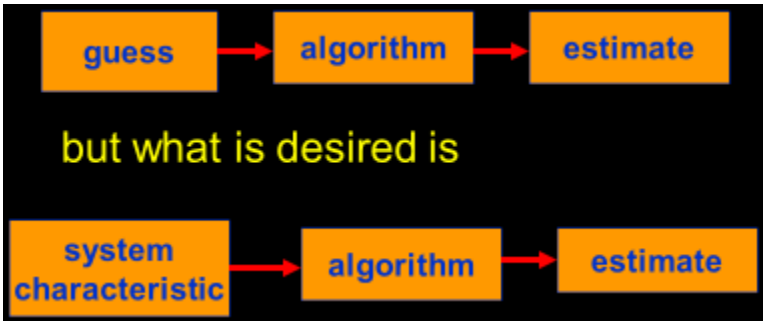


- Produce overall estimate using effort driver(s)
- distribute proportions of overall estimate to components

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

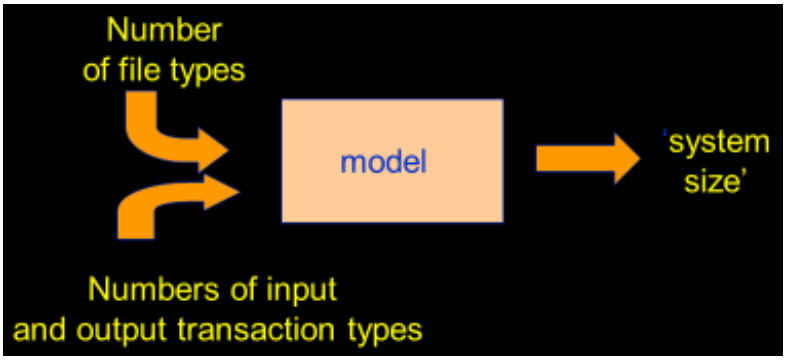
- productivity = (system size) / effort
  - based on past projects

This is analogous to calculating speed from distance and time.

Parametric models

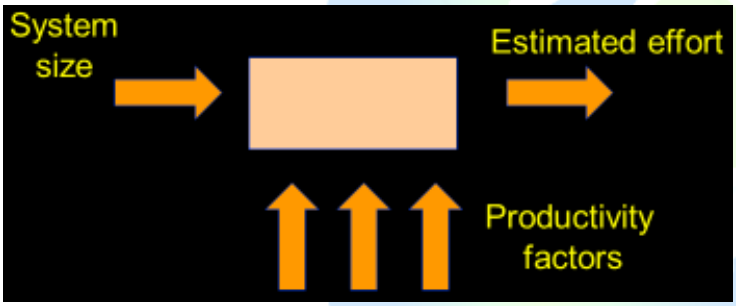
Some models focus on task or system size e.g. Function Points

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

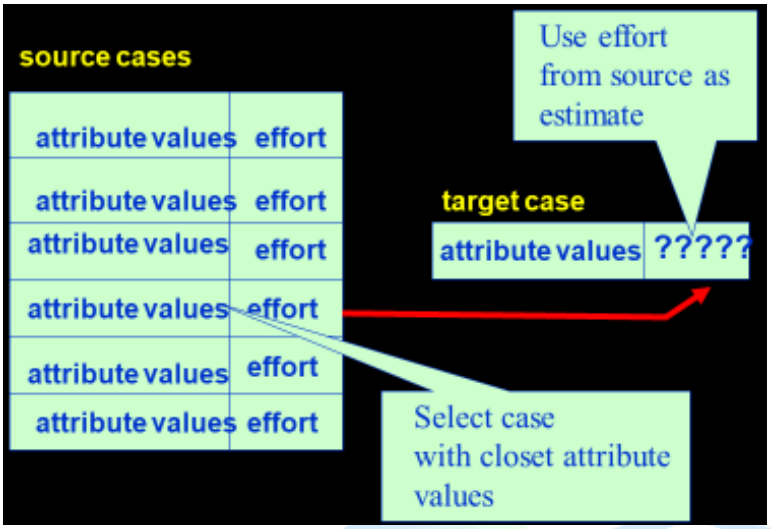


COCOMO originally was based on a size parameter of lines of code (actually 'thousand of delivered sourcecode instructions' or kdsi). Newer versions recognize the use of functions points as a size measure, but convert them to a number called 'equivalent lines of code (eloc).

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

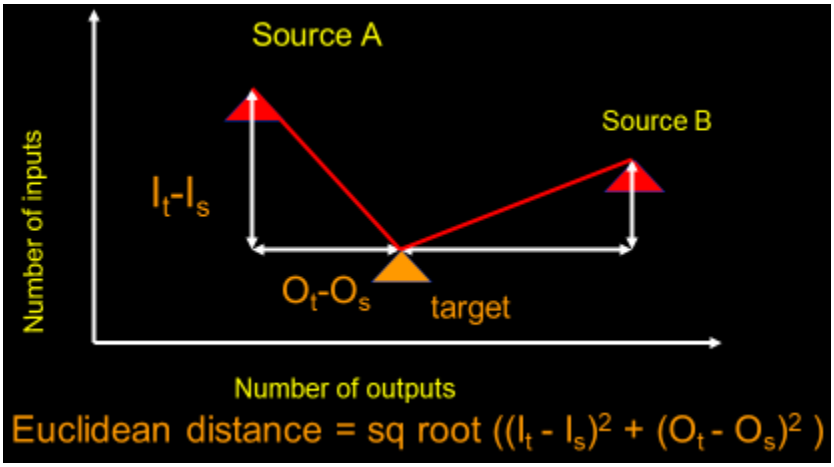


The source cases, in this situation, are completed projects. For each of details of the factors that would have a bearing on effort are recorded. These might include lines of code, function points (or elements of the FP counts such as the number of inputs, outputs etc), number of team members etc etc. 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**

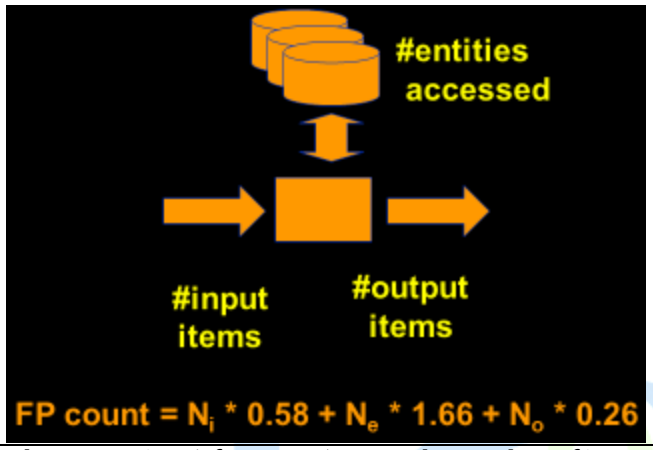
If previous projects delivered 5 FPs a day, implementing the above should take 30/5 = 6 days

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.

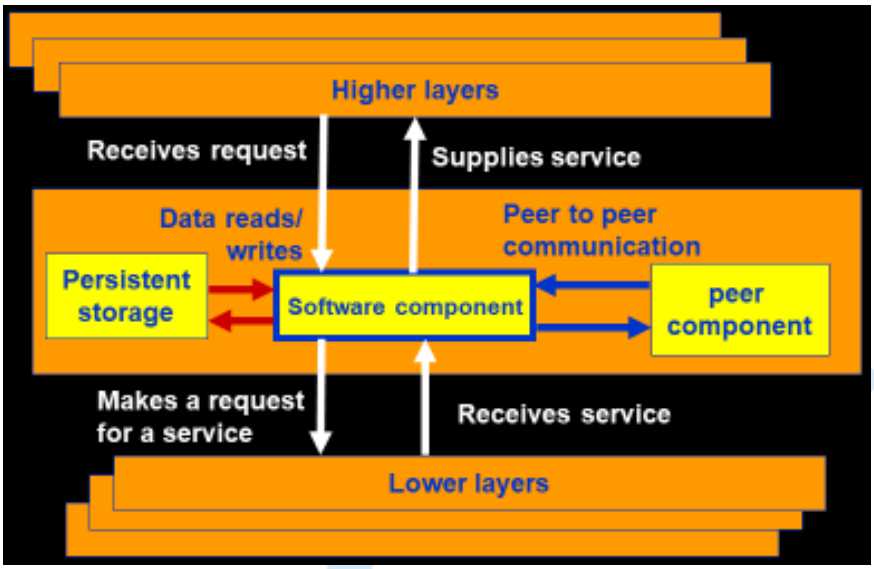
Function points for embedded systems

- 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

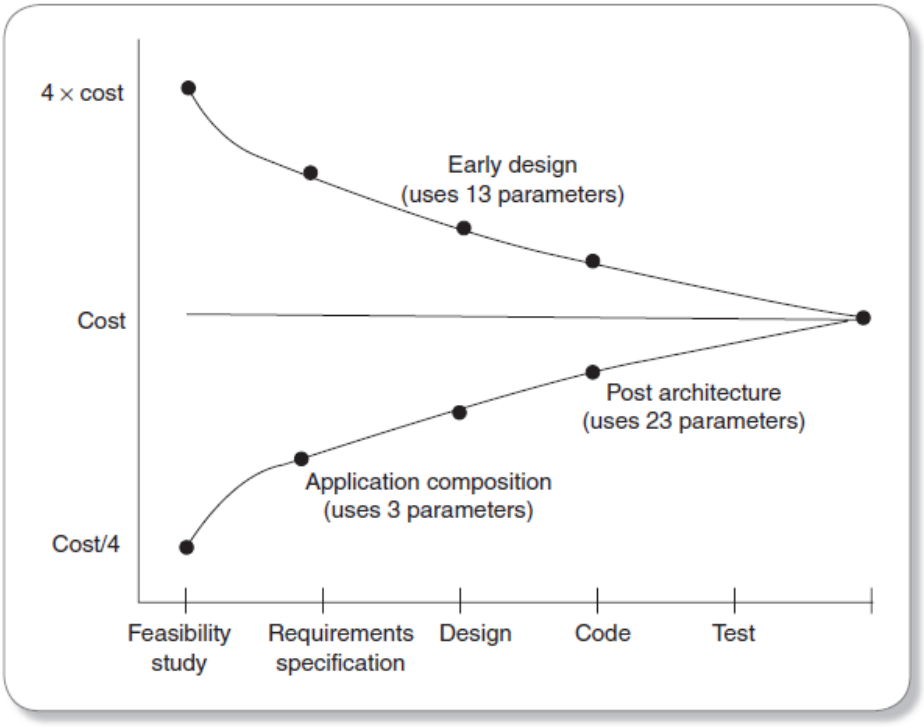


FIGURE 5.4 Accuracy of different COCOMO II estimations

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**

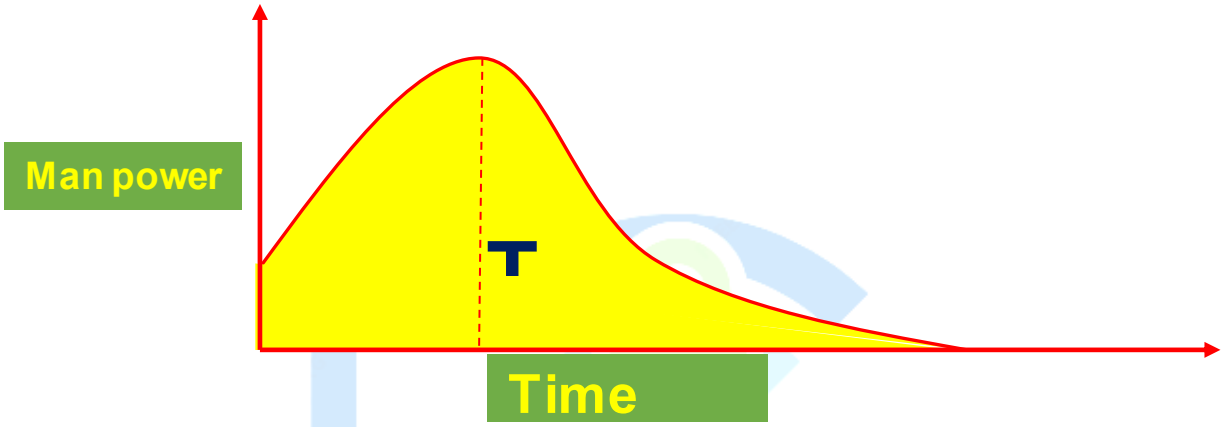
PDIF			**0.87**	**1.00**	**1.29**	**1.81**	**2.61**
PERS	**2.12**	**1.62**	**1.26**	**1.00**	**0.83**	**0.63**	**0.50**
PREX	**1.59**	**1.33**	**1.12**	**1.00**	**0.87**	**0.74**	**0.62**
FCIL	**1.43**	**1.30**	**1.10**	**1.00**	**0.87**	**0.73**	**0.62**
SCED		**1.43**	**1.14**	**1.00**	**1.00**	**1.00**	

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

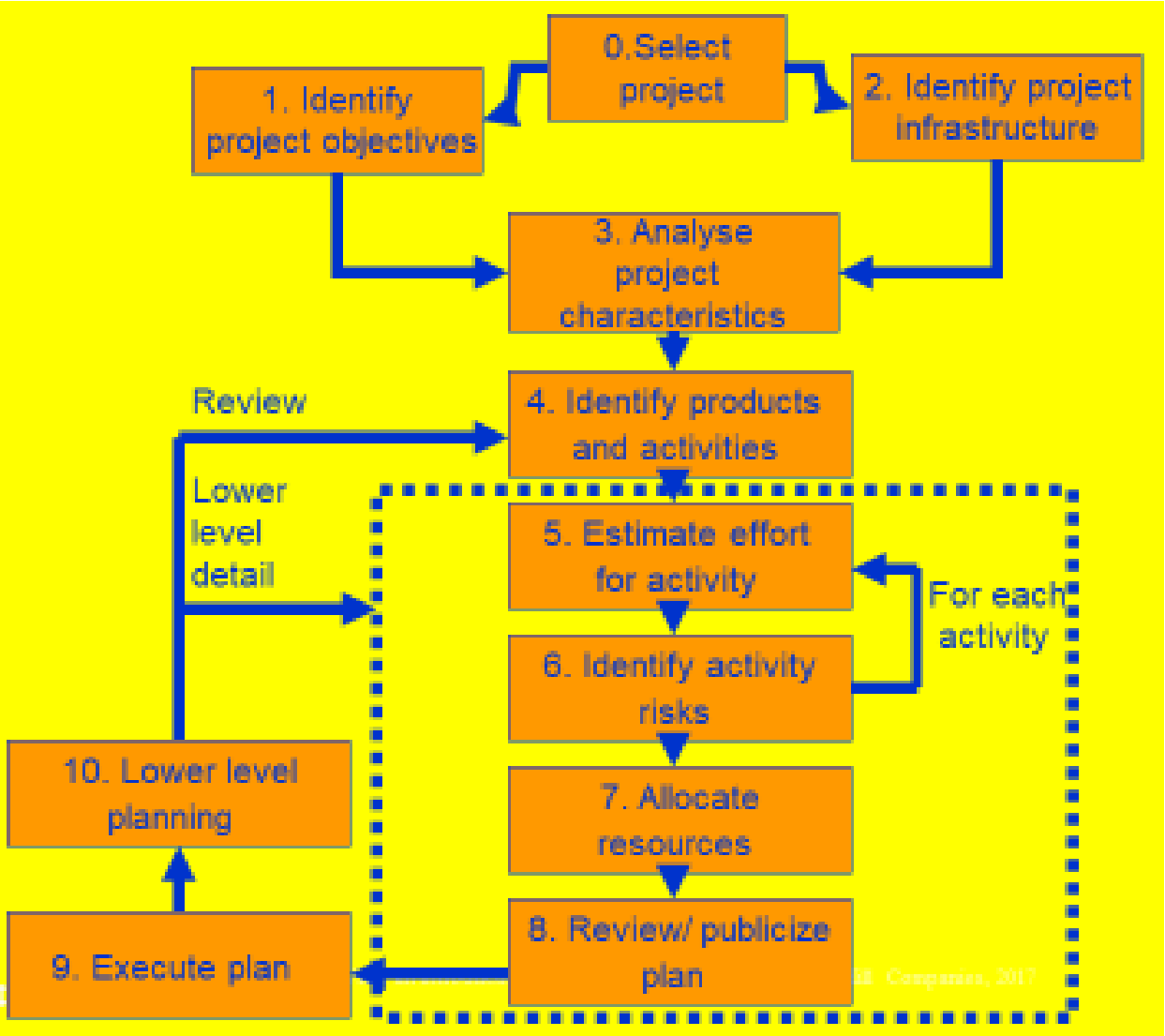
This helps the co-ordination and motivation of the project team

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.