SRINIVASAN COLLEGE OF ARTS & SCIENCE

(Affiliated to Bharathidasan University, Trichy)

PERAMBALUR – 621 212.

DEPARTMENT OF COMPUTER APPLCATIONS

Degree: **MCA** Year: **I** Semester: **II**

LECTURE NOTES ON

P16MCAE2: SOFTWARE PROJECT MANAGEMENT

Prepared by:

Mr. M. RAJKUMAR, MCA., M.Phil., M.Tech., (Ph.D.)

HEAD / CA,

MARCH – 2020

COURSE SYLLABUS

Objective: To impart knowledge related to the various concepts, methods of Software Project Management using management process framework, management disciplines, and risk management techniques.

Unit I: Basics of Software Project Management:

Introduction to Software - Introduction to Software Project Management.

Unit II: Software Project Initiation:

Software Project Evaluation - Contract Management - Requirements Management.

Unit III: Software Project Planning:

Software Estimation Tools - Techniques and Models - Software Project Management Plan - Schedule Management - Cost Management.

Unit IV: Software Project Execution, Monitoring and Control:

Risk Management - Quality Management - Software Project Reviews.

Unit V: Project Closure and Maintenance:

Software Project Closure - Software Maintenance, Support and Implementation.

Text Books:

1. "Software Project Management" - Subramanian Chandramouli, SaikatDutt - Pearson India Education Services Pvt. Ltd, 2015.

2. "Software Project Management" - Bob Hughes & Mike Cotterell – Fourth Edition - 2008 - ISBN: 978-0-07-061985-2.

*****

UNIT-I: BASICS OF SOFTWARE PROJECT MANAGEMENT:

1.1. What is a Project?

The dictionary definitions put a clear emphasis on the project being a *planned* activity. The definition of a project as being planned assumes that to a large extent we can determine how we are going to carry out a task before we start. There may be some projects of an exploratory nature where this might be quite difficult. Planning is in essence thinking carefully about something before you do it – and even in the case of uncertain projects this is worth doing as long as it is accepted that the resulting plans will have provisional and speculative elements. Other activities, relating, for example, to routine maintenance, might have been performed so many times that everyone involved knows exactly what needs to be done. In these cases, planning hardly seems necessary, although procedures might need to be documented to ensure consistency and to help newcomers to the job.

"A project is an activity with specific goals which takes place over a finite period of time". The types of activity that will benefit most from conventional project management are likely to lie between these two extremes – see Figure 1.1. There is a hazy boundary between the non-routine project and the routine job. The first time you do a routine task it will be like a project. On the other hand, a project to develop a system similar to previous ones that you have developed will have a large element of the routine.

The characteristics which distinguish projects can be summarized as follows:

- non-routine tasks are involved;
- planning is required;
- specific objectives are to be met or a specified product is to be created;
- the project has a pre-determined time span;
- work is carried out for someone other than yourself;
- work involves several specialisms;
- work is carried out in several phases;
- the resources that are available for use on the project are constrained;
- The project is large or complex.

The more any of these factors apply to a task, the more difficult that task will be. Project size is particularly important. A project that employs 200 project personnel is going to be trickier to manage than one with just two people. The examples and exercises used in this book usually relate to smaller projects. This is just to make them more manageable from a learning point of view: the techniques and issues discussed are of equal relevance to larger projects.

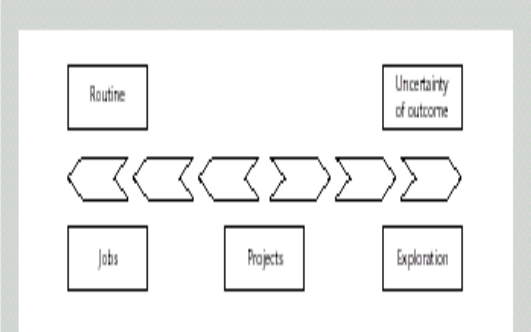


Figure 1.1 Activities most likely to benefit from project management

Examples of projects include:

- Developing a new product or service.
- Effecting a change in structure, staffing, or style of an organization.
- Designing a new transportation vehicle.
- Developing or acquiring a new or modified information system.
- Constructing a building or facility.
- Building a water system for a community in a developing country.
- Running a campaign for political office.
- Implementing a new business procedure or process.

1.2. What is Project Management?

Project Management is the art of maximizing the probability that a project delivers its goals on **Time**, to **Budget** and at the required **Quality**.
Project management is the **application of knowledge, skills, tools, and techniques** to project activities to meet project requirements. Project management is accomplished through the use of the processes such as: initiating, planning, executing, controlling, and closing. It is important to note that many of the processes within project management are iterative in nature.

1.3. Software projects versus other Types of project:

Many of the techniques of general project management are applicable to software project management, but Fred Brooks pointed out that the products of software projects have certain characteristics which make them different. One way of perceiving software project management is as the process of making visible that which is invisible.

Invisibility When a physical artefact such as a bridge or road is being constructed the progress being made can actually be seen. With software, progress is not immediately visible.

Complexity Per dollar, pound or euro spent, software products contain more complexity than other engineered artefacts.

Conformity The 'traditional' engineer is usually working with physical systems and physical materials like cement and steel. These physical systems can have some complexity, but are governed by physical laws that are consistent. Software developers have to conform to the requirements of human clients. It is not just that individuals can be inconsistent. Organizations, because of lapses in collective memory, in internal communication or in effective decision-making can exhibit remarkable 'organizational stupidity' that developers have to cater for.

Flexibility The ease with which software can be changed is usually seen as one of its strengths. However, this means that where the software system interfaces with a physical or organizational system, it is expected that, where necessary, the software will change to accommodate the other components

rather than vice versa. This means the software systems are likely to be subject to a high degree of change.

1.4. Contract management and technical project management:

Many organizations contract out IT development to outside specialist developers. In such cases, the client organization will often appoint a 'project manager' to supervise the contract. This project manager will be able to delegate many technically oriented decisions to the contractors. For instance, the project manager will not be concerned about estimating the effort needed to write individual software components as long as the overall project is fulfilled within budget and on time. On the supplier side, there will need to be project managers who are concerned with the more technical management issues.

1.5. What is Software Project Management?

When the plan starts to involve different things happening at different times, some of which are dependent on each other, plus resources required at different times and in different quantities and perhaps working at different rates, the paper plan could start to cover a vast area and be unreadable.

1.6. Activities covered by Software Project Management [SPM]:

A software project is not only concerned with the actual writing of software. In fact, where a software application is bought in 'off the shelf', there may be no software writing as such. This is still fundamentally a software project because so many of the other elements associated with this type of project are present. Usually there are three successive processes that bring a new system into being – see Figure 1.2.

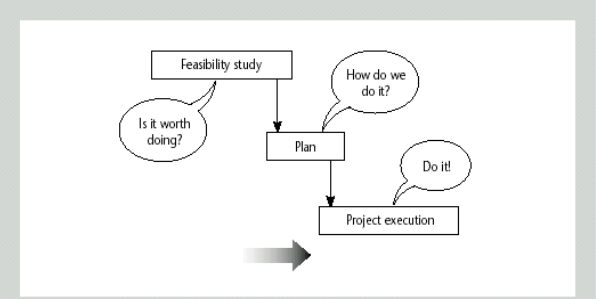


Figure 1.2 The feasibility study/plan/execution cycle

1. The feasibility study This is an investigation into whether a prospective project is worth starting. Information is gathered about the requirements of the proposed application. The probable developmental and operational costs, along with the value of the benefits of the new system, are estimated. With a large system, the feasibility study could be treated as a project in its own right – and have its own planning sub-phase. The study could be part of a strategic planning exercise examining and prioritizing a range of potential software developments. Sometimes an organization has a policy where a group of projects is planned as a *programme* of development.

2. Planning If the feasibility study produces results which indicate that the prospective project appears viable, planning of the project can take place.

However, for a large project, we would not do all our detailed planning right at the beginning. We would formulate an outline plan for the whole project and a detailed one for the first stage. More detailed planning of the later stages would be done as they approached. This is because we would have more detailed and accurate information upon which to base our plans nearer to the start of the later stages.

3. Project execution The project can now be executed. The execution of a project often contains *design* and *implementation* sub-phases. Students new to project planning often find it difficult to separate planning and design, and often the boundary between the two can be hazy. Essentially, design is thinking and making decisions about the precise form of the *products* that the project is to create. In the case of software, this could relate to the external appearance of the software, that is, the user interface, or the internal architecture. The plan lays down the *activities* that have to be carried out in order to create these products. Planning and design can be confused because at the most detailed level, planning decisions are influenced by design decisions. For example, if a software product is to have five major components, then it is likely that there will be five groups of activities that will create them.

Individual projects are likely to differ considerably but a classic project life cycle is shown in Figure 1.3. The stages in the life cycle are described in a little more detail below:

Requirements analysis This is finding out in detail what the users require of the system that the project is to implement. Some work along these lines will almost certainly have been carried out when the project was evaluated, but now the original information obtained needs to be updated and supplemented. Several different approaches to the users' requirements may be explored. For example,

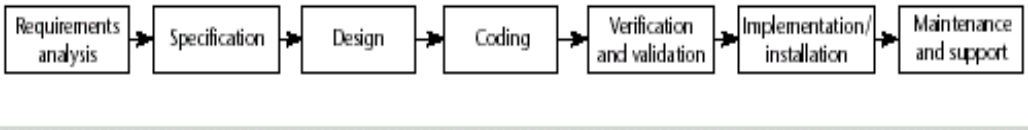


Figure 1.3 A typical project life cycle

Small system which satisfies some, but not all, of the users' needs at a low price might be compared to a system with more functions but a higher price.

Specification Detailed documentation of what the proposed system is to do.

Design A design has to be drawn up which meets the specification. As noted earlier, this design will be in two stages. One will be the external or user design concerned with the external appearance of the application. The other produces the physical design which tackles the way that the data and software procedures are to be structured internally.

Coding This may refer to writing code in a procedural language such as C or Ada or could refer to the use of an application-builder such as Microsoft Access. Even where software is not being built from scratch, some modification to the base package could be required to meet the needs of the new application.

Verification and validation Whether software is developed specially for the current application or not, careful testing will be needed to check that the proposed system meets its requirements.

Implementation/installation Some system development practitioners refer to the whole of the project after design as 'implementation' (that is, the implementation of the design) while others insist that the term refers to the installation of the system after the software has been developed. In this latter case it includes setting up operational data files and system parameters, writing user manuals and training users of the new system.

Maintenance and support Once the system has been implemented there is a continuing need for the correction of any errors that may have crept into the system and for extensions and improvements to the system. Maintenance and support activities may be seen as a series of minor software projects. In many environments, most software development is in fact maintenance.

1.7. Plans, methods and methodologies:

A plan for an activity must be based on some idea of a *method* of work. To take a simple example, if you were asked to test some software, even though you do not know anything about the software to be tested, you could assume that you would need to:

- analyse the requirements for the software;
- devise and write test cases that will check that each requirement has been satisfied;
- create test scripts and expected results for each test case;
- compare the actual results and the expected results and identify discrepancies.

While a *method* relates to a type of activity in general, a *plan* takes that method (and perhaps others) and converts it to real activities, identifying for each activity:

- its start and end dates;
- who will carry it out;
- what tools and materials will be used.

'Material' in this context could be information, for example, a requirements document. With complex procedures, several methods may be deployed, in sequence or in parallel. The output from one method might be the input to another. Groups of methods or techniques are often referred to as *methodologies*. Object oriented design, for example, can be seen as a methodology made up of several component methods.

1.8. Some ways of categorizing software projects:

It is important to distinguish between the main types of software project because what is appropriate in one context may not be so in another. For example, some have suggested that the object-oriented approach, while useful in many contexts, might not be ideal for designing applications to be built around relational databases.

a. Information systems versus embedded systems:

A distinction may be made between *information systems* and *embedded systems*. Very crudely, the difference is that in the former case the system interfaces with the organization, whereas in the latter case the system interfaces with a machine. A stock control system would be an information system that controls when the organization reorders stock. An embedded, or process control, system might control the air-conditioning equipment in a building. Some systems may have elements of both so that the stock control system might also control an automated warehouse.

b. Objectives versus products:

Projects may be distinguished by whether their aim is to produce a *product* or to meet certain *objectives*. A project might be to create a product the details of which have been specified by the client. The client has the responsibility for justifying the product. On the other hand, the project may be required to meet certain objectives. There could be several ways of achieving these objectives. A new information system might be implemented to improve some service to users inside or outside an organization. The level of service that is the target would be the subject of an agreement rather than the characteristics of a particular information system. Service level agreements are becoming increasingly important as organizations contract out functions to external service suppliers.

Many software projects have two stages. The first stage is an objectives-driven project which results in a recommended course of action and may even specify a new software system to meet identified requirements. The next stage is a project actually to create the software product.

1.9. Management Definition:

The Open University Software Project Management module suggested that management involves the following activities:

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

1.10. Problems with software projects:

Management has been seen as the preserve a distinct class within the organization. As technology has made the tasks undertaken by an organization more sophisticated, many management tasks have become dispersed throughout the organization; there are management systems rather than managers. A person who makes projects successful have to focus on the overcoming of obstacles by the problems that confront the project. A survey managers has identified the following commonly experienced problems:

- Poor estimates and plans;
- Lack of quality standards and measures;
- Lack of guidance about making organizational decisions;
- Lack of techniques to make progress visible;
- Poor role definition – who does what?
- Incorrect success criteria.

Above lists are from the manager's point of view.
Below are the problems from the view of Staff members of the project team:

- Inadequate specification of work;
- Management ignorance of ICT;
- Lack of knowledge of application area;
- Lack of standards;
- Lack of up-to-date documentation;
- Preceding activities not completed on time;
- Lack of communication between the users and technicians;
- Lack of communication leading to duplicate of work;
- Lack of commitment;
- Narrow scope of technical expertise;
- Changing software environments;
- Deadline pressure;
- Lack of quality control;
- Remote management;
- Lack of training.

1.11. Project Phases:

Organizations performing projects will usually divide each project into several **Project phases** to improve management control and provide for links to the ongoing operations of the performing organization.

Collectively, the project phases are known as the **project life cycle**. Software development, just like most other activities, has a beginning, middle and an end.

The end of one development activity is sometimes perceived as being linked to the beginning of a new development activity thus producing a cycle of beginning- middle-end, link, beginning-middle-end, link, and so forth.

This view of software development is referred to as the **Software Development Life Cycle [SDLC]**. A project has the following five phases:

- **Initiation:** Articulate your vision for the project, establish goals, assemble your team, and define expectations and the scope of your project.
- **Planning:** Refine the scope, identify specific tasks and activities to be completed, and develop a schedule and budget.
- **Executing:** Accomplish your goals by leading your team, solving problems, and building your project.
- **Controlling** Monitor changes to the project make corrections, adjust your schedule to respond to problems, or adjust your expectations and goals.
- **Closing** Deliver your project to your audience, acknowledge results, and assess its success. Take the time to compose a written evaluation of the project and the development effort.

****

Characteristics of Project Phases:

Each project phase is marked by completion of one or more deliverables. A *deliverable* is a tangible, verifiable work product such as a feasibility study, a detail design, or a working prototype. The deliverables, and hence the phases, are part of a generally sequential logic designed to ensure proper definition of the product of the project.

The conclusion of a project phase is generally marked by a review of both key deliverables and project performance to date, to a) determine if the project should continue into its next phase and b) detect and correct errors cost effectively. These phase-end reviews are often called *phase exits*, *stage gates*, or *kill points*.

Each project phase normally includes a set of defined deliverables designed to establish the desired level of management control. The majority of these items are related to the primary phase deliverable, and the phases typically take their names from these items: requirements, design, build, test, startup, turnover, and others, as appropriate.

Characteristics of the Project Life Cycle

The project life cycle serves to define the beginning and the end of a project. For example, when an organization identifies an opportunity to which it would like to respond, it will often authorize a needs assessment and/or a feasibility study to decide if it should undertake a project. The project life-cycle definition will determine whether the feasibility study is treated as the first project phase or as a separate, standalone project.

Project life cycles generally define:

What technical work should be done in each phase (e.g., is the work of the architect part of the definition phase or part of the execution phase?).
Who should be involved in each phase (e.g., implementers who need to be involved with requirements and design).

Project life-cycle descriptions may be very general or very detailed. Highly detailed descriptions may have numerous forms, charts, and checklists to provide structure and consistency. Such detailed approaches are often called "*project management methodologies*".

Most project life-cycle descriptions share a number of common characteristics:

Cost and staffing levels are low at the start, higher toward the end, and drop rapidly as the project draws to a conclusion.

The probability of successfully completing the project is lowest, and hence risk and uncertainty are highest, at the start of the project. The probability of successful completion generally gets progressively higher as the project continues.

The ability of the stakeholders to influence the final characteristics of the project's product and the final cost of the project is highest at the start and gets progressively lower as the project continues. A major contributor to this phenomenon is that the cost of changes and error correction generally increases as the project continues.

Introduction to Step Wise project planning:

Many different techniques can be used in project planning and this chapter gives an overview of the points at which these techniques can be applied during project planning.

The framework described is called the Step Wise method to help to distinguish it from other methods such as PRINCE2. PRINCE2 is a set of project management standards that were originally sponsored by what is now the Office of Government Commerce (OGC) for use on British government ICT and business change projects. The standards are now also widely used on non-government projects in the United Kingdom. Stepwise should be compatible with PRINCE2. It should be noted, however, that Step Wise covers only the planning stages of a project and not monitoring and control.

In Table 1.1 we outline the general approach that might be taken to planning these projects. Figure 1.4 provides an outline of the main planning activities. Steps 1 and 2 'Identify project scope and objectives' and 'Identify project infrastructure' could be tackled in parallel in some cases. Steps 5 and 6 will need to be repeated for each activity in the project.

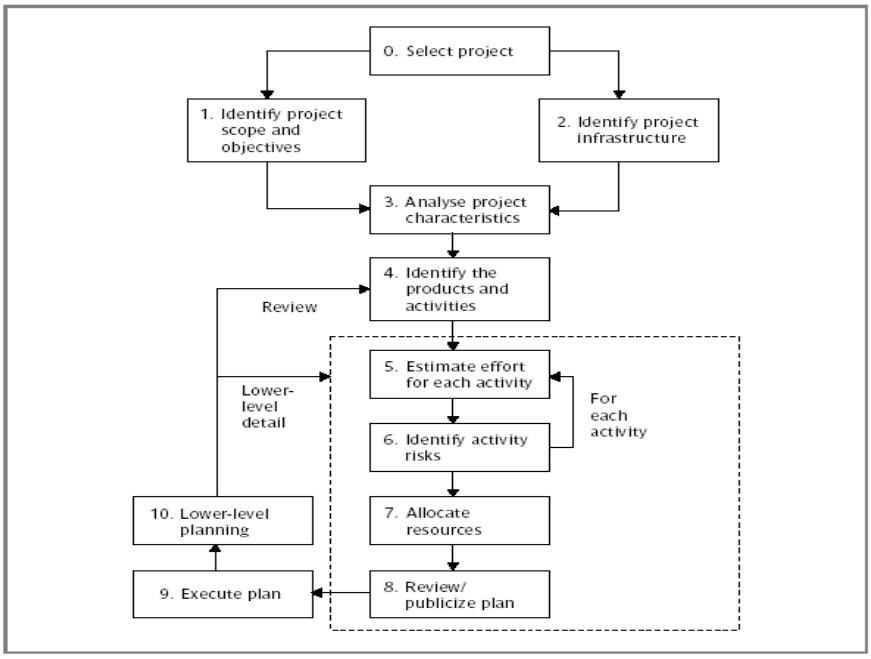


Fig 1.4 An overview of Step Wise Project planning.

Table: An outline of Stepwise Planning Activities:

Step	Activities within step
0	Select project
1	Identify project scope and objectives
1.1	Identify objectives and measures of effectiveness in meeting them
1.2	Establish a project authority
1.3	Identify stakeholders
1.4	Modify objectives in the light of stakeholder analysis
1.5	Establish methods of communication with all parties
2	Identify project infrastructure
2.1	Establish relationship between project and strategic planning
2.2	Identify installation standards and procedures
2.3	Identify project team organization
3	Analyse project characteristics
3.1	Distinguish the project as either objective- or product-driven
3.2	Analyse other project characteristics
3.3	Identify high-level project risks
3.4	Take into account user requirements concerning implementation
3.5	Select general life-cycle approach
3.6	Review overall resource estimates
4	Identify project products and activities
4.1	Identify and describe project products (including quality criteria)
4.2	Document generic product flows
4.3	Recognize product instances
4.4	Produce ideal activity network
4.5	Modify ideal to take into account need for stages and checkpoints
5	Estimate effort for each activity
5.1	Carry out bottom-up estimates
5.2	Revise plan to create controllable activities

6 Identify activity risks
6.1 Identify and quantify activity-based risks
6.2 Plan risk reduction and contingency measures where appropriate
6.3 Adjust plans and estimates to take account of risks
7 Allocate resources
7.1 Identify and allocate resources
7.2 Revise plans and estimates to take account of resource constraints
8 Review/publicize plan
8.1 Review quality aspects of project plan
8.2 Document plans and obtain agreement
9/10 Execute plan/lower levels of planning

This may require the reiteration of the planning process at a lower level.

==========

A major principle of project planning is to plan in outline first and then in more detail as the time to carry out an activity approaches. Hence the lists of products and activities that are the result of Step 4 will be reviewed when the tasks connected with a particular phase of a project are considered in more detail. This will be followed by a more detailed iteration of Steps 5 to 8 for the phase under consideration.

Step 0: Select project
This is called Step 0 because in a way it is outside the main project planning process. Proposed projects do not appear out of thin air –some process must decide to initiate this project rather than some other. While a feasibility study might suggest that there is a business case for the project, it would still need to be established that it should have priority over other projects. This evaluation of the merits of projects could be part of project portfolio management.

Step 1: Identify project scope and objectives
The activities in this step ensure that all the parties to the project agree on the objectives and are committed to the success of the project. We have already looked at the importance of the correct definition of objectives.

Step 1.1: Identify objectives and practical measures of the effectiveness in meeting those objectives:

Step 1.2: Establish a project authority: We have already noted that a single overall project authority needs to be established so that there is unity of purpose among all those concerned.

Step 1.3: Stakeholder analysis – identify all stakeholders in the project and their interests: Essentially all the parties who have an interest in the project need to be identified.

Step 1.4: Modify objectives in the light of stakeholder analysis: In order to gain the full cooperation of all concerned, it might be necessary to modify the project objectives. This could mean adding new features to the system which give a benefit to some stakeholders as a means of

assuring their commitment to the project. This is potentially dangerous as the system size may be increased and the original objectives obscured. Because of these dangers, it is suggested that this process be done consciously and in a controlled manner.

Step 1.5: Establish methods of communication with all parties: For internal staff this should be fairly straightforward, but a project leader implementing for example, a payroll system would need to find a contact point with BACS (Bankers Automated Clearing Scheme), for instance. This step could lead to the first draft of a communications plan.

Step 2: Identify project infrastructure

Projects are never carried out in a vacuum. There is usually some kind of existing infrastructure into which the project must fit. Where project managers are new to the organization, they must find out the precise nature of this infrastructure. This could be the case where the project manager works for an outside organization carrying out the work for a client.

Step 2.1: Identify relationship between the project and strategic planning: We know how project portfolio management supported the selection of the projects to be carried out by an organization. Also, how programme management can ensure that a group of projects contribute to a common organizational strategy. There is also a technical framework within which the proposed new systems are to fit. Hardware and software standards, for example, are needed so that various systems can communicate with each other. These technical strategic decisions should be documented as part of an enterprise architecture process. Compliance with the enterprise architecture should ensure that successive ICT projects create software and other components compatible with those created by previous projects and also with the existing hardware and software platforms.

Step 2.2: Identify installation standards and procedures: Any organization that develops software should define their development procedures. As a minimum, the normal stages in the software life cycle to be carried out should be documented along with the products created at each stage. Change control and configuration management standards should be in place to ensure that changes to requirements are implemented in a safe and orderly way. The procedural standards may lay down the quality checks that need to be done at each point of the project life cycle or these may be documented in a separate quality standards and procedures manual. The organization, as part of its monitoring and control policy, may have a measurement programme in place which dictates that certain statistics have to be collected at various stages of a project. Finally the project manager should be aware of any project planning and control standards. These will relate to how the project is controlled: for example, the way that the hours spent by team members on individual tasks are recorded on timesheets. control.

Step 2.3: Identify project team organization: Project leaders, especially in the case of large projects, might have some control over the

way that their project team is to be organized. Often, though, the organizational structure will be dictated to them. For example, a high-level managerial decision might have been taken that software developers and business analysts will be in different groups, or that the development of business-to-consumer web applications will be done within a separate group from that responsible for 'traditional' database applications.

If the project leader does have some control over the project team organization then this would best be considered at a later stage (see Step 7: Allocate resources).

Step 3: Analyse project characteristics: The general purpose of this part of the planning operation is to ensure that the appropriate methods are used for the project.

Step 3.1: Distinguish the project as either objective- or product-driven: This has already been discussed. As development of a system advances it tends to become more product-driven, although the underlying objectives always remain and must be respected.

Step 3.2: Analyse other project characteristics (including quality-based ones): For example, is an information system to be developed or a process control system, or will there be elements of both? Will the system be safety critical, where human life could be threatened by a malfunction?

Step 3.3: Identify high-level project risks: Consideration must be given to the risks that threaten the successful outcome of the project. Generally speaking, most risks can be attributed to the operational or development environment, the technical nature of the project or the type of product being created.

Step 3.4: Take into account user requirements concerning implementation: The clients may have their own procedural requirements. For example, an organization might mandate the use of a particular development method.

Step 3.5: Select development methodology and life-cycle approach: The development methodology and project life cycle to be used for the project will be influenced by the issues raised above. The idea of a methodology, that is, the group of methods to be used in a project, was already discussed. For many software developers, the choice of methods will seem obvious: they will use the ones that they have always used in the past. While the setting of objectives involves identifying the problems to be solved, this part of planning is working out the ways in which these problems are to be solved. For a project that is novel to the planner, some research into the methods typically used in the problem domain is worthwhile. For example, sometimes, as part of a project, a questionnaire survey has to be conducted. There are lots of books on the

techniques used in such surveys and a wise move would be to look at one or two of them at the planning stage.

Step 3.6: Review overall resource estimates: Once the major risks have been identified and the broad project approach has been decided upon, this would be a good point at which to re-estimate the effort and other resources required to implement the project. Where enough information is available an estimate based on function points might be appropriate.

Step 4: Identify project products and activities

The more detailed planning of the individual activities now takes place. The longer-term planning is broad and in outline, while the more immediate tasks are planned in some detail.

Step 4.1: Identify and describe project products (or deliverables): Identifying all the things the project is to create helps us to ensure that all the activities we need to carry out are accounted for. Some of these products will be handed over to the client at the end of the project – these are deliverables. Other products might not be in the final configuration, but are needed as intermediate products used in the process of creating the deliverables. These products will include a large number of technical products, such as training material and operating instructions. There will also be products to do with the management and the quality of the project. Planning documents would, for example, be management products. The products will form a hierarchy. The main products will have sets of component products which in turn may have sub-component products and so on. These relationships can be documented in a Product Breakdown Structure (PBS) – see Figure 1.5. In this example the products have been grouped into those relating to the system as a whole, and those related to individual modules. A third 'group', which happens to have only one product, is called 'management products' and consists of progress reports. The asterisk in the progress reports indicates that there will be new instances of the entity 'progress report' created repeatedly throughout the project. Note that in Figure 1.5 the only boxes that represent tangible products are those at the bottom of the hierarchy that are not further subdivided. Thus there are only six individual product types shown in the diagram. The boxes that are higher up – for example 'module products' – are simply the names of groups of items. Some products are created from scratch, for example new software components. A product could quite easily be a document, such as a software design document. It might be a modified version of something that already exists, such as an amended piece of code. A product could even be a person, such as a 'trained user', a product of the process of training. These specify that products at the bottom of the PBS should be documented by Product Descriptions which contain:

- the name/identity of the product;
- the purpose of the product;
- the derivation of the product (that is, the other products from which it is derived);

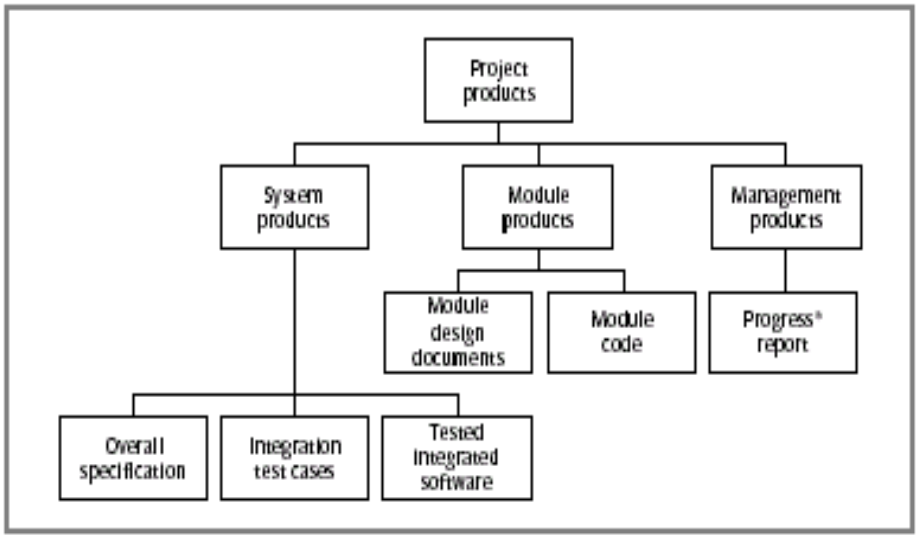


Fig 1.5 A fragment of a Product Breakdown Structure for a system development task.

- the composition of the product;
- the form of the product;
- the relevant standards;
- the quality criteria that define whether the product is acceptable.

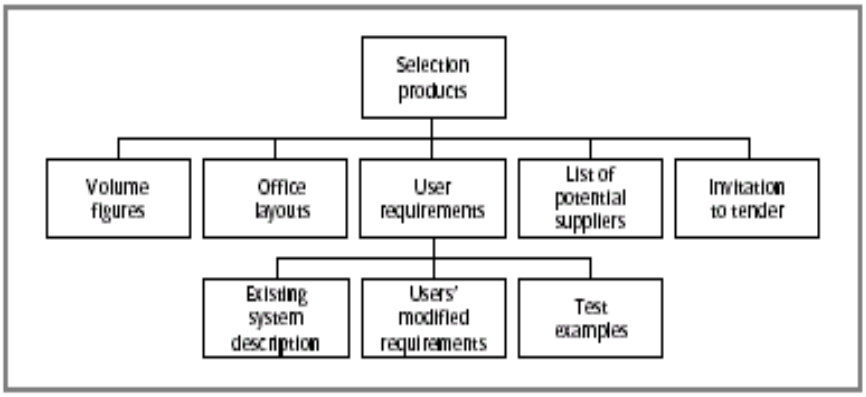


Fig 1.6 A Product Breakdown Structure (PBS) for the products needed to produce an invitation to tender (ITT)

Step 4.2: Document generic product flows: Some products will need one or more other products to exist first before they can be created. For example, a program design must be created before the program can be written and the program specification must exist before the design can be commenced. These relationships can be portrayed in a Product Flow Diagram (PFD). Figure 1.7 gives an example. Note that the 'flow' in the diagram is assumed to be from top to bottom and left to right.

In the example in Figure 1.7, 'user requirements' is in an oval which means that it is used by the project but is not created by it. It is often convenient to identify an overall product at the bottom of the diagram, in this case 'integrated/tested software', into which all the other products feed. PFDs should not have links between products which loop back iteratively. This is emphatically not because iterations are not

recognized. On the contrary, the PFD allows for looping back at any point. For example, in the PFD shown in Figure 1.7, say that during integration testing it was found that a user requirement had been missed in the overall system specification. If we go back to overall system specification and change it we can see from the PFD that all the products that follow it might need to be reworked. A new

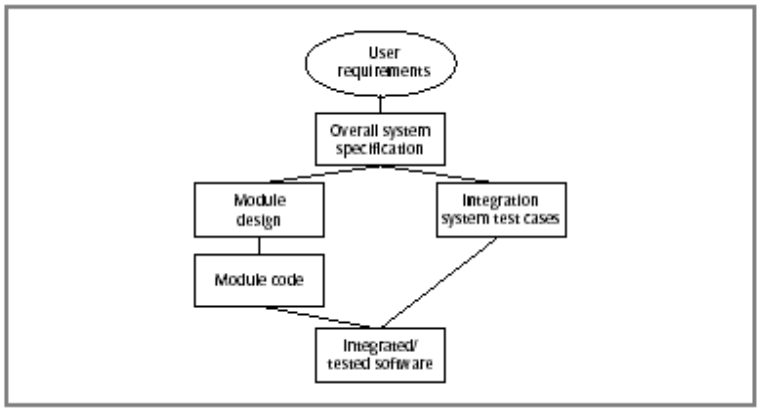


Fig 1.7 A fragment of a Product Flow Diagram (PFD) for a software development task.

module might need to be designed and coded, test cases would need to be added to check that the new requirements had been successfully incorporated, and the integration testing would need to be repeated. The form that a PFD takes will depend on assumptions and decisions about how the project is to be carried out. These decisions may not be obvious from the PFD and so a textual description explaining the reasons for the structure can be helpful.

Step 4.3: Recognize product instances: Where the same generic PFD fragment relates to more than one instance of a particular type of product, an attempt should be made to identify each of those instances. In the example in Figure 1.5, it could be that in fact there are just two component software modules in the software to be built.

Step 4.4: Produce ideal activity network: In order to generate one product from another there must be one or more activities that carry out the transformation. By identifying these activities we can create an activity network which shows the tasks that have to be carried out and the order in which they have to be executed.

The activity networks are 'ideal' in the sense that no account has been taken of resource constraints. For example, in Figure 1.8, it is assumed that resources are available for both software modules to be developed in parallel. A good rule is that activity networks are never amended to take account of resource constraints.

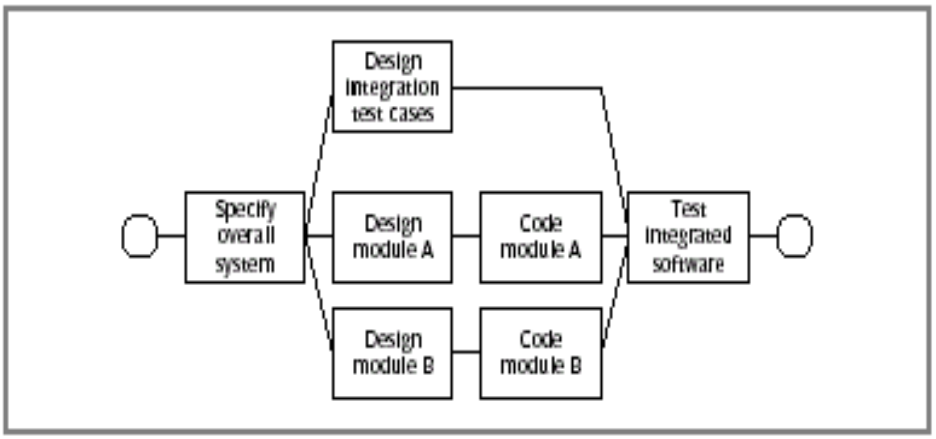


Fig 1.8 An Example of an Activity Network.

Step 4.5: Modify the ideal to take into account need for stages and checkpoints: The approach to sequencing activities described above encourages the formulation of a plan which will minimize the overall duration, or 'elapsed time', for the project. It assumes that an activity will start as soon as the preceding ones upon which it depends have been completed. There might, however, be a need to modify this by dividing the project into stages and introducing checkpoint activities. These are activities which draw together the products of preceding activities to check that they are compatible. This could potentially delay work on some elements of the project – there has to be a trade-off between efficiency and quality. The people to whom the project manager reports could decide to leave the routine monitoring of activities to the project manager. However, there could be some key activities, or milestones, which represent the completion of important stages of the project of which they would want to take particular note. Checkpoint activities are often useful milestones.

Step 5: Estimate effort for each activity

Step 5.1: Carry out bottom-up estimates: Some overall estimates of effort, cost and duration will already have been done (see Step 3.6). At this point, estimates of the staff effort required, the probable elapsed time and the non-staff resources needed for each activity will need to be produced. The method of arriving at each of these estimates will vary depending on the type of activity.

The difference between elapsed time and effort should be noted. Effort is the amount of work that needs to be done. If a task requires three members of staff to work for two full days each, the effort expended is six days. Elapsed time is the time between the start and end of a task. In our example above, if the three members of staff start and finish at the same time then the elapsed time for the activity would be two days.

The individual activity estimates of effort should be summed to get an overall bottom-up estimate which can be reconciled with the previous top-down estimate.

The activities on the activity network can be annotated with their elapsed times so that the overall duration of the project can be calculated.

Step 5.2: Revise plan to create controllable activities: The estimates for individual activities could reveal that some are going to take quite a long time. Long activities make a project difficult to control. If an activity involving system testing is to take 12 weeks, it would be difficult after six weeks to judge accurately whether 50 per cent of the work is completed. It would be better to break this down into a series of smaller subtasks.

There might be a number of activities that are important, but individually take up very little time. For a training course, there might be a need to book rooms and equipment, notify those attending, register students on the training system, order refreshments, copy training materials and so on. In a situation like this it would be easier to bundle the activities into a single merged activity 'make training course arrangements' which could be supplemented with a checklist.

In general, try to make activities about the length of the reporting period used for monitoring and controlling the project. If you have a progress meeting every two weeks, then it would convenient to have activities of two weeks' duration on average, so that progress meetings would normally be made aware of completed tasks each time they are held.

Step 6: Identify activity risks

Step 6.1: Identify and quantify activity-based risks: Risks inherent in the overall nature of the project have already been considered in Step 3. We now want to look at each activity in turn and assess the risks to its successful outcome. Any plan is always based on certain assumptions. Say the design of a component is planned to take five days. This is based on the assumption that the client's requirement is clear and unambiguous. If it is not then additional effort to clarify the requirement would be needed.

The possibility that an assumption upon which a plan is based is incorrect constitutes a risk. In this example, one way of expressing the uncertainty would be to express the estimate of effort as a range of values.

A project plan will be based on a huge number of assumptions, and so some way of picking out the risks that are most important is needed. The damage that each risk could cause and the likelihood of it occurring have to be gauged. This assessment can draw attention to the most serious risks. The usual effect if a problem materializes is to make the task longer or more costly.

Step 6.2: Plan risk reduction and contingency measures where appropriate: It may be possible to avoid or at least reduce some of the

identified risks. On the other hand, contingency plans specify action that is to be taken if a risk materializes. For example, a contingency plan could be to use contract staff if a member of the project team is unavailable at a key time because of serious illness.

Step 6.3: Adjust overall plans and estimates to take account of risks: We may change our plans, perhaps by adding new activities which reduce risks. For example, a new programming language might mean we schedule training courses and time for the programmers to practice their new programming skills on some non-essential work.

Step 7: Allocate resources

Step 7.1: Identify and allocate resources: The type of staff needed for each activity is recorded. The staff available for the project are identified and are provisionally allocated to tasks.

Step 7.2: Revise plans and estimates to take into account resource constraints: Some staff may be needed for more than one task at the same time and, in this case, an order of priority is established. The decisions made here may have an effect on the overall duration of the project when some tasks are delayed while waiting for staff to become free. Ensuring someone is available to start work on an activity as soon as the preceding activities have been completed might mean that they are idle while waiting for the job to start and are therefore used inefficiently. The product of Steps 7.1 and 7.2 would typically be a Gantt chart – see Figure 1.9. The Gantt chart gives a clear picture of when activities will actually take place and highlights which ones will be executed at the same time. Activity networks can be misleading in this respect.

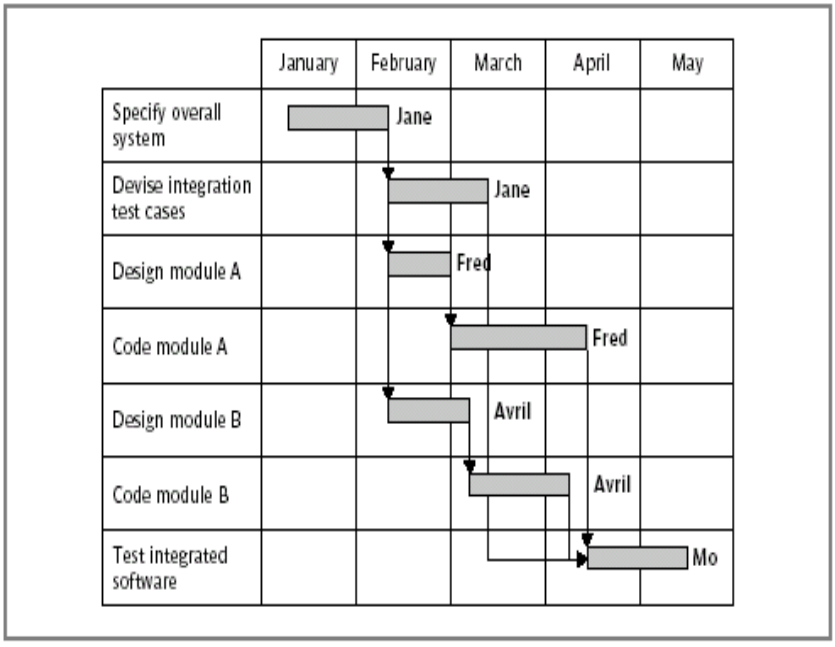


Fig 1.9 Gantt chart showing when staff will be carrying out tasks.

Step 8: Review/publicize plan

Step 8.1: Review quality aspects of the project plan: A danger when controlling any project is that an activity can reveal that an earlier activity was not properly completed and needs to be reworked. This, at a stroke, can transform a project that appears to be progressing satisfactorily into one that is badly out of control. It is important to know that when a task is reported as completed, it really is – hence the importance of quality reviews. Each task should have quality criteria. These are quality checks that have to be passed before the activity can be 'signed off' as completed.

Step 8.2: Document plans and obtain agreement: It is important that the plans be carefully documented and that all the parties to the project understand and agree to the commitments required of them in the plan. This may sound obvious, but it is amazing how often this is not done.

Steps 9 and 10: Execute plan/lower levels of planning

Once the project is under way, plans will need to be drawn up in greater detail for each activity as it becomes due. Detailed planning of the later stages will need to be delayed because more information will be available nearer the start of the stage. Of course, it is necessary to make provisional plans for the more distant tasks, because thinking about what needs to be done can help unearth potential problems, but sight should not be lost of the fact that these plans are provisional.

****

UNIT-2: Software Project Initiation:

PROJECT EVALUATION

1. ***Strategic Assessment***
2. ***Technical Assessment***
3. ***Cost Benefit Analysis***
4. ***Cash Flow Forecasting***
5. ***Cost Benefit Evaluation Techniques***
6. ***Risk Evaluation***

Project Evaluation:

A high level assessment of the project:

- to see whether it is worthwhile to proceed with the project
- to see whether the project will fit in the strategic planning of the whole organization

Why

- Want to decide whether a project can proceed before it is too late
- Want to decide which of the several alternative projects has a better success rate, a higher turnover, a higher
- Is it desirable to carry out the development and operation of the software system

Who

- Senior management
- Project manager/coordinator
- Team leader

When

- Usually at the beginning of the project e.g. Step 0 of Step Wise Framework

What

- Strategic assessment
- Technical assessment
- Economic assessment

How

- Cost-benefit analysis
- Cash flow forecasting
- Cost-benefit evaluation techniques

1. Strategic Assessment (SA):

- Used to assess whether a project fits in the *long-term goal* of the organization
- Usually carried out by senior management
- Needs a strategic plan that clearly defines the objectives of the organization
- Evaluates individual projects against the strategic plan or the overall business objectives Programme management:
  - Suitable for projects developed for use in the organization Portfolio management

- Suitable for project developed for other companies by software houses

SA – Programme Management

Individual projects as components of a programme within the organization. *Programme as "a group of projects that are managed in a coordinated way to gain benefits that would not be possible were the projects to be managed independently".*

SA – Programme Management Issues

Objectives
- How does the project contribute to the *long-term goal* of the organization?
- Will the product increase the market share?
- By how much?

IS plan
- Does the product fit into the overall IS plan?
- How does the product relate to other existing systems?
- Organization structure
- How does the product affect the existing organizational structure? the existing workflow? the overall business model?

MIS
- What information does the product provide?
- To whom is the information provided?
- How does the product relate to other existing MISs?

Personnel
- What are the staff implications?
- What are the impacts on the overall policy on staff development?

Image
- How does the product affect the image of the organization?

SA – Portfolio Management:
- Suitable for product developed by a software company for an organization may need to assess the product for the client.
- Programme management issues need to carry out strategic assessment for the providing software company.

2. Technical Assessment:
- Functionality against hardware and software
- The strategic IS plan of the organization
- Any constraints imposed by the IS plan

Economic Assessment:

Why?

- Consider whether the project is the best among other options
- Prioritise the projects so that the resources can be allocated effectively if several projects are underway

How?

- Cost-benefit analysis
- Cash flow forecasting
- Various cost-benefit evaluation techniques
- NPV and IRR

3. Cost-benefit Analysis: A standard way to assess the economic benefits.

- Two steps
  - Identify and estimate all the costs and benefits of carrying out the project
  - Express the costs and benefits in a common unit for easy comparison (e.g. $)

Costs:

- Development costs
- Setup costs
- Operational costs Benefits
- Direct benefits
- Assessable indirect benefits
- Intangible benefits

Development Costs

- Salaries (base, incentives, and bonuses)
- Equipment for development:
  – Hardware
  – Software

Setup Cost

- Hardware and software infrastructure
- Recruitment/staff training
- Installation and conversion costs

Operational Costs: Costs of operating the system once it has been installed.
– Support costs
– Hosting costs
– Licensing costs
– Maintenance costs
– Backup costs

Benefit Estimation:

Estimate benefits of new system based on– Estimation of cost savings and money generation when deployed– Value of information obtained for objective driven project.

Cost Benefits Types:

- ***Direct benefits***
- ***Indirect benefits***
- ***Intangible benefits***

Direct Benefits: Directly accountable to new system.

- Cost savings (e.g., less staff, less paper, quicker turnaround)
- Money generation (e.g., new revenue stream, new markets) Measurable after system is operational
- Have to be estimated for cost/benefit analysis

Intangible Benefits:

- Positive side effects of new system
- External system (e.g., increase branding, entry to new markets)
- Internal system (increased interest in job for users, enabler for other systems) Often very specific to a project; not measurable even after a system is operational Part of strategic decision rather than cost/benefit analysis

4. Cash Flow Forecasting:

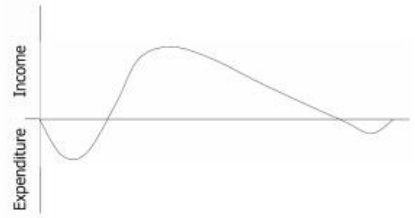
What?

- Estimation of the cash flow over time

Why?

- An excess of estimated benefits over the estimated costs is not sufficient
- Need detailed estimation of benefits and costs versus time

Need detailed estimation of benefits and costs versus time



- Need to forecast the expenditure and the income
- Accurate forecast is not easy
- Need to revise the forecast from time to time

5. Cost-benefit Evaluation Techniques:

- **Net profit** = Total income – Total costs
- **Payback period** = Time taken to break even
- **Return on Investment (ROI** $= \frac{\text{average annual profit}}{\text{total investment}} \times 100\%$

NPV Net present value (NPV):

- It is the sum of the present values of all future amounts.
- *Present value* is the value which a future amount is worth at present
- It takes into account the profitability of a project and the timing of the cash flows

Let *n* be the number of yea r and *r* be the discount rate, the present value (PV) is give

$$PV = \frac{\text{value in year } n}{(1+r)^n}$$

– r is the discount rate
– n is the number of years into the future that the cash flow occurs
– (1+r)^n is known as discount factor

Issues in NPV

- Choosing an appropriate discount rate is difficult
- Ensuring that the rankings of projects are not sensitive to small changes in discount rate

Guidelines:

- Use the standard rate prescribed by the organization
- Use interest rate + premium rate
- Use a target rate of return
- Rank the projects using various discount rates

Disadvantage: May not be directly comparable with earnings from other investments or the costs of borrowing capital

Internal Rate of Return (IRR)

- The percentage discount rate that would produce a NPV of zero

Advantages: Directly comparable with rate of return on other projects and with interest rates

Useful:

- Dismiss a project due to its small IRR value
- Indicate further precise evaluation of a project
- Supported by MS Excel and Lotus 1-2-3

Estimation:

Why? – to define the project budget and to 'refine' the product to realize the budget
Who? – the manager
What? – size and cost
When? – always
How? – techniques and models

Issues related to Estimation

- Difficult to make accurate estimation
- Better to have previous data and analyze the actual values against their estimates so that you know how accurate you are

- Even better to have previous data of the whole organization so that you know how accurate the estimation method, if any, used within the organization

Positive Attitude Towards Estimation

- Use your estimation as a guide to manage your project
- From time to time, you need to revise your estimation based on the current status of the project

Estimation Approaches:

1. **Expert judgment:** Ask the knowledgeable experts
2. **Estimation by analogy:** Use the data of a similar and completed project
3. **Pricing to win:** Use the price that is low enough to win the contract
4. **Top-down:** An overall estimate is determined and then broken down into each component task
5. **Bottom-up:** The estimates of each component task are aggregated to form the overall estimate
6. **Algorithmic model:** Estimation is based on the characteristics of the product and the development environment

Problems related to size estimation

- Nature of software
- Novel application of software
- Fast changing technology
- Lack of homogeneity of project experience
- Subjective nature of estimation
- Political implications within the organization

MANAGING CONTRACTS: INTRODUCTION:

Contract management or **contract administration** is the management of contracts made with customers, vendors, partners, or employees. Contract management includes negotiating the terms and conditions in contracts and ensuring compliance with the terms and conditions, as well as documenting and agreeing on any changes that may arise during its implementation or execution. It can be summarized as the process of systematically and efficiently managing contract creation, execution, and analysis for the purpose of maximizing financial and operational performance and minimizing risk.

Common commercial contracts include employment letters, sales invoices, purchase orders, and utility contracts. Complex contracts are often necessary for construction projects, goods or services that are highly regulated, goods or services with detailed technical specifications, intellectual property (IP) agreements, and international trade.

A study has found that for "42% of enterprises the top driver for improvements in the management of contracts is the pressure to better assess and mitigate risks" and additionally, “nearly 65% of enterprises report that Contract Lifecycle Management (CLM) has improved exposure to financial and legal risk.

TYPES OF CONTRACT:

1) **Fixed price contracts :**

- Price is fixed when the contract is signed.
- Customer knows that when there is no changes in the contract terms they have to pay the price.

--For this type customer requirement must be fixed and known.

2) **Time and material contracts :**

- Customer is charged at a fixed rate per unit of effort.
- Payment is based on the unit of effort applied.

Fixed price per unit delivered contracts:

- Size of system to be delivered is to be calculated.
- Basis for the calculation is in terms of line of code.

Another way of categorizing contracts:

- Open tendering process:
  Any supplier can bid to supply goods and services.
- Restricted tendering process:
  There are bids only from supplier who have been invited by the customer.

Stages in contract placement:

- **Requirement analysis**

-- Before potential supplier is approached there must be clear set of requirements.
--requirement defines functions, standards clearly.
--Each requirement must be defined as compulsory or optional.

- **Evaluation plan**

--after requirement list is drawn
--now there is need to evaluate the proposal.
--First check for all compulsory requirements
--check the quality standards
--cost also taken into account

- **Invitation to tender**

--Deadline must be specified

- **Evaluation of proposals**

--check the document that it contains all requirements
--interviewing suppliers
--demonstrations
--site visits
--practical tests

Typical terms of a contract

- Definition-Form of agreement-lease, license, Sale
- Goods and services to be supplied
- Ownership of software
- Environment
- Acceptance Standards

- Time table
- Price and payment method
- Legal requirements

Contract Management-STEPS

- There must be communication between supplier and customer while the contracted work is carried out.
- This interaction leads to changes which vary the terms of contract.
- When the contract is negotiated, certain points in project may be identified where customer approval is needed.
- Example :-a project to develop the large system could be divided into increments ,for each increment there is interface design phase, customer has to approve the interface first
- For each decision point, the deliverable to be presented by suppliers.
- Most changes to requirement may emerge .This vary the contract terms.

Acceptance

- When the work is completed, customer needs to tack action to carry out acceptance testing.
- The contract may put a time limit on how long acceptance testing can take.

REQUIREMENTS MANAGEMENT:

Preview:

Software requirements engineering is a process of discovery, refinement, modeling and specification. The system requirements and role allocated to software-initially established by the system engineer-are refined in detail. Models of the required data information and control flow and operational behavior are created. Alternative solutions are analyzed and a complete analysis model is created.

Requirements engineering is the systematic use of proven principles, techniques, languages, and tools for the cost effective analysis, documentation, and on-going evolution of user needs and the specification of the external behavior of a system to satisfy those user needs. Notice that like all engineering disciplines, requirements engineering is not conducted in a sporadic random or otherwise haphazard fashion, but instead is the systematic use of proven approaches.

Both the software engineer and customer take an active role in software requirements engineering - a set of activities that is often referred to as *analysis*. The customer attempts to reformulate a sometimes nebulous system-level description of data, function and behavior into concrete detail. The developer acts as interrogator, consultant, problem solver and negotiator.

The overall role of Software in a larger system is identified during system engineering. However, it's necessary to take a harder look at software's role-to understand the specific requirements that must be achieved to build high-

quality software. That's the job of software requirements analysis. To perform the job properly, you should follow a set of underlying concepts and principles. Generally, a software engineer performs requirements analysis However, for complex business applications a 'system analyst' trained in the business aspects of the application domain may perform the task. If you don't analyze, it's highly likely that you'll build a very elegant software solution that solves the wrong problem. The result is wasted time and money, personal frustration and unhappy customers.

Data, functional, and behavioral requirements are identified by eliciting information from the customer. Requirements are refined and analyzed to assess their clarity, completeness, and consistency.

- **Requirements analysis:**

Requirements analysis is a software engineering task that bridges the gap between system level requirements engineering and software design (Figure 1).

Requirements engineering activities result in the specification of software's operational characteristics (function, data; and behavior), indicate software's interface with other system elements, and establish constraints that software must meet. Requirements analysis allows the software engineer (sometimes called analyst in this. role) to refine the software allocation and build models of the data, functional, and behavioral domains that will be treated by software. Requirements analysis provides the software designer with a representation of information, function, and behavior that can be translated to data, architectural, interface, and component-level designs. Finally, the requirements specification provides the developer and the customer with the means to assess quality once software is built.

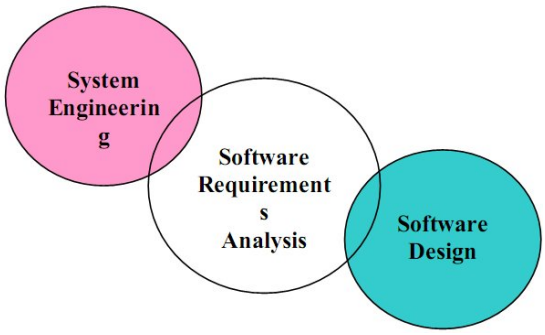


Figure 1: A bridge between system engineering and software design.

Software requirements analysis may be divided into five areas of effort:

(1) Problem recognition,
(2) Evaluation and synthesis,
(3) Modeling,
(4) Specification, and
(5) Review.

Initially, the analyst studies the System Specification (if one exists) and the Software Project Plan. It is important to understand software in a system

context and to review the software scope that was used to generate planning estimates. Next, communication for analysis must be established so that problem recognition is ensured. The goal is recognition of the basic problem elements as perceived by the customer/users.

Problem evaluation:

Problem evaluation and solution synthesis is the next major area of effort for analysis. The analyst must define all externally observable data objects, evaluate the flow and content of information, define and elaborate all software functions, understand software behavior in the context of events that affect the system, establish system interface characteristics, and uncover additional design constraints. Each of these tasks serves to describe the problem so that an overall approach or solution may be synthesized.

For example, an inventory control system is required for a major supplier of auto parts. The analyst finds that problems with the current manual system include: (1) Inability to obtain the status of a component rapidly (2) Two or three-day turnaround to update a card file (3) Multiple reorders to the same vendor because there is no way to associate vendors with components, and so forth.

Once problems have been identified, the analyst determines what information is to be produced by the new system and what data will be provided to the system. For instance, is the customer desires a daily report that indicates what parts have been taken from inventory and how many similar parts remain. The customer indicates that inventory clerks will log the identification number of each part as it leaves the inventory area.

Requirements Elicitation (gathering) for Software:

Before requirements can be analyzed, modeled, or specified they must be gathered through an elicitation process. A customer has a problem that may be amenable to a computer-based solution. A developer responds to the customer's request for help.

Communication has begun. But, as we have already noted, the road from communication to understanding is often full of potholes.

1. Initiating the Process

The most commonly used requirements elicitation technique is to conduct a meeting or interview. The first meeting between a software engineer (the analyst) and the customer can be likened to the awkwardness of a first date between two adolescents. Neither person knows what to say or ask; both are worried that what they do say will be misinterpreted; both are thinking about where it might lead (both likely have radically different expectations here); both want to get the thing over with, but at the same time, both want it to be a success. Yet, communication must be initiated. Gause and Weinberg [GAU89]

suggest that the analyst start by asking context-free questions. That is, a set of questions that will lead to a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of the first encounter itself. The first set of context- free, questions focuses on the customer, the overall goals, and the benefits. For example; the analyst might ask:

– Who is behind the request for this work?
– Who will use the solution?
– What will be the economic benefit of a successful solution?
– Is there another source for the solution that you need?

These questions help to identify all stakeholders who will have interest in the software to be built. In addition, the questions identify the measurable benefit of a successful implementation and possible alternatives to custom software development.

The next set of questions enables the analyst to gain a better understanding of the problem and the customer to voice his or her perceptions about a solution:

– How would you characterize "good" output that would be generated by a successful solution?
– What problem(s) will this solution address?
– Can you show me (or describe) the environment in which the solution will be used?
– Will special performance issues or constraints affect the way the solution is approached?

The final set of questions focuses on the effectiveness of the meeting. Gause and Weinberg call these meta-questions and propose the following (abbreviated) list:

– Are you the right person to answer these questions? Are your answers "official"?
– Are my questions relevant to the problem that you have?
– Am I asking too many questions?
– Can anyone else provide additional information?
– Should I be asking you anything else?

These questions (and others) will help to "break the ice" and initiate the communication that is essential to successful analysis. But a question and answer meeting format is not an approach that has been overwhelmingly successful. In fact, the Q&A session should be used for the first encounter only and then replaced by a meeting format that combines elements of problem solving, negotiation, and specification.

2. Facilitated Application Specification Techniques:

Too often, customers and software engineers have an unconscious "us and them" mind-set. Rather than working as a team to identify and refine requirements, each constituency defines its own "territory" and communicates through a series of memos, formal position papers, documents, and question

and answer sessions. History has shown that this approach doesn't work very well. Misunderstandings abound, important information is omitted, and a successful working relationship is never established.

It is with these problems in mind that a number of independent investigators have developed a team-oriented approach to requirements gathering that is applied during early stages of analysis and specification, called facilitated application specification techniques "(FAST).

This approach encourages the creation of a joint team of customers and developers who work together to: Identify the problem Propose elements of the solution Negotiate different approaches and specify a preliminary set of solution requirements.

FAST has been used predominantly by the information systems community, but the technique offers potential for improved communication in applications of all kinds.

Many different approaches to FAST have been proposed. Each makes use of a slightly different scenario, but all apply some variation on the following basic guidelines:

– A meeting is conducted at a neutral site and attended by both software engineers and customers.
– Rules for preparation and participation are established.
– An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
– A ‘facilitator’ (can be a; customer, a developer, or an outsider) controls the meeting.
– A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used.

The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere that is conducive to the accomplishment of the goal. To better understand the flow of events as they occur in a typical FAST meeting, we present a brief scenario that outlines the sequence of events that lead up to the meeting, occur during the meeting, and follow the meeting.

Initial meetings between the developer and customer occur and basic questions and answers help to establish the scope of the problem and the overall perception of a solution. Out of these initial meetings, the developer and customer write a one- or two- page "product request." A meeting place, time, and date for FAST are selected and a facilitator is chosen. Attendees from both the development and customer/user organizations are invited to attend. The product request is distributed to all attendees before the meeting date.

3. Quality Function Deployment:

A quality management technique that translates needs of customers into technical requirements of software.

- Normal Requirement: meeting objectives & goals stated for a product or system during meeting.
- Expected Requirement: Implicit to products / system and may be so fundamental that customer does not explicitly state them.
- Exciting Requirement: Features beyond customer's expectation and prove to be very satisfying when present.

4. Use Cases:

As requirements are gathered as part of:

- Informal meeting
- FAST or QFD

S/W Engineer can create a set of scenario that identify a thread of usage for system to be constructed; providing a description of how system will be used.

5. Analysis Principles:

A variety of modeling notations are developed by investigators. Each analysis method has a unique point of view. However all analysis methods are related by a set of operational principles like:

– The information domain of a problem must be represented and understood.
– The functions that the software is to perform must be defined.
– The behavior of the software (as a sequence of external events) must be represented.
– The models that depict information function and behavior must be partitioned in a manner that uncovers details in a layered (or hierarchical) fashion.
– The analysis process should move from essential information toward implementation detail.

6. Software Prototyping:

Analysis should be conducted regardless of the SW engineering paradigm. (Various approaches apply).

In some cases it is possible to apply operational analysis principles and derive a model of SW from which a design can be developed.

In other situation Requirement Elicitation (FAST, QFD etc) is conducted and a model is built, called Prototype.

- **The Software Requirements Specification**

The *Software Requirements Specification* is produced at the culmination of the analysis task, The function and performance allocated to software as part of system engineering are refined by establishing a complete information description, a detailed functional description, a representation of system behavior, an indication of performance requirements and design constraints, appropriate validation criteria, and other: information pertinent to requirements. The National Bureau of Standards; IEEE (Standard No. 830-1984), and the U.S. Department of Defense have all proposed candidate formats for software requirements specifications (as well as other software engineering documentation).

Mode of specification has a great impact on quality of solution. Forcing SWE to work with incomplete, inconsistence, or misleading specifications result in frustration and confusion affecting:

– Quality
– Timeliness and
– Completeness of SW product
– A complete information description
– A detailed functional description
– A representation of system behavior
– An indication of performance requirements and design constraints
– Appropriate validation criteria and other information pertinent to requirements.

The *Information Description* provides a detailed description of the problem that the software must solve. Information content, flow, and structure are documented. Hardware, software, and human interfaces are described (or external system elements: and internal software functions.

A description of each function required to solve the problem is presented in the *Functional Description.*

A processing narrative is provided for each function:

– *Design constraints are stated and justified*
– *Performance characteristics are stated, and*
– *One or more diagrams are included to graphically represent the overall structure of the software and interplay among software functions and other system elements*

The *Behavioral Description* section of the specification examines the operation of the software as a consequence of external events and internally generated control characteristics.

Validation Criteria is probably the most important and, ironically, the most often, neglected section of the *Software Requirements specification.*

– How do we recognize a successful implementation?
– What *classes* of tests must be conducted to validate function, performance, and constraints?

Finally, the specification includes a *Bibliography and Appendix.*

In many cases the *Software Requirements Specification* may be accompanied by an executable prototype (which in some cases may replace the specification), a paper prototype or a *Preliminary User's Manual.* The *Preliminary Users Manual* presents the software as a black box: That is, heavy emphasis is placed on user input and the resultant output. The manual can serve as a valuable tool for uncovering problems at the human/machine interface.

- **Review:**

A review of the *Software Requirements Specification* (and/or prototype) is conducted by both the software developer and the customer.

Extreme care should be taken in conducting the review because the specification forms the foundation of the development phase. The review is first conducted at a macroscopic level; that is, reviewers attempt to ensure that the specification is complete, consistent, and accurate when the overall information, functional, and behavioral domains; are considered.

Once the review is complete, the Software Requirements Specification is "signed- off by both the customer and the developer. The specification becomes a "contract" for software development. Requests for changes in requirements after the specification is finalized will not be eliminated. But the customer should note that each after- the-fact change is an extension of software scope and therefore can increase cost, and/or protract the schedule.

****

UNIT-3: SOFTWARE PROJECT PLANNINNG:

Software Project Estimation:

In software development, software estimation is the estimation of the software size, software development effort, software development cost, and software development schedule for a specified software project in a specified environment, using defined methods, tools, and techniques.

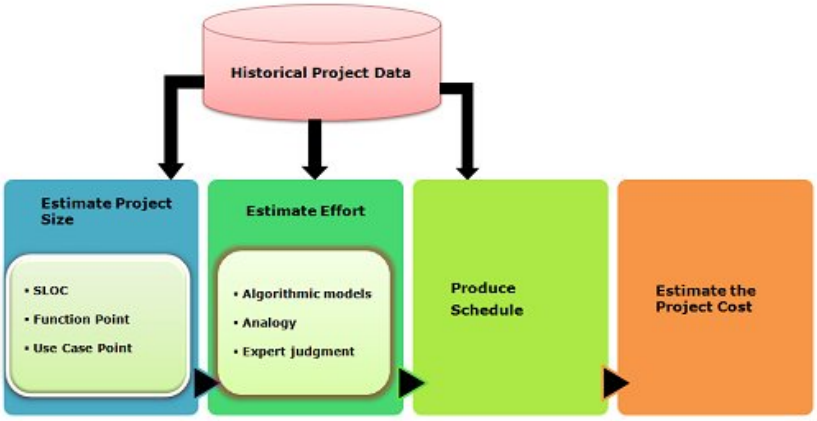


Figure 3.1: Conceptual View of Software Estimation Techniques.

Basic steps for software project estimation as mentioned in above Figure 3.1 are:

1. Estimate the **size** of the development product.
2. Estimate the **effort** in person-months or person-hours.
3. Estimate the **schedule** in calendar months.
4. Estimate the **project cost** in dollars (or local currency).

Effective software project estimation is one of the most challenging and important activities in software development. Proper project planning and control is not possible without a sound and reliable estimate. As a whole, the software industry doesn't estimate projects well and doesn't use estimates appropriately. We suffer far more than we should as a result and we need to focus some effort on improving the situation.

Under-estimating a project leads to under-staffing it (resulting in staff burnout), under-scoping the quality assurance effort (running the risk of low quality deliverables), and setting too short a schedule (resulting in loss of credibility as deadlines are missed). For those who figure on avoiding this situation by generously padding the estimate, over-estimating a project can be just about as bad for the organization! If you give a project more resources than it really needs without sufficient scope controls it will use them. The project is then likely to cost more than it should (a negative impact on the bottom line), take longer to deliver than necessary (resulting in lost opportunities), and delay the use of your resources on the next project.

Software Project Estimation:

The four basic steps in software project estimation are:

1) Estimate the size of the development product. This generally ends up

in either Lines of Code (LOC) or Function Points (FP), but there are other possible units of measure.

2) Estimate the effort in person-months or person-hours.
3) Estimate the schedule in calendar months.
4) Estimate the project cost in dollars (or local currency).

1. Estimating size:

An accurate estimate of the size of the software to be built is the first step to an effective estimate. Your source(s) of information regarding the scope of the project should, wherever possible, start with formal descriptions of the requirements - for example, a customer's requirements specification or request for proposal, a system specification, a software requirements specification. If you are [re-]estimating a project in later phases of the project's lifecycle, design documents can be used to provide additional detail. Don't let the lack of a formal scope specification stop you from doing an initial project estimate. A verbal description or a white-board outline are sometimes all you have to start with. In any case, you must communicate the level of risk and uncertainty in an estimate to all concerned and you must re-estimate the project as soon as more scope information is determined.

Two main ways you can estimate product size are:

1) By analogy. Having done a similar project in the past and knowing its size, you estimate each major piece of the new project as a percentage of the size of a similar piece of the previous project.
Estimate the total size of the new project by adding up the estimated sizes of each of the pieces. An experienced estimator can produce reasonably good size estimates by analogy if accurate size values are available for the previous project and if the new project is sufficiently similar to the previous one.

2) By counting product features and using an algorithmic approach such as Function Points to convert the count into an estimate of size. Macro-level "product features" may include the number of subsystems, classes/modules, methods/functions. More detailed "product features" may include the number of screens, dialogs, files, database tables, reports, messages, and so on.

2. Estimating effort:

Once you have an estimate of the size of your product, you can derive the effort estimate. This conversion from software size to total project effort can only be done if you have a defined software development lifecycle and development process that you follow to specify, design, develop, and test the software. A software development project involves far more than simply coding the software in fact, coding is often the smallest part of the overall effort. Writing and reviewing documentation, implementing prototypes, designing the deliverables, and reviewing and testing the code take up the larger portion of overall project effort. The project effort estimate requires you to identify and estimate, and then sum up all the activities you must perform to build a product of the estimated size.

There are two main ways to derive effort from size:

1) The best way is to use your organization's own historical data to determine how much effort previous projects of the estimated size have taken. This, of course, assumes:

(a) your organization has been documenting actual results from previous projects,

(b) that you have at least one past project of similar size (it is even better if you have several projects of similar size as this reinforces that you consistently need a certain level of effort to develop projects of a given size), and

(c) that you will follow a similar development lifecycle, use a similar development methodology, use similar tools, and use a team with similar skills and experience for the new project.

2) If you don't have historical data from your own organization because you haven't started collecting it yet or because your new project is very different in one or more key aspects, you can use a mature and generally accepted algorithmic approach such as Barry Boehm's COCOMO model or the Putnam Methodology to convert a size estimate into an effort estimate. These models have been derived by studying a significant number of completed projects from various organizations to see how their project sizes mapped into total project effort. These "industry data" models may not be as accurate as your own historical data, but they can give you useful ballpark effort estimates.

3. Estimating schedule:

The third step in estimating a software development project is to determine the project schedule from the effort estimate. This generally involves estimating the number of people who will work on the project, what they will work on (the Work Breakdown Structure), when they will start working on the project and when they will finish (this is the "staffing profile"). Once you have this information, you need to lay it out into a calendar schedule. Again, historical data from your organization's past projects or industry data models can be used to predict the number of people you will need for a project of a given size and how work can be broken down into a schedule.

If you have nothing else, a schedule estimation rule of thumb [McConnell 1996] can be used to get a rough idea of the total calendar time required:
Schedule in months = 3.0 * (effort-months) 1/3 Opinions vary as to whether 2.0 or 2.5 or even 4.0 should be used in place of the 3.0 value – only by trying it out will you see what works for you.

4. Estimating Cost:

There are many factors to consider when estimating the total cost of a project. These include labor, hardware and software purchases or rentals, travel for meeting or testing purposes, telecommunications (e.g., long distance phone calls, video-conferences, dedicated lines for testing, etc.), training courses, office space, and so on.

Exactly how you estimate total project cost will depend on how your organization allocates costs. Some costs may not be allocated to individual projects and may be taken care of by adding an overhead value to labor rates ($ per hour). Often, a software development project manager will only estimate the labor cost and identify any additional project costs not considered "overhead" by the organization.

The simplest labor cost can be obtained by multiplying the project's effort estimate (in hours) by a general labor rate ($ per hour). A more accurate labor cost would result from using a specific labor rate for each staff position (e.g., Technical, QA, Project Management, Documentation, Support, etc.). You would have to determine what percentage of total project effort should be allocated to each position. Again, historical data or industry data models can help.

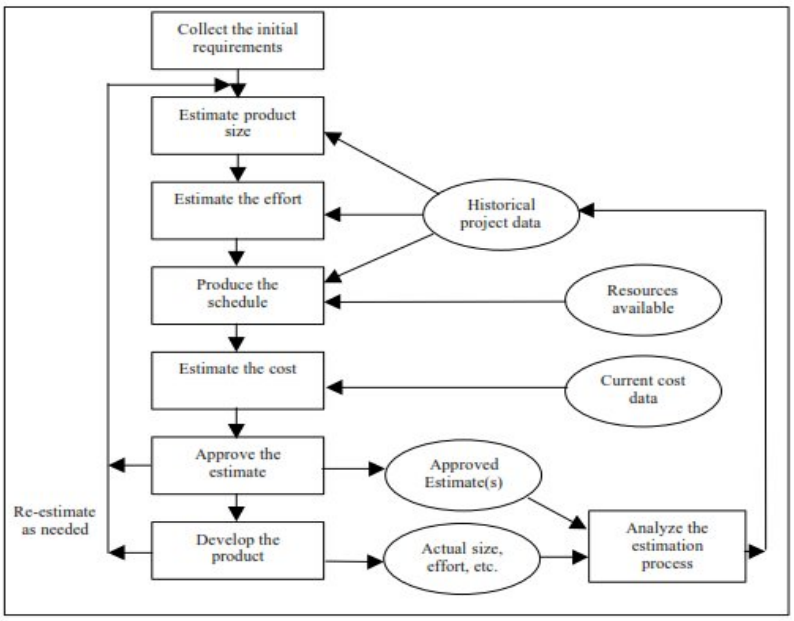


Figure: The Basic Project Estimation Process.

===

Software Project Estimation Tools:

Estimation tools may be stand-alone products or may be integrated into the functionality of larger project management products. Estimation tools may just support the size estimation process, or just the conversion of size to effort, schedule and cost, or both. Project management tools have been discussed in an earlier ADS newsletter (November 1997). Tools that support just size estimation include LOC counters, Function Point analysis programs, and even requirements capture and management applications. This section of this report just focuses on estimation tools that are stand-alone products and support the conversion of size to effort etc.

No estimation tool is the "silver bullet" for solving your estimation problems. They can be very useful items in your estimation toolkit, and you should seriously consider using one (or more), but their output is only as good as the inputs they are given and they require you to have an estimation and software development process in place to support them. Beware of any vendor claiming

their tool is able to produce estimates within +/- some small percentage of actual unless they also highlight all the things you must be able to do in a predictable manner and what must go right during a project to ensure an estimate can be that accurate.

There are a variety of commercial and public domain estimation tools available. Searching for software project estimation tools on the web isn't as straightforward as one might expect. A mix of keywords and search engines needs to be used to discover about 80% of the tools and web-sites where tool lists were available identified the rest. Web-based information on the capabilities and pricing of the tools is variable and occasionally very superficial, so some phone calls and email should be used to augment what is gleaned from the web.

The following provides a summary of the important features and criteria you should consider when evaluating a software project estimation tool.

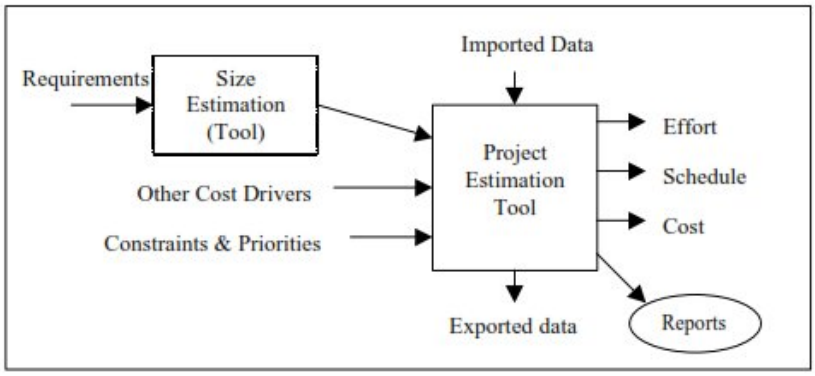


Figure: Estimation Tool Context.

Price: Commercial estimation tools can be categorized as either "for rent" (you pay an annual fee) or "for purchase" (one-time fee), and they come in 3 price ranges: affordable ($1000 or less), mid-range ($1001$5000), or expensive ($5001 to $20000 or more). Probably only larger organizations or large projects would consider the mid-range or high-priced tools. The tools under $1000 implement non-proprietary models published by others (e.g., COCOMO) and may lack some of the functionality and extensive support of the expensive options, but can still generate more than adequate estimates.

Platform and Performance: Does it run on your current hardware/software? Does it run on more than one platform? How much RAM and disk space is required? Will its database handle the amount of historical data and the size and number of project estimates you will be entering?

Ease of Use and Documentation: Can you generate an initial project estimate easily the first time you use the tool, or do you have to study the underlying model in detail for days first learning all the acronyms and struggling with attribute definitions? Can you tailor your estimates easily? Do the user manual and help text provide an understanding how to use the tool to generate project estimates, as well as a simple list of what general functionality the tool possesses? Is sample project data available?

Networking Capability: Is there a common, shared database so that multiple users can access and add to the historical project data, and view or update estimates (assuming this is important to you)?

Upgrades: Model data shouldn't be static – as new programming languages and new development paradigms appear, as different kinds of development projects are studied, updates to the model data in the tool should be made. Does the vendor make model updates available to customers? Is the vendor committed to making continuous enhancements that offer new functionality and support new platforms etc.? What do these upgrades cost?

Support: It is important to understand that although estimation tools are becoming more affordable and easier to use, the model(s) they implement are quite complex and you may have questions or need some guidance now and then. Does the vendor provide technical support and a means for asking "how to" questions? Does the vendor offer estimation training courses that extend beyond just how to use the tool or can they recommend supporting courses and training/reading material?

======

Estimation Model(s) Supported:

Some tools use one or more proprietary models where little detailed information is published; others use non-proprietary models where you can purchase a book and/or download detailed information from the web to learn more. It takes a lot of resources to develop a sophisticated model of software development so its not surprising that there are only a handful of models.

Regardless of how much you can learn about the tool's internal algorithms, what you must determine is whether or not the estimates generated by the tool are useful in estimating your organization's type of software development projects. Parametric models typically have a bias – for example, some suit a military development process while others suit commercial development. The only way to quickly become confident that a tool can give you valuable results is to obtain an evaluation or demo copy and estimate previous projects where actual are known. Compare the estimates from the tool with what you know about previous projects and see if the results are "in the ballpark".

Assess whether the tool allows you to capture historical information on your past projects and how it requires you to enter it. Some tools can be calibrated to your projects only by modifying the underlying model data (i.e., you have to derive the values yourself); others allow you to simply enter project metrics like actual size, effort, schedule and then the tool derives the model data changes. Does the tool support the estimation of maintenance & enhancement projects? Does it have support for object-oriented, COTS, software re-use or other issues important to your projects?

Other Cost Drivers:
The models generally allow you to specify values for a number of cost, or productivity drivers (e.g., staff capabilities and experience, lifecycle

requirements, use of tools, etc.) in order to tailor the estimate to your organization and your project's particular situation. What cost drivers are available and are the values you can set them to useful for your situation?

Constraints and Priorities:
Does the tool allow you to specify constraints (e.g., Maximum Schedule=12 months; Peak Staff=10) when calculating an estimate? Does the tool allow you to specify priorities (e.g., "Shortest possible schedule has highest priority"; "Lowest number of staff has highest priority") when calculating an estimate?

========

Software Estimation Techniques & Models:

- ✓ Bottom-up
- ✓ Algorithmic cost modelling.
- ✓ Expert judgement.
- ✓ Estimation by analogy.
- ✓ Parkinson's Law.
- ✓ Pricing to win.

Algorithmic cost modelling	**A model based on historical cost information that relates some software metric (usually its size) to the project cost is used. An estimate is made of that metric and the model predicts the effort required.**
Expert judgement	**Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached.**
Estimation by analogy	**This technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects. Myers (Myers 1989) gives a very clear description of this approach.**
ParkinsonÕs Law	**ParkinsonÕsLaw states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months.**
Pricing to win	**The software cost is estimated to be whatever the customer has available to spend on the project. The estimated effort depends on the customerÕs budget and not on the software functionality.**

Bottom-up estimation:

- Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.
- Usable when the architecture of the system is known and components identified.
- This can be an accurate method if the system has been designed in detail.
- It may underestimate the costs of system level activities such as integration and documentation.

Top-down estimation:

- Start at the system level and assess the overall system functionality and how this is delivered through sub-systems.
- Usable without knowledge of the system architecture and the components that might be part of the system.
- Takes into account costs such as integration, configuration management and documentation.
- Can underestimate the cost of solving difficult low-level technical problems.

Algorithmic cost modelling:

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers:
  - Effort = A ´ $Size^{B}$ ´ M
  - A is an organisation-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes.
- The most commonly used product attribute for cost estimation is code size.
- Most models are similar but they use different values for A, B and M.

Pricing to win:

- The project costs whatever the customer has to spend on it.
- This approach may seem unethical and un-businesslike.
- However, when detailed information is lacking it may be the only appropriate strategy.
- The project cost is agreed on the basis of an outline proposal and the development is constrained by that cost.
- A detailed specification may be negotiated or an evolutionary approach used for system development.
- **Advantages:** You get the contract.
- **Disadvantages:** The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required.

=======

Introduction to Schedule management:

Definition: Schedule management is the process of developing, maintaining and communicating schedules for time and resource.

General: A schedule is the timetable for a project, programme or portfolio. It shows how the work will progress over a period of time and takes into account factors such as limited resources and estimating uncertainty.

The scheduling process starts with the work that is needed to deliver stakeholder requirements. This includes the technical work that creates outputs, the change management work that delivers benefits, and the management activity that handles aspects such as risk management and stakeholder management.

Some types of work can be defined much more easily than other types. The work involved in building a house is clear from the start. The work involved in maintaining a generator is not clear until inspections are complete. Engineering work tends to have complete specifications from the start, whereas change management and some IT work follow a more iterative approach to defining what needs to be done.

Approaches to calculating schedules have to be equally flexible. In some cases, rigorous techniques can be used to model the work and calculate detailed timings. In other cases, broad estimates have to be made initially, with constant refinement as more information becomes available.

A detailed model can be used to perform 'what-if' calculations to test the result of potential events (e.g. 'What if resource x is not available in February?', or 'What if there is adverse weather in March?').

The detailed and high-level scheduling approaches are both combined in 'rolling wave' scheduling. Short-term work is typically the best defined and can be subject to the most rigorous scheduling. Longer-term work is more vague and subject to change. The window of detail moves along the schedule like a rolling wave.

Schedules: are presented in many different ways in order to suit the circumstances. The choice of presentation will depend upon:

- The level of detail required;
- Whether time and/or resource is being shown;
- The context of the work (e.g. construction, IT, engineering or business change);
- The dimension being scheduled (project, programme or portfolio);
- The target audience.

The most common form of graphical schedule is the Gantt chart. In its simplest form this uses bars on a horizontal timescale to show the start, duration and finish of packages of work. Variants of the Gantt chart can convey all manner of information to suit the circumstances.

A communication management plan is used to explain who is to receive scheduling information and when. The choice of presentation is tailored to the recipients.

Schedules are contained within the P3 management plan. The schedules that form part of the approval of the work become the baseline against which progress is tracked.

Schedules are fundamental to the control of the project, programme or portfolio. Care must be taken in selecting modeling and calculation techniques, forms of presentation and software tools. Scheduling policies will be set out in the P3 management plan so that scheduling is consistent and widely understood.

On conclusion of the work, schedules that show what was planned and what actually happened are an important resource in determining lessons learned.

Project: Outputs are produced at project level and this is where the most detailed scheduling can take place. The approach to scheduling will depend upon the nature of the project in relation to the triple constraints of time, cost and scope.

Some projects may have to be delivered by a specific date, others within a limited budget, or using limited resources. Some projects may have a very well-defined scope of work and others may intend to develop the scope iteratively throughout the life cycle (often known as Agile project management).

Where there is a well-defined scope of work it will often be presented as a work breakdown structure (WBS) showing how major packages of work are progressively broken down into individual activities.

Schedule management runs in parallel with other processes such as scope management, risk management and quality management. In the early parts of the project life cycle the amount of detailed information available is limited. A schedule may be restricted to indicating target dates for major milestones. As these cannot be calculated from a detailed description of work, the milestone dates may initially be estimated using comparative or parametric forms of estimating.

Once scope management has identified the work required, a detailed model can be built showing how the work will be performed. Various methods can be used on the model to calculate the start and finish dates of all the component activities. Some of these methods only consider the estimated time required to perform each piece of work, while others will allow resources and productivity rates to be included.

Traditionally, the schedule concentrates on the technical activity of delivering the project's output. It is becoming increasingly common to include management activity in the schedule, such as communications activity, quality control activity or risk response activity. In some cases these are combined into a complete schedule and sometimes there are separate schedules, although there must be a mechanism for monitoring dependencies between schedules.

Programme: A programme schedule should not be simply an accumulation of detail from the component project schedules. Within a programme there will be:

- Projects that are under way and have their own detailed schedules;
- Projects that are in the early stages of definition;
- Projects that are yet to be initiated;
- Programme management activity;
- Change management activity in relation to outputs that have been delivered;
- Change management activity in relation to outputs being developed.

This variety of types and detail of activity must be collected into an effective and manageable programme schedule.

First of all, the programme must define a consistent approach to scheduling that will be used across the programme. This includes consistent techniques for estimating and calculating schedules, consistent software tools and a consistent approach to summarizing information for inclusion in the programme schedule.

Ideally, the programme schedule will include milestones from the project schedules and the change management activity. In particular it must include interdependencies between different projects and other work.

The programme schedule not only provides estimates of the timing and resource usage of the programme, it must also enable decisions to be made about the acceleration and deceleration of work within the programme. Individual projects will be focused on their designated targets. Sometimes, a programme management team will need to divert resources from one project to another in the overall interests of the programme.

Portfolio: A portfolio schedule must encompass a very wide range of work with greatly varying degrees of detail and accuracy. Portfolio schedules must always be communicated in a way that enables stakeholders to understand what information is derived from detailed schedules and what is more speculative.

An important function of the portfolio management team is a version of capacity planning. Portfolio management must ensure that the necessary resources can be procured to deliver the portfolio. It must also avoid bottlenecks and conflicting demands on limited resources. This means estimating the number of resources required and the timing of their utilization. The balancing phase of the portfolio life cycle must constantly review changing resource demands and prioritize the allocation of limited resources. Doing this effectively will depend upon the schedule information aggregated from the component projects and programmes.

Ideally, the portfolio support function will provide scheduling expertise to all the component projects and programmes and ensure that aggregated information is consistent and accurate.

====

Costs and Cost Management:

Cost management is concerned with the process of planning and controlling the budget of a **project** or business. It includes activities such as **planning, estimating, budgeting, financing, funding, managing**, and **controlling costs** so that the **project** can be completed within the approved budget**. [OR]**

What is Project Cost Management?

Project Cost Management includes the processes required to ensure that the project is completed within the approved budget.

- **Resource Planning**—determining what resources (people, equipment, materials and what quantities of each should be used to perform project activities.
- **Cost Estimating**—developing an approximation (estimate) of the costs of the resources needed to complete project activities.
- **Cost Budgeting**—allocating the overall cost estimate to individual work activities.
- **Cost Control**—controlling changes to the project budget.

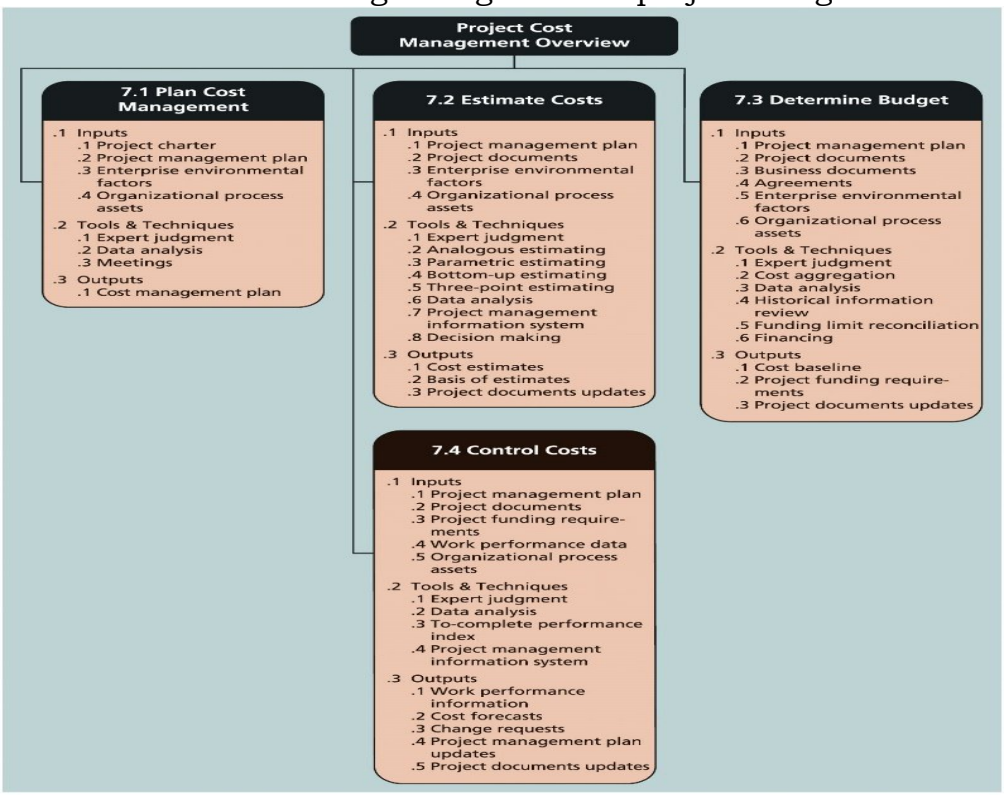


FIGURE 7-1 Project cost management overview

Basic Principles of Cost Management:

- Most members of an executive board better understand and are more interested in financial terms than IT terms; they need to be able to present and discuss project information in both
  - Profits: revenues minus expenditures
  - Profit margin: ratio of profits to revenues
  - Life cycle costing: considers total cost of ownership, or development plus support costs, for a project
  - Cash flow analysis: determines estimated annual costs and benefits for a project and resulting annual cash flow

Types of costs and benefits:

- Tangible costs or benefits are those costs or benefits that an organization can easily measure in dollars

- Intangible costs or benefits are costs or benefits that are difficult to measure in monetary terms
- Direct costs are costs that can be directly related to producing the products and services of the project
- Indirect costs are costs that are not directly related to the products or services of the project, but are indirectly related to performing the project
- Sunk cost is money that has been spent in the past; when deciding what projects to invest in or continue, you should not include sunk costs

Planning Cost Management: The first step in project cost management is planning how the costs will be managed throughout the life of the project.

- The project team uses expert judgment, analytical techniques, and meetings to develop the cost management plan.
- Cost management plan includes:
    - Level of accuracy
    - Units of measure
    - Organizational procedure links
    - Control thresholds
    - Rules of performance measurement
    - Reporting formats
    - Process descriptions

Estimating Costs: Project managers must take cost estimates seriously if they want to complete projects within budget constraints.

- Types of cost estimates
- Tools and techniques for estimating costs
- Typical problems associated with IT cost estimates

Typical Problems with IT Cost Estimates:

Reasons for inaccuracies:

- Estimates are done too quickly
- People lack estimating experience
- Human beings are biased toward underestimation
- Management desires accuracy

****

UNIT-4: SOFTWARE PROJECT EXECUTION, MONITORING AND CONTROL:

Definition of Risk: A risk is a potential problem – it might happen and it might not Conceptual definition of risk.

– Risk concerns future happenings
– Risk involves change in mind, opinion, actions, places, etc.
– Risk involves choice and the uncertainty that choice entails.

Two characteristics of risk:

– **Uncertainty** – the risk may or may not happen, that is, there are no 100% risks (those, instead, are called constraints)

– **Loss** – the risk becomes a reality and unwanted consequences or losses occur

1. Risk Categorization – Approach

Project risks: They threaten the project plan. If they become real, it is likely that the project schedule will slip and that costs will increase

Technical risks: They threaten the quality and timeliness of the software to be produced. If they become real, implementation may become difficult or impossible.

Business risks: They threaten the viability of the software to be built. If they become real, they jeopardize the project or the product Sub-categories of Business risks.

Market risk: Building an excellent product or system that no one really wants.

Strategic risk: Building a product that no longer fits into the overall business strategy for the company.

Sales risk: Building a product that the sales force doesn't understand how to sell.

Management risk: Losing the support of senior management due to a change in focus or a change in people.

Budget risk: Losing budgetary or personnel commitment.

Known risks: Those risks that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date)

Predictable risks: Those risks that are extrapolated from past project experience (e.g., past turnover).

Unpredictable risks: Those risks that can and do occur, but are extremely difficult to identify in advance.

2. Reactive Vs. Proactive Risk Strategies:

Reactive risk strategies:

- "Don't worry, I'll think of something"
- The majority of software teams and managers rely on this approach
- Nothing is done about risks until something goes wrong

The team then flies into action in an attempt to correct the problem rapidly (fire fighting). – Crisis management is the choice of management techniques

Proactive risk strategies:

- Steps for risk management are followed (see next slide)
- Primary objective is to avoid risk and to have a contingency plan in place to handle unavoidable risks in a controlled and effective manner

Steps for Risk Management: Identify possible risks; recognize what can go wrong.

Analyze each risk to estimate the probability that it will occur and the impact (i.e., damage) that it will do if it does occur.

3) Rank the risks by probability and impact: - Impact may be negligible, marginal, critical, and catastrophic.

Develop a contingency plan to manage those risks having high probability and high impact.

Risk Identification: Risk identification is a systematic attempt to specify threats to the project plan.

By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.

Generic risks: – Risks that are a potential threat to every software project.

Product-specific risks:

- Risks that can be identified only by those a with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built
- This requires examination of the project plan and the statement of scope
- "What special characteristics of this product may threaten our project plan?"

Risk Item Checklist: Used as one way to identify risks.

Focuses on known and predictable risks in specific subcategories can be organized in several ways.

- A list of characteristics relevant to each risk subcategory
- Questionnaire that leads to an estimate on the impact of each risk
- A list containing a set of risk component and drivers and their probability of occurrence

Known and Predictable Risk Categories:

- ✓ **Product size** – risks associated with overall size of the software to be built.
- ✓ **Business impact** – risks associated with constraints imposed by management or the marketplace.
- ✓ **Customer characteristics** – risks associated with sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
- ✓ **Process definition** – risks associated with the degree to which the software process has been defined and is followed.
- ✓ **Development environment** – risks associated with availability and quality of the tools to be used to build the project.
- ✓ **Technology to be built** – risks associated with complexity of the system to be built and the "newness" of the technology in the system.
- ✓ **Staff size and experience** – risks associated with overall technical and project experience of the software engineers who will do the work.

Questionnaire on Project Risk:

- ✓ Have top software and customer managers formally committed to support the project?
- ✓ Are end-users enthusiastically committed to the project and the system/product to be built?
- ✓ Are requirements fully understood by the software engineering team and its customers?
- ✓ Have customers been involved fully in the definition of requirements?
- ✓ Do end-users have realistic expectations?
- ✓ Is the project scope stable?
- ✓ Does the software engineering team have the right mix of skills?
- ✓ Are project requirements stable?
- ✓ Does the project team have experience with the technology to be implemented?
- ✓ Is the number of people on the project team adequate to do the job?
- ✓ Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

Risk Components and Drivers:

The project manager identifies the risk drivers that affect the following risk components:

– Performance risk - the degree of uncertainty that the product will meet its requirements and be fit for its intended use.

– Cost risk - the degree of uncertainty that the project budget will be maintained.

– Support risk - the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.

– Schedule risk - the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk driver on the risk component is divided into one of **four impact levels:** – Negligible, marginal, critical, and catastrophic.

Risk drivers can be assessed as impossible, improbable, probable, and frequent.

Risk Projection (Estimation): Risk projection (or estimation) attempts to rate each risk in **two ways:**

- The probability that the risk is real
- The consequence of the problems associated with the risk, should it occur

The project planner, managers, and technical staff perform four risk projection steps. The intent of these steps is to consider risks in a manner that leads to prioritization.

Be prioritizing risks, the software team can allocate limited resources where they will have the most impact.

1. Risk Projection/Estimation Steps

- Establish a scale that reflects the perceived likelihood of a risk (e.g., 1-low, 10-high)
- Delineate the consequences of the risk
- Estimate the impact of the risk on the project and product
- Note the overall accuracy of the risk projection so that there will be no misunderstandings

2. Contents of a Risk Table

A **risk table** provides a project manager with a simple technique for risk projection. It consists of **five columns**:

- Risk Summary – short description of the risk
- Risk Category – one of seven risk categories
- Probability – estimation of risk occurrence based on group input
- Impact – (1) catastrophic (2) critical (3) marginal (4) negligible
- RMMM – Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan

3. Developing a Risk Table

- List all risks in the first column (by way of the help of the risk item checklists)
- Mark the category of each risk
- Estimate the probability of each risk occurring
- Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value (See next slide)
- Sort the rows by probability and impact in descending order
- Draw a horizontal cutoff line in the table that indicates the risks that will be given further attention

4. Assessing Risk Impact

Three factors affect the consequences that are likely if a risk does occur.

– **Its nature** – This indicates the problems that are likely if the risk occurs

– **Its scope** – This combines the severity of the risk (how serious was it) with its overall distribution (how much was affected)

– **Its timing** – This considers when and for how long the impact will be felt

The overall risk exposure formula is RE = P x C.

- P = the probability of occurrence for a risk
- C = the cost to the project should the risk actually occur

Example:

- P = 80% probability that 18 of 60 software components will have to be developed
- C = Total cost of developing 18 components is $25,000
- RE = .80 x $25,000 = $20,000

Risk Mitigation, Monitoring, and Management:

An effective strategy for dealing with risk must consider **three issues** (Note: these are not mutually exclusive).

- Risk mitigation (i.e., avoidance)
- Risk monitoring
- Risk management and contingency planning

Risk mitigation (avoidance) is the primary strategy and is achieved through a plan. Example: Risk of high staff turnover.

Seven Principles of Risk Management

1. **Maintain a global perspective**: View software risks within the context of a system and the business problem that is intended to solve.
2. **Take a forward-looking view:** Think about risks that may arise in the future; establish contingency plans.
3. **Encourage open communication:** Encourage all stakeholders and users to point out risks at any time.
4. **Integrate risk management**: Integrate the consideration of risk into the software process.

5. **Emphasize a continuous process of risk management:** Modify identified risks as more becomes known and add new risks as better insight is achieved.
6. **Develop a shared product vision:** A shared vision by all stakeholders facilitates better risk identification and assessment.

7. **Encourage teamwork when managing risk:** Pool the skills and experience of all stakeholders when conducting risk management activities.

=========

Quality Management: What is quality?

- ✓ Managing the quality of the software process and products. Quality, simplistically, means that a product should meet its specification
- ✓ Concerned with ensuring that the required level of quality is achieved in a software product
- ✓ Involves defining appropriate quality standards and procedures and ensuring that these are followed
- ✓ Should aim to develop a 'quality culture' where quality is seen as everyone's responsibility

The quality compromise:

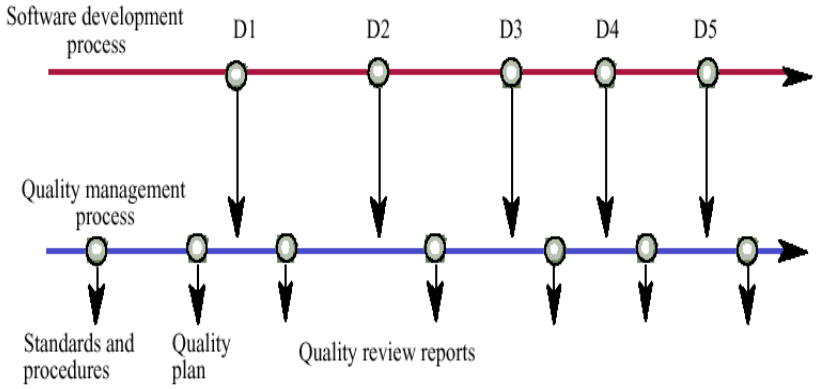
- ✓ We cannot wait for specifications to improve before paying attention to quality management
- ✓ Must put procedures into place to improve quality in spite of imperfect specification
- ✓ Quality management is therefore not just concerned with reducing defects but also with other product qualities

Quality management activities:

- ✓ Quality assurance
  - o Establishing organizational quality standards and procedures
- ✓ Quality planning
  - o Selecting and modifying applicable quality standards and procedures for a particular project
- ✓ Quality control
  - – ensuring quality standards and procedures are followed by development team

Note: Quality management should be separated from project management to ensure independence.

Quality management and software development:



ISO 9000:

- ✓ International set of standards for quality management
- ✓ Applicable to a range of organisations from manufacturing to service industries
- ✓ ISO 9001 applicable to organisations which design, develop and maintain products
- ✓ ISO 9001 is a generic model of the quality process Must be instantiated for each organisation

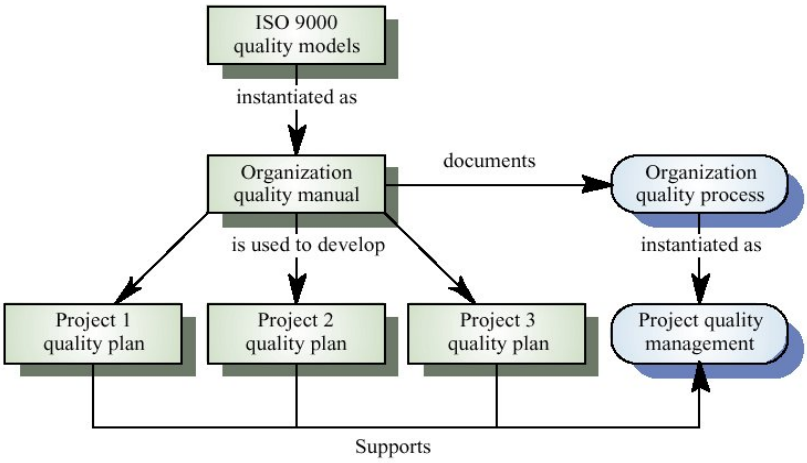
ISO 9001:

Management responsibility	Quality system
Control of non-conforming products	Design control
Handling, storage, packaging and delivery	Purchasing
Purchaser-supplied products	Product identification and traceability
Process control	Inspection and testing
Inspection and test equipment	Inspection and test status
Contract review	Corrective action
Document control	Quality records
Internal quality audits	Training
Servicing	Statistical techniques

ISO 9000 certification:

- ✓ Quality standards and procedures should be documented in an organisational quality manual
- ✓ External body may certify that an organisation's quality manual conforms to ISO 9000 standards
- ✓ Customers are, increasingly, demanding that suppliers are ISO 9000 certified

ISO 9000 and quality management:



Quality assurance and standards:

- ✓ Standards are the key to effective quality management
- ✓ They may be international, national, organizational or project standards
- ✓ Product standards define characteristics that all components should exhibit e.g. a common programming style
- ✓ Process standards define how the software process should be enacted

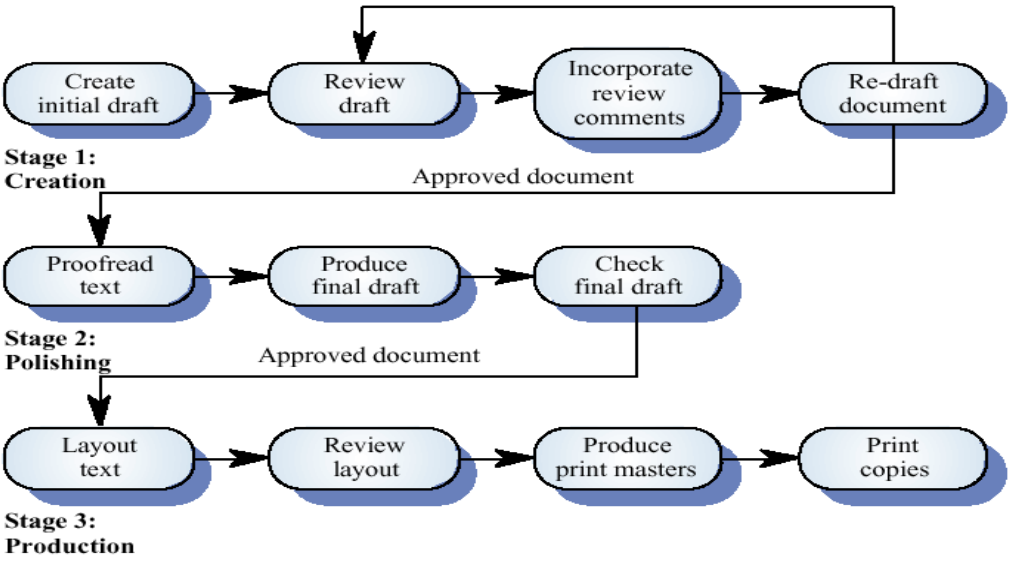
Standards development:

- ✓ Involve practitioners in development. Engineers should understand the rationale underlying a standard
- ✓ Review standards and their usage regularly. Standards can quickly become outdated and this reduces their credibility amongst practitioners
- ✓ Detailed standards should have associated tool support. Excessive clerical work is the most significant complaint against standards

Documentation standards:

- ✓ Particularly important - documents are the tangible manifestation of the software
- ✓ Documentation process standards
  - o How documents should be developed, validated and maintained
- ✓ Document standards
  - o Concerned with document contents, structure, and appearance
- ✓ Document interchange standards
  - o How documents are stored and interchanged between different documentation systems

Documentation process:



Document standards:

- ✓ Document identification standards
  - How documents are uniquely identified
- ✓ Document structure standards
  - Standard structure for project documents
- ✓ Document presentation standards
  - Define fonts and styles, use of logos, etc.
- ✓ Document update standards
  - Define how changes from previous versions are reflected in a document

Document interchange standards:

- ✓ Documents are produced using different systems and on different computers
- ✓ Interchange standards allow electronic documents to be exchanged, mailed, etc.
- ✓ Need for archiving. The lifetime of word processing systems may be much less than the lifetime of the software being documented
- ✓ XML is an emerging standard for document interchange which will be widely supported in future

Process and product quality:

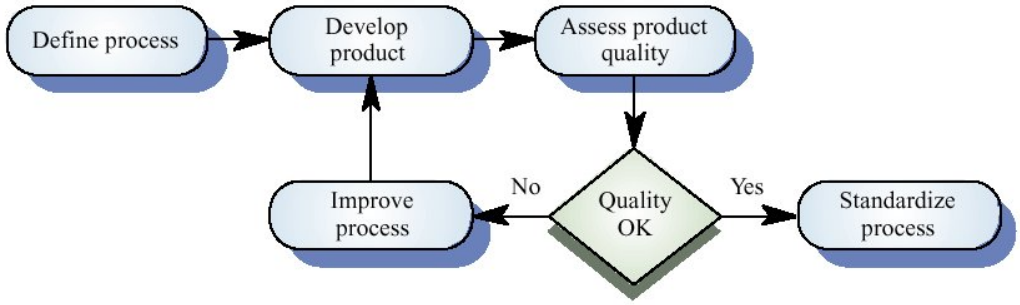
- ✓ The quality of a developed product is influenced by the quality of the production process
- ✓ Particularly important in software development as some product quality attributes are hard to assess
- ✓ However, there is a very complex and poorly understood between software processes and product quality

Process-based quality:

- ✓ Straightforward link between process and product in manufactured goods
- ✓ More complex for software because:

- The application of individual skills and experience is particularly important in software development
- External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality

✓ Care must be taken not to impose inappropriate process standards



Practical process quality:

✓ Define process standards such as how reviews should be conducted, configuration management, etc.

✓ Monitor the development process to ensure that standards are being followed

✓ Report on the process to project management and software procurer

Quality planning:

✓ A quality plan sets out the desired product qualities and how these are assessed and define the most significant quality attributes

✓ It should define the quality assessment process

✓ It should set out which organisational standards should be applied and, if necessary, define new standards

Quality plan structure:

✓ Product introduction

✓ Product plans

✓ Process descriptions

✓ Quality goals

✓ Risks and risk management

✓ Quality plans should be short, succinct documents

- If they are too long, no-one will read them

Software quality attributes:

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Quality control:

✓ Checking the software development process to ensure that procedures and standards are being followed

✓ Two approaches to quality control

- Quality reviews
- Automated software assessment and software measurement

Quality reviews:

- ✓ The principal method of validating the quality of a process or of a product
- ✓ Group examined part or all of a process or system and its documentation to find potential problems
- ✓ There are different types of review with different objectives
  - Inspections for defect removal (product)
  - Reviews for progress assessment(product and process)
  - Quality reviews (product and standards)

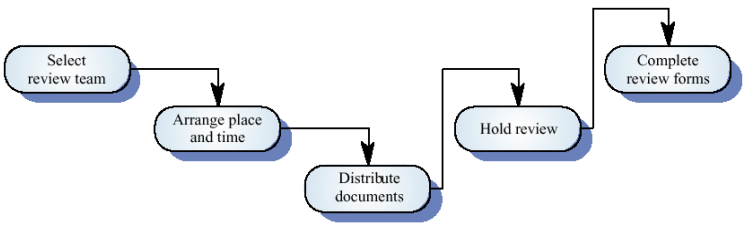
Types of review:

Review type	Principal purpose
Design or program inspections	To detect detailed errors in the design or code and to check whether standards have been followed. The review should be driven by a checklist of possible errors.
Progress reviews	To provide information for management about the overall progress of the project. This is both a process and a product review and is concerned with costs, plans and schedules.
Quality reviews	To carry out a technical analysis of product components or documentation to find faults or mismatches between the specification and the design, code or documentation. It may also be concerned with broader quality issues such as adherence to standards and other quality attributes.

Quality reviews:

- ✓ A group of people carefully examine part or all of a software system and its associated documentation.
- ✓ Code, designs, specifications, test plans, standards, etc. can all be reviewed.
- ✓ Software or documents may be 'signed off' at a review which signifies that progress to the next development stage has been approved by management.

The review process:



Review functions:

- ✓ Quality function - they are part of the general quality management process
- ✓ Project management function - they provide information for project managers
- ✓ Training and communication function - product knowledge is passed between development team members

Quality reviews:

- ✓ Objective is the discovery of system defects and inconsistencies
- ✓ Any documents produced in the process may be reviewed
- ✓ Review teams should be relatively small and reviews should be fairly short
- ✓ Review should be recorded and records maintained

Review results:

- ✓ Comments made during the review should be classified.
  - o No action. No change to the software or documentation is required.
  - o Refer for repair. Designer or programmer should correct an identified fault.
  - o Reconsider overall design. The problem identified in the review impacts other parts of the design. Some overall judgement must be made about the most cost-effective way of solving the problem.
- ✓ Requirements and specification errors may have to be referred to the client.

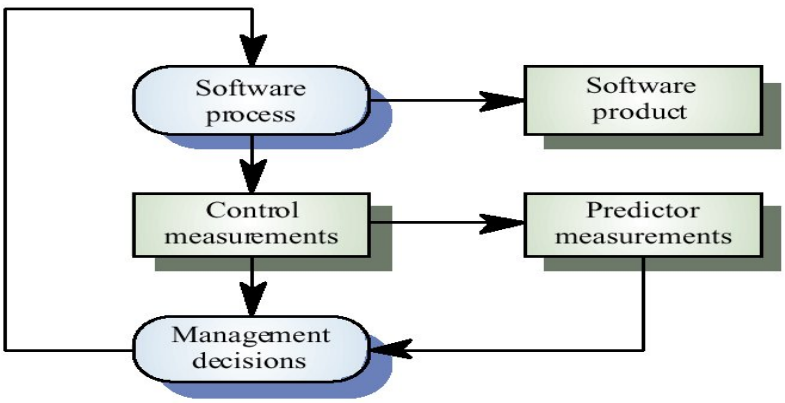
Software measurement and metrics:

- ✓ Software measurement is concerned with deriving a numeric value for an attribute of a software product or process
- ✓ This allows for objective comparisons between techniques and processes
- ✓ Although some companies have introduced measurement programmes, the systematic use of measurement is still uncommon
- ✓ There are few standards in this area

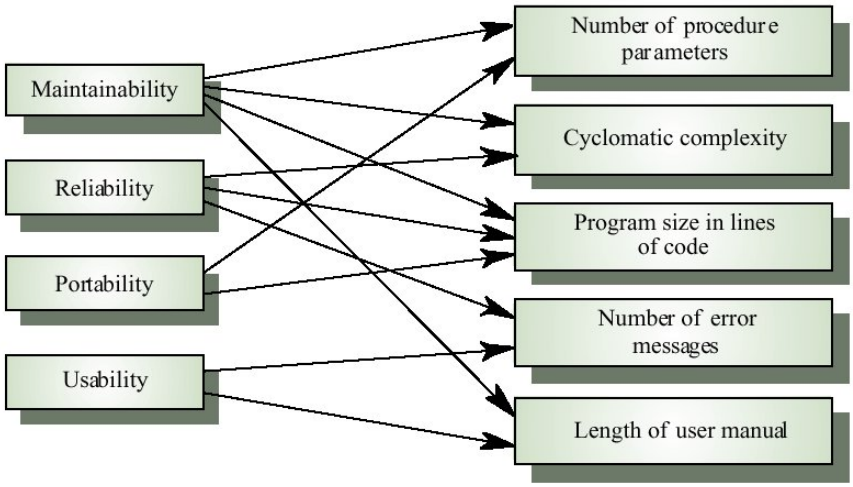
Software metric:

- ✓ Any type of measurement which relates to a software system, process or related documentation
  - o Lines of code in a program, the Fog index, number of person-days required to develop a component
- ✓ Allow the software and the software process to be quantified
- ✓ Measures of the software process or product
- ✓ May be used to predict product attributes or to control the software process

Predictor and control metrics:



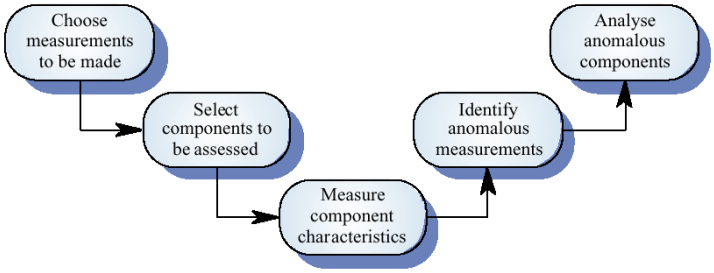
Internal and external attributes:



The measurement process:

- ✓ A software measurement process may be part of a quality control process
- ✓ Data collected during this process should be maintained as an organisational resource
- ✓ Once a measurement database has been established, comparisons across projects become possible

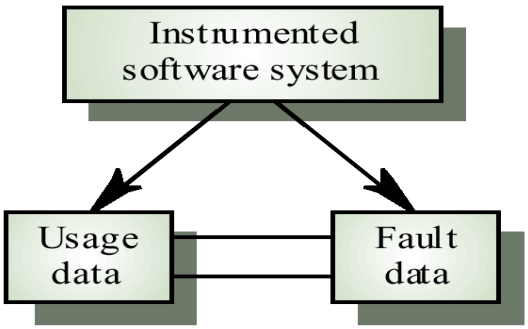
Product measurement process:



Data collection:

- ✓ A metrics programme should be based on a set of product and process data
- ✓ Data should be collected immediately (not in retrospect) and, if possible, automatically
- ✓ Three types of automatic data collection
  - o Static product analysis
  - o Dynamic product analysis
  - o Process data collation

Automated data collection:



Data accuracy:

- ✓ Don't collect unnecessary data
  - The questions to be answered should be decided in advance and the required data identified
- ✓ Tell people why the data is being collected
  - It should not be part of personnel evaluation
- ✓ Don't rely on memory
  - Collect data when it is generated not after a project has finished

Software product metrics:

Software metric	Description
Fan in/Fan-out	Fan-in is a measure of the number of functions that call some other function (say X). Fan-out is the number of functions which are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.
Length of code	This is a measure of the size of a program. Generally, the larger the size of the code of a program's components, the more complex and error-prone that component is likely to be.
Cyclomatic complexity	This is a measure of the control complexity of a program. This control complexity may be related to program understandability. The computation of cyclomatic complexity is covered in Chapter 20.
Length of identifiers	This is a measure of the average length of distinct identifiers in a program. The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the depth of nesting of if-statements in a program. Deeply nested if statements are hard to understand and are potentially error-prone.
Fog index	This is a measure of the average length of words and sentences in documents. The higher the value for the Fog index, the more difficult the document may be to understand.

****

UNIT-5: PROJECT CLOSURE AND MAINTENANCE:

Project Implementation: Introduction:

In this last chapter we will focus on implementation of a project. The Project implementation focuses on installing or delivering the project's major deliverables in the organization. Implementation is the stage where all the planned activities are put into action. Examples for information system project implementation would be the installation of new databases and application programs, and the adoption of new manual procedures etc.

In general, implementing the product of an IT project can follow one of the four approaches. These approaches are direct cutover, parallel, phased or pilot. Each approach has unique advantages and disadvantages that make a particular approach appropriate for a given situation.

As you know a project has a definite beginning and a definite end. Once the project is implemented, the project manager and his team must prepare for closing the project. A project is properly closed for two reasons. Firstly, there is a tendency for projects to drift on and become, or develop into, other projects. Secondly, it is important to ensure that the work of the project team is acknowledged and that the lessons to be learned from the project are formally investigated and recorded for use on the next project.

Once the project is closed, the project manager should evaluate each project team member individually in order to provide feedback to the individual about his performance on the project. In addition, the project should be reviewed by an impartial outside party. An audit or outside review can provide valuable insight on how well the project was managed and on how well the project members functioned as a team.

The project's overall goal was defined as the MOV, or Measurable Organizational Value. It is clearly defined and agreed upon in the early stages of the project that whether the project is successful, as defined by its MOV.

Project Implementation:

After developing the project, the IS is transferred successfully from the development and test environment to the operational environment of the customer. Choosing an inappropriate implementation approach can negatively impact the project's remaining schedule and budget. In general, the project team can take one of three approaches for implementing the IS.

These approaches are (i) Direct Cutover (ii) Parallel (iii) Pilot and (iv) Phased.

i. Direct Cutover:

The direct cutover approach, as illustrated in below figure produces the changeover from old system to the new system instantly. This approach can be effective when quick delivery of the new system is critical and this approach

may also be appropriate when the system's failure will not have a major impact on the organization i.e., the system is not mission critical.

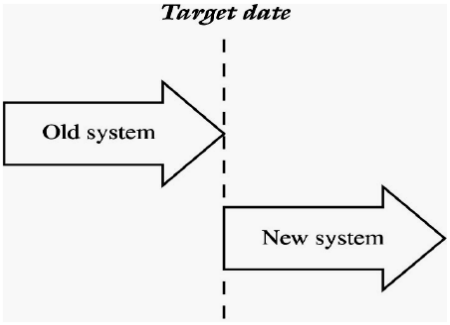


Figure: Direct Cutover.

Companies often choose the direct cutover approach for implementing commercial software package. Although there are some advantages to using the approach, there are also a number of risks involved that generally make this the least favored approach. Using this approach you may think as walking a tightrope without a safety net. You may get from one end of the tightrope to other quickly, but not without a great deal of risk. Subsequently, there is no going back to the old system to the new system. As a result, the organization could experience major delays, lost revenues and missed deadlines. The pressure of assuring that everything is right can create a great deal of stress for the project team.

ii. Parallel:

Parallel approach as shown in below figure is the method in which both the new system and the old system will operate at the same time, for a specified period of time, in order to check the new system for complexities.

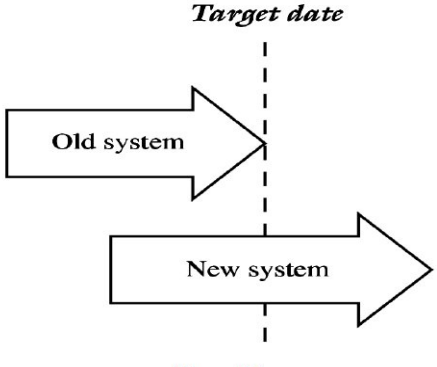


Figure: Parallel.

Data is input in both system and the results are verified. This approach is impractical if the systems are dissimilar or does not support each other. The cost using this approach is relatively high, because both systems are operating requiring more man power in terms of management. Using this approach provides confidence that the new system is functioning and performing properly before relying on it entirely. It is also impractical to use this approach as the new system and old system technically incompatible.

iii. Pilot:

It is the combination of both direct cutover and parallel approach. The pilot method involves implementing the new system at a selected location like a branch office, one department in a company, etc. called pilot site, and the old system continues to operate for the entire organization.

Risk and cost, associated in this method are relatively less, because only one location runs the system and the new system is only installed and implemented at pilot sites; reducing the risk of failure. After the new system proves that the system is successfully at the pilot site, it is implementing in the rest of the organization, usually using the direct cutover method.

iv. Phased:

The Phased approach allows implementing the new system in phases or modules or stages in different parts of the organization incrementally as shown in the below figure. E.g., an organization may implement an accounting information system package by first implementing the general ledger component, then accounts payable etc.

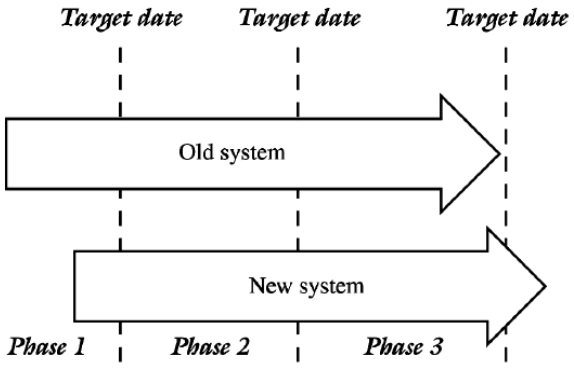


Figure: Phased.

This method is one of the least risky because implementation only takes effect in part, in case an error goes wrong with the new system, only that particular affected part is at risk. A phased approach may also allow the project team to learn form its experiences during the initial implementation so that the later implementations run smoothly.

Although the phased approach may take more time than the direct cutover approach, it may be less risky and much manageable. After all the modules have been tested independently it is possible to implement the new system in the organization, which would be error free. Also, overly optimistic target dates or problems experienced during the early phases of implementation may create a chain reaction that pushes back the scheduled dates of the remaining planned implantations.

===

Software Maintenance:

Software maintenance is an easy task and requires less effort than actual software development. If change requests are made toward the end of the project, then maintenance activities can contribute to large costs and effort overruns. Moreover, contrary to the popular view, implementing changes in the software product in the maintenance stage is a painstaking task.

According to surveys, changeability is one of the most important attributes of complex, multifunction software systems. This calls for continuous adjustments of the software to suit the changing environment. While changes are inevitable, it is not always possible to create products foreseeing the changes. Therefore, the product needs to be changed according to the changing scenario. You might need to upgrade the software product based on new system requirements or remove redundant functionality. The need for a change brings software maintenance into the picture. Maintenance activities can either be in-house or outsourced. In the case of in-house maintenance, the development team performs the maintenance activities. On the other hand, an independent maintenance team performs outsourced maintenance activities.

Software maintenance deals with the sustenance of a piece of software after it is released. Maintenance activity is needed when errors are detected after the software product is released. Changing requirements of the software product also require maintenance activity to cater to the changes. Maintenance activity assumes greater importance for a project manager. This is because managing software maintenance is more challenging than managing software development. Consider an example. You are maintaining a banking operation. A small Program change is introduced as part of the maintenance activity. Suppose this change results in an incorrect interest calculation. You can well imagine the kind of confusion this error will create, not to mention the loss of client faith and goodwill in the bank. Therefore, as a project manager, you need to be extra cautious while managing a maintenance project.

Types of Maintenance Activities:

Maintenance is an important phase in SDLC: The need for maintenance activities has increased with the increase in software packages. Maintenance activities include correction and prevention of defects, enhancements to incorporate changing needs, and porting of applications and adaptability. The changing hardware and software scenarios have increased the demand for software maintenance. Today, in the entire SDLC of a software application, the maintenance effort is about 80 percent and the development effort is only 20 percent. There are four types of maintenance activities:

I. Corrective
II. Adaptive
III. Perfective
IV. Preventive

I. Corrective maintenance is about fixing bugs. This takes approximately 17 percent of the maintenance time. During corrective maintenance, the existing

code is used to correct the fault that causes the code to deviate from its documented requirements. Here, the focus is on fixing defects.

II. During adaptive maintenance, the existing code is changed to adapt the new features and functionality. These new features are usually part of a new release of a code. This change normally takes 18 percent of the maintenance time. Perfective maintenance improves the maintainability of the code. During this activity, the code is restructured to make it easily understood and to remove ambiguities. The enhancement of code occupies 60 percent of the maintenance time.

III. Preventive maintenance is undertaken to protect the code against failure. Here, the focus is on adhering to coding standards and reduce the chances of code failure. Preventive maintenance .activity takes around 5 percent of the maintenance time. In the above list of maintenance activities, you can see that out of total maintenance effort, only 20 percent is spent on corrective maintenance and 80 percent on the rest. This refutes the popular belief that maintenance activities is all about fixing mistakes.

IV. Maintenance Activities:

To perform maintenance activities, the maintenance team needs to acquire the business and technical knowledge of the client systems. The maintenance process is usually divided into three phases:

a. Initiation Phase
b. Preparation Phase
c. Execution Phase

a. Initiation Phase:

The initiation phase starts with the commencement of maintenance activities. It is primarily a knowledge acquisition phase. During this phase, the maintenance team takes over the system from the development team. The maintenance team familiarizes itself with the ways and functioning of the system. This is essentially a phase of knowledge transfer from the development team to the maintenance team. The major activities of the initiation phase are baseline assessment and operating procedures. During baseline assessment, the interfacing and communication methods between the development and maintenance teams are defined. The user-coordinator from the user side and the information technology help from the maintenance team are identified. During this phase, the plan for acquiring the systems knowledge is also finalized. During the operating procedures activities, the maintenance team understands the application and the functionality involved in the maintenance task.

The maintenance team reviews the current operating procedures of the system. The team also obtains details about the environment set up, deployment of software, the resource allocation process, and priority settings. Other' activities such as root cause analysis, work assignment to the development teams, and updating of documentation procedures is also done. The maintenance team refers to the problem management procedures, on-call or escalation

procedures; and operational process for handling: faults. It also makes decisions based on the severity of the faults, application criticality, and infrastructure issues. The maintenance team finalizes procedures for transferring support work from the development team. The maintenance team members are initiated to hands-on exposure of the systems. The maintenance team finalizes the quality systems and standards and guidelines to be followed.

b. Preparation Phase:

The preparation phase begins after the initiation phase. During this phase, the maintenance team sets up the administrative and support procedures needed for maintenance activities. In this phase, the environment is set up and hardware, software, and network are made ready for operation. All the procedures decided and finalized during the initiation phase are put into practice in this phase. According to the documentation procedures finalized in the initiation phase, the latest documentation is kept in the library. During this phase, bug reporting, problem solving, and other support methodologies are established. The security measures are also put in place and practiced during this phase.

c. Execution Phase:

The execution phase follows the preparation phase. In this phase, the maintenance team starts executing the maintenance activities after the preparation in the earlier phase. During this phase, the maintenance team is in close touch with the development team for a small period of time. Such communication is recommended to ensure better coordination.

Maintenance Curve:

Maintenance activity begins after product implementation. The effort spent on maintenance varies over a period of time. Normally, maintenance effort is very high immediately after implementation. The system faces numerous changes during this time. The changes could be in the form of bugs, changes in network environment, or problems caused by user mishandling. During this time, the effort spent on maintenance activities is very high. With time, the system stabilizes and maintenance activities also slowdown. During this period, the users are more or less satisfied with the system. The system also runs smoothly without requiring any major changes. Therefore, maintenance activity is at its lowest during this phase.

However, over a period of time, business processes change and the system also depreciates. During this period, major changes start coming in. The changes are in the form of enhancements and additions of new systems or modules. These changes take place continuously to keep the system up-to-date with the changing needs and requirements. Maintenance activities are also at a maximum during this phase. However, soon the system becomes obsolete and is phased out. Along with the system phase- out, maintenance activities also come to an end. Below figure shows the variation of maintenance activity with time. In the curve, maintenance effort is very high during the initial phase. Then the maintenance effort stabilize during the maturity phase. Finally, the

maintenance effort increases sharply just before system phase out. The system is phased out to pave the way for new systems.

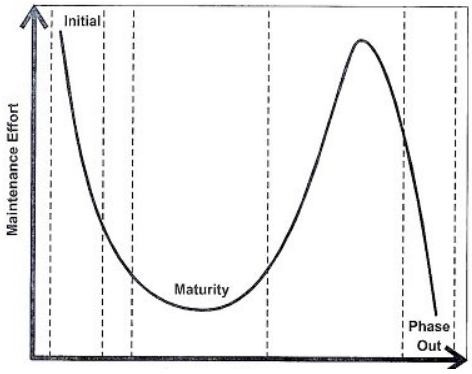


Figure. Maintenance Curve.

Maintenance Process:

There are certain prerequisites that you need to follow before beginning with the maintenance process. All documentation needs to be complete and up-to-date. All existing items need to be configured, and the latest copy of the code should be available. The hardware, software, and network environment for testing and maintenance should be made available. Before beginning with the maintenance process, the maintenance team is identified. The user representative from the user side is identified and the maintenance team is constituted. The maintenance team consists of the head of IT, the project coordinator, and team members. The maintenance project 1anager is the single point contact between the user representative and the maintenance team. The project manager categorizes the type of maintenance activity to be performed based on specifications. The project manager ensures proper communication between all the entities 1volved in maintenance activity. To ensure that the maintenance process runs smoothly, there is a thorough checking of all deliverables. Maintenance activity is the most critical because the system is running and already in place. Therefore, the project manager needs to ensure that no error arises during the maintenance process.

The common approach followed in the maintenance process is as follows. After the user makes a request for change, the maintenance team performs an impact analysis. Then the team plans for the system release. The system goes through design changes, coding, and testing. Finally, the system is released. The maintenance activities undertaken are corrective maintenance, adaptive maintenance, perfective maintenance, and preventive maintenance.

Corrective Maintenance Process:

The corrective maintenance process begins with the user representative reporting bugs to the project coordinator by using the problem report form. The project coordinator analyzes the problem and assigns the problem to one of the team members for corrective action. If the bug fixing leads to any changes in the system, the change is carried out based on the change request process. The team member requests the SCM team to check out the item to be modified. Then the bugs are fixed and tested in the test environment. After

this, the user acceptance is carried out for the successful completion of the corrective maintenance process. Finally, the SCM team releases the modified item. All related documents are modified to reflect the changes and the project coordinator closes the problem report.

Adaptive Maintenance Process:

In the adaptive maintenance process, changes are made to the system to accommodate changes in its external environment. Adaptive maintenance includes work related to enhancing software functionality. During this process, certain system changes, additions, insertions, deletions, and modifications are included to meet the evolving needs of the user and the environment. At times, the original environment of the software product might change. For example, there might be changes in operating system, business rules, and external product characteristics. Adaptive maintenance refers to modifications that adapt to these changes. The project coordinator plans the changes to be made. The detail plan and the quality goals are defined for the changes. The team is formed based on the quantum of work. The change request procedure is followed to carry out the changes. Finally, the changes are deployed based on a deployment strategy.

Perfective Maintenance Process:

Perfective maintenance refers to enhancements that make the product better, faster, smaller, better documented, better structured with more functions and reports. It includes all efforts to improve the quality of software. Perfective maintenance extends the software beyond its original functional requirements. During perfective maintenance, the user or the system personnel initiates the changes to the system. The initiator raises the change request to the project coordinator. The changes could be a result of enhancements of new functionality in the existing system.

Preventive Maintenance Process:

Preventive maintenance activities make changes to software programs so that they can be more easily corrected, adapted, and enhanced. During the preventive maintenance process, the project coordinator identifies the potential risks. Recommendations are made to the management for approval of the maintenance actions. The project coordinator prepares the project detail plan for carrying out the changes. The preventive maintenance team is formed based on the quantum of work. The team implements the change based on the request procedure. Finally, deployment happens based on the deployment strategy. The maintenance process begins when the user initiates a request for a change. The process ends when the system changes are tested and the change is released for operation. In between, there are many activities that are coordinated and planned using change management.

Re-Engineering:

Re-engineering is a process of abstraction of a new system from an old system. Re-engineering is the modification and evolution of software product to meet

the constantly changing business requirements. Failure to reengineer can prevent an organization from remaining competitive and persistent. This is because every system faces the risk of being phased out sooner than they were earlier. Therefore, you need to develop systems that support multiple platforms and are open to modifications and changes. Here, the role of reengineering comes into play. Re-engineering comprises reverse engineering and forward engineering.

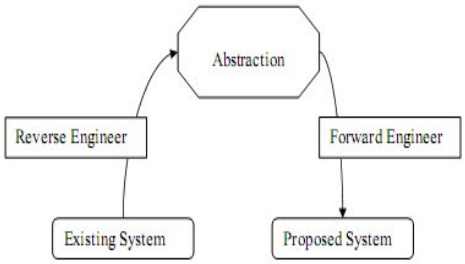


Figure: Displays the Re-Engineering process.

Reverse Engineering:

Reverse engineering is a process of recovering the design of the running system from the source code. Reverse engineering involves analyzing a software system to achieve two objectives. First, reverse engineering helps to identify the system components and their Interactions. Second, it helps make representations of the system on a different, higher level of abstraction.

Forward Engineering:

Forward engineering is the process of building a new program from the requirements and design specifications of an old program. During forward engineering, old design information is recovered from existing software and is used to reconstitute the existing system. In the process, the quality of the existing system is enhanced and new functions are added. Therefore, forward engineering moves from a high-level abstraction and design to a low-level implementation. Forward engineering is used to improve the overall performance of the system.

===

Software Project Closure: Administrative Closure:

Although all projects come to an end, a project can be terminated for any number of reasons. Gray and Larson (2000) define five circumstances for ending a project: normal, premature, perpetual, failed and changed priorities.

 Normal – A project that completed as planned. The project scope is achieved within cost, quality and schedule objectives with some modification and variation along the way. The project is transferred to the project sponsor and the end of the project is marked with a celebration, awards for a good job.

 Premature – A project team may be pushed to complete a project early even though the system may not include all of the envisioned features or functionality.

 Perpetual – These projects never seem to end. These projects may result from delays or a scope or MOV that was never clearly defined. Then the project sponsor may attempt to add on various features to the system, which results in
added time and resources that increase the project schedule and drain the project budget. Attention to defining and agreeing to the project's MOV, the project scope processes, and timely project reviews can reduce the risk of perpetual projects.

 Failed – Sometimes projects are just unsuccessful. In general an IT project fails if insufficient attention is paid to the people, processes or technology.

 Changed Priority – In some cases, a project may be terminated as a result of a change in priorities. Financial or economic reasons may dictate that resources are no longer available to the project. This change happens when the original importance of the project was misrepresented. Ideally a project is closed under normal circumstances. Unfortunately, closing a project does not often happen. As J. Davidson Frame (1998) points out, the project manager and team should be prepared to deal with the following realities:

 Team members are concerned about future jobs. Often the team members of the project team are borrowed from different departments. Once the project is finished, they will return to their previous jobs. Regardless as the project nears its end, these project team members may begin to wonder what they will do next. For some, there will be rewarding life after the project and for some other looking for the new job. As a result, the project team members may not focus on what has to be done to close the project, and wrapping up the project may be a challenge.

 Bugs still exist. Software quality testing may not find all the defects, and certain bugs may not become known until after the system has been implemented. Unless these defects and bugs are promptly addressed and fixed, the project sponsor's satisfaction with the project team and the information system may become an issue.

 Resources are running out. Resources and the project schedule are consumed from the project's earliest inception. At the end of the project, both resources and time remaining are usually exhausted. So the project manager may find adequate resources to deal with problems effectively are not
available.

 Documentation attains paramount importance. The IT projects require system, training and user documentation. Many times, however documentation is put off until the end of the project. As a result, documentation becomes increasingly important at the end of the project and may require more time and resources to complete.

 Promised delivery dates may not be met. Most projects experience schedule slippage due to poor project management, implementation risks, and overly optimistic estimates. Any misjudgment concerning what has to be done,

what is needed to complete the job and how long will it take will result in a variance between the planned and actual schedule and budget.

 The players may posses a sense of panic. The project stakeholders may experience panic as schedule begins to slip and the resources become exhausted. The sponsor may worry that the IS will not be delivered on time and within the budget. As the sense of panic increases, the chances for an orderly closeout grow dim.

A good closeout allows the team to wrap up the project in a neat, logical manner. From an administrative view, this procedure allows for all loose ends to be tied up. From a psychological perspective, it provides all of the project stakeholders with a sense that the project was under control form the beginning through to its end (Frame 1998)

i. Project Sponsor Acceptance:

The most important requirement for closure under normal circumstances is obtaining the project sponsor's acceptance of the project. Since acceptance depends heavily on the fulfillment of the project's scope, the project manager becomes responsible for demonstrating that all project deliverables have been completed according to the specification.

Rosenau (1998) observes that there are two basic types of project sponsors. Shortsighted sponsors and knowledgeable sponsors. Shortsighted sponsors – view the project as a short-term buyer-seller relationship in which getting the most for their money is the most important criteria for accepting the project.
Knowledgeable sponsors – they have an important stake in the outcome of the project. They will be actively involved throughout the project in a constructive manner.

Regardless of whether the sponsor is shortsighted or knowledgeable, the project manager and team can improve the likelihood that the project will be accepted if they:

(i) Acceptance criteria clearly defined in the early stages of project.
(ii) Completion of all project deliverables and milestones thoroughly documented.

A clearly definition of the project deliverables is an important concern for project scope management. Defining and verifying that the project scope and system requirements are accurate and complete is only one component. Project milestones ensure that the deliverables are complete. Documenting each deliverables and milestone throughout the project provides confidence to the project sponsor that the project has been completed fully.

ii. The Final Project Report:

The project manager and team should develop a final report and presentation for the project sponsor and other key stakeholders to ensure that the project has been completed as outlined in the business case, project charter and

project plan. The report may be circulated to key stakeholders before the presentation in order to get feedback and to identify any open items that need to be scheduled for completion. Once finalized, the final project report provides a background and history of the project. The report should include:

(i) Project summary
- o Project Description
- o Project MOV
- o Scope, Schedule, Budget and Quality Objectives

(ii) Comparison of Planned versus Actual
- o Original Scope and history of any approved changes
- o Original scheduled deadline versus actual completion date
- o Original budget versus actual cost of completing the project
- o Test plans and test results

(iii) Outstanding Issues
- o Itemized list and expected completion
- o Any ongoing support required and duration

(iv) Project Documentation List
- o Systems Documentation
- o User Manuals
- o Training Materials
- o Maintenance Documentation

iii. The Final Meeting and Presentation:

If the project has been diligent in gaining the confidence of the project sponsor, the final meeting and presentation should be simple, straightforward. Buttrick (2000) suggests that the final meeting is useful for:

 Communicating that the project is over. By inviting key stakeholders to the meeting, the project manager is formally announcing that the project is over.

 Transferring the information system from the project team to the organization. Although the system may have been implemented and is being used by the organization, the final meeting provides a formal exchange of the project's product from the project team to the organization.

 Acknowledging contribution. The meeting provides a forum for the project manager to acknowledge the hard work and contributions of the project team and other key stakeholders.

 Getting formal signoff. The meeting can provide a ceremony for the sponsor or client to formally accept the IS by signing off on the project.

iv. Closing the Project:

After the project is accepted by the client, a number of administrative closure processes remain. Administrative closure is a necessity because once the project manager and team are officially released from the current project,

getting them to wrap up the last of the details will be difficult. The requirements for administrative closure include:

- Verifying that all deliverables and open items are complete.
- Verifying the project sponsor or customer's formal acceptance of the project.
- Organizing and archiving all project deliverables and documentation.
- Planning for the release of all project resources
- Planning for the evaluation and reviews of the project team members and the project itself
- Closing of all project accounts
- Planning a celebration to mark the end of a project.

====== All the Best ======