Lecture Notes
Unit 1
Lecture 1: Fundamentals of Software Project Management

-Definition of SPM:
-Project manager:
-Skill of the project manager
-Role of the project manager
-Responsibility of the project manager
-Problem faced by project manager
-Project interfaces-
-Stakeholder's analysis
-type
-advantages
-risk
-Methods

Lecture 2: Need Identification & Establishing requirements

-Introduction

Identify the causes of failure of project……

-Need of requirements

Verity of purpose is included

- Project scoping
- Cost estimating
- Budgeting
- Project scheduling
- Software design
- Software testing
- Documentation and training manuals

-classifying and documenting requirements

We typically capture requirement in three separate documents-

a) **Documenting Stakeholders need**
Stakeholders needs are the part of problem domain, describes what stakeholders require for successful project.

ID	Need	% of responses
N1	Need to notify……..	
N2	Need to assign………………..	

Corresponding features mapped to these needs:

ID	Need	% of response	Map to
F1			N1, N2
F2			N2 only

b) **Documenting Software features**
Features are the part of solution domain
c) **Documenting software requirements**
Requirement must satisfy one or more of the following criteria
   - Contract obligation
   - Standards
   - Desired needs and features

-Requirement analysis process

a) **Importance of requirements analysis**

Explain why requirement analysis is necessary?

b) **Steps in requirement process**

- Fixed system boundaries
- Identify the customers
- Requirement elicitation
  What their requirements from the application are and what the expect the application to do. The level of detail of the requirements list is based on the number and size of user group, the degree of complexity of business processes and size of the application.

c) **Problem faced in requirements elicitation**

- Ambiguous understanding of process
- Inconsistency with in a single process by multiple user
- Insufficient input from stakeholder
- Conflicting stakeholders interest
- Change in requirements after project has begun

d) **Tool used in requirement elicitation**

- Prototypes
- Use case
- DFD
- Transition process diagram
- User interface

e) **Requirement analysis process**

Some technique require for Requirement analysis process

- Animation
- Automated reasoning
- Knowledge based critiquing
- Consistency checking
- Analogical and case-based reasoning

f) Requirement specification

SRS is a document that list out stakeholders' need and communicate these to the technical community that will design and build the system.

SRS documented separately as user requirements and system requirements and serve as a starting point for software, hardware and database design.

Lecture 3: Vision and Scope document

Documents that covers the business requirements is called Vision and Scope document. Vision of the project covers long run what the project will achieve and scope part covers what would be taken up.

1. **Problem statements/business requirements**
   - Project background, business opportunity and customer's need
   - Business objectives and success criteria
   - Business Risk
2. **Vision of the solution**
   - Vision statements
   - List of features
   - Assumptions and dependencies

3. **Scope and limitations**
   - Scope of the phase released
   - Features that will not be developed

4. **Business context**
   - Stakeholder profile
   - Project priorities

Case study:

<u>Business requirements:</u>

Background, business opportunity and customer needs:

Business objectives and success criteria:

Business objectives:

1) Business objectives 1:
2) Business objectives 2:
3) Business objectives 3:

Success criteria:

1) success criteria 1:
2) success criteria 2:
3) success criteria 3:

Business Risk:

1) Risk1
2) Risk2
3) Risk3

Vision of the solution:

Vision statements:

List of features:

1) Feature 1:
2) Feature 2:
3) Feature 3:
4) Feature 4:
5) Feature 5:
6) Feature 6:
7) Feature 7:
8) Feature 8:
9) Feature 9:

Assumptions and dependencies:

1) Assumptions 1:
2) Assumptions 2:

Dependencies 1:

Scope and limitations:

Scope of initial and subsequent releases:

Features	Release 1	Release 2	Release 3
FE 1			
FE 2	Not implemented	Not implemented	Fully implemented
FE 3			
FE 4			
FE 5			

FE 6			
FE 7			
FE 8			

Limitations:

1) Limitations1:
2) Limitations 2:

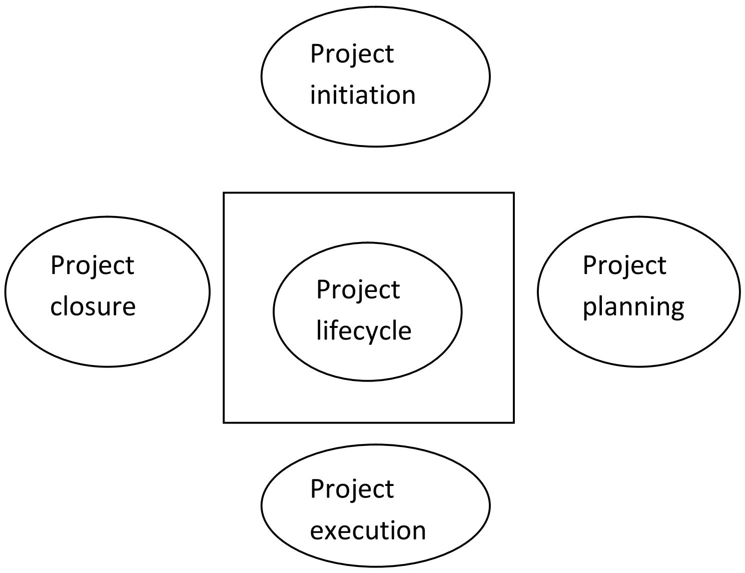
Business context:

Stakeholder profile:

Stakeholders	**Major value**	**Attribute**	**Major interest**	**Constraint**
Corporate				
Management cafeteria				
Staff customers				
Restaurant manager				

Lecture 4: Project management lifecycle

a) Project initiation phase
b) Project planning phase
c) Projection execution
d) Project closure



Project initiation phase:

- Develop a business case
- Undertake a feasibility study
- Project definition report
- Appoint the project team
- Setup the project office
- Perform phase review

Project planning phase:

- Create a project plan
- Create a financial plan
- Create a risk plan
- Create a communication plan
- Create a recourse plan
- Create an acceptance plan
- Create a procurement plan

- Contract the suppliers

Projection execution phase:

a. **Build deliverables**
b. **Monitor and control**
    - Perform time management
    - Perform cost management
    - Perform quality management
    - Perform change management
    - Perform risk management
    - Perform issue management
    - Perform procure management
    - Perform acceptance management
    - Perform communication management

Project closure phase:

- Perform project closure
- Review project completion

Lecture 5: SPM Objectives, Management Spectrum

SPM Objectives:

Project objective is a statement specifying the result to be achieved. These statements form the foundation for the entire planning process, including development of the master schedule.

A well defined project objective has several characteristics:

1. Attainable
2. Definitive
3. Quantifiable
4. Specific duration

Trade-off function:

Primary objective of SPM is to establish relation between performance, time and budget**.**

Performance = f (time, budget)

Time = f (budget, performance)

Budget = f (performance, time)

SMART Principals:

- Specific
- Measurable
- Realistic
- Timely

Management Spectrum:

Effective SPM focuses on the three P's:

- People
- Problem
- Process

People: The software Engineering Institute (SEI) sponsored the People Management Meturity Model that defines the key practice areas for software people that's given below:

- Recruiting
- Selection
- Performance management

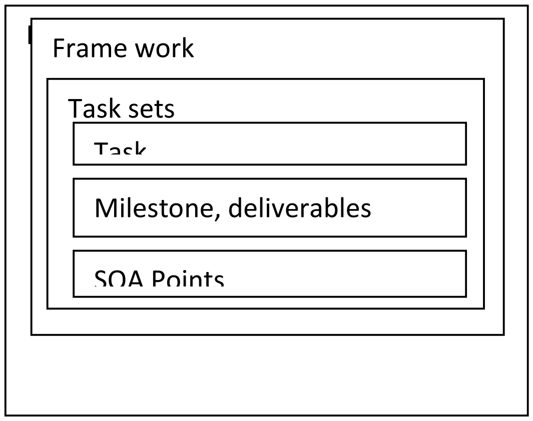
- Training
- Compensation
- Career development
- Organization team and culture development

Problem: A structured customer communication process is required for problem analysis for example Joint application Design (JAD)

For well define the problem JAD provide five phases as given below:

- Project definition
- Research
- Preparation
- Meeting
- Document preparation

Process: a software process can be characterize by process framework



To determine the organization's current state of process maturity we use an assesment, questionaire and a five point grade scheme these are ……..

Level 1: Initial

Level 2: Repeatable

Level 3: Defined

Level 4: Managed

Level 5: Optimizing

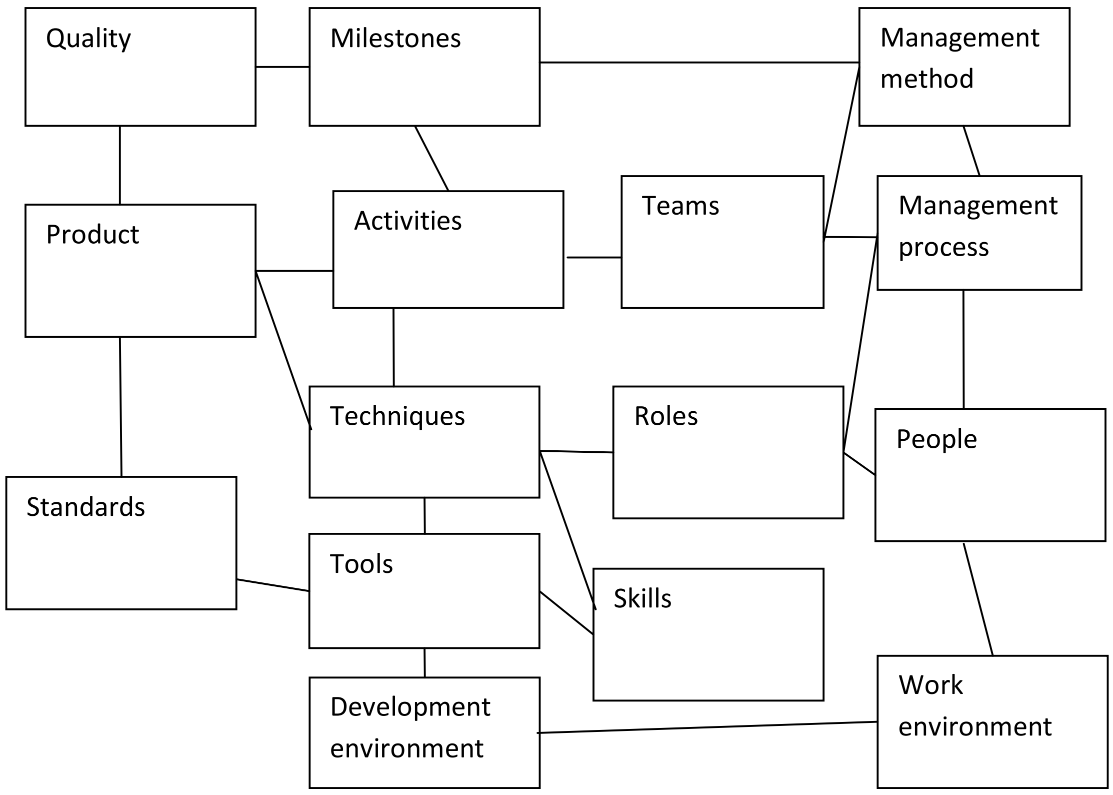
These five levels derived as a result of evaluating response to the assessment and questionnaire based on CMM and numerical grade point that indicate organization’s process maturity

Key Process Area (KPA) each maturity level is given below:

- Goal
- Commitments
- Abilities
- Activities
- Method of verifying implementation

Lecture 6: SPM Framework, Software Project Planning

SPM framework is a process pattern view of project activities that focus on the communication and people centric aspect of project management and SPM can be built on this activities. A SPM frame work interaction /interrelation between project management activivities are given below….



Hierarchies of project attributes are organized under SPM framework.

a) Product related attribute
   - Reliability
   - Speed
   - Memory
b) Project related attribute
   - Cost
   - Time

- Quality

Software Project Planning:

-Introduction:

The project plan defines the work that will be done on the project and who will do it.

It consists of:

- A statement of work (SOW) that describes all work products that will be produced and a list of people who will perform that work.
- A resource list that contain a list of all resources that will be needed for the product and their availability.
- A work breakdown structure and a set of estimates
- A project schedule
- A risk plan that identify any risks that might be encountered and indicates ho those risks would be handeled.

-Select project:

-Identify its scope and objectives:

-Identify project infrastructure:

-Analyze project characteristics:

-Identify project product and activities:

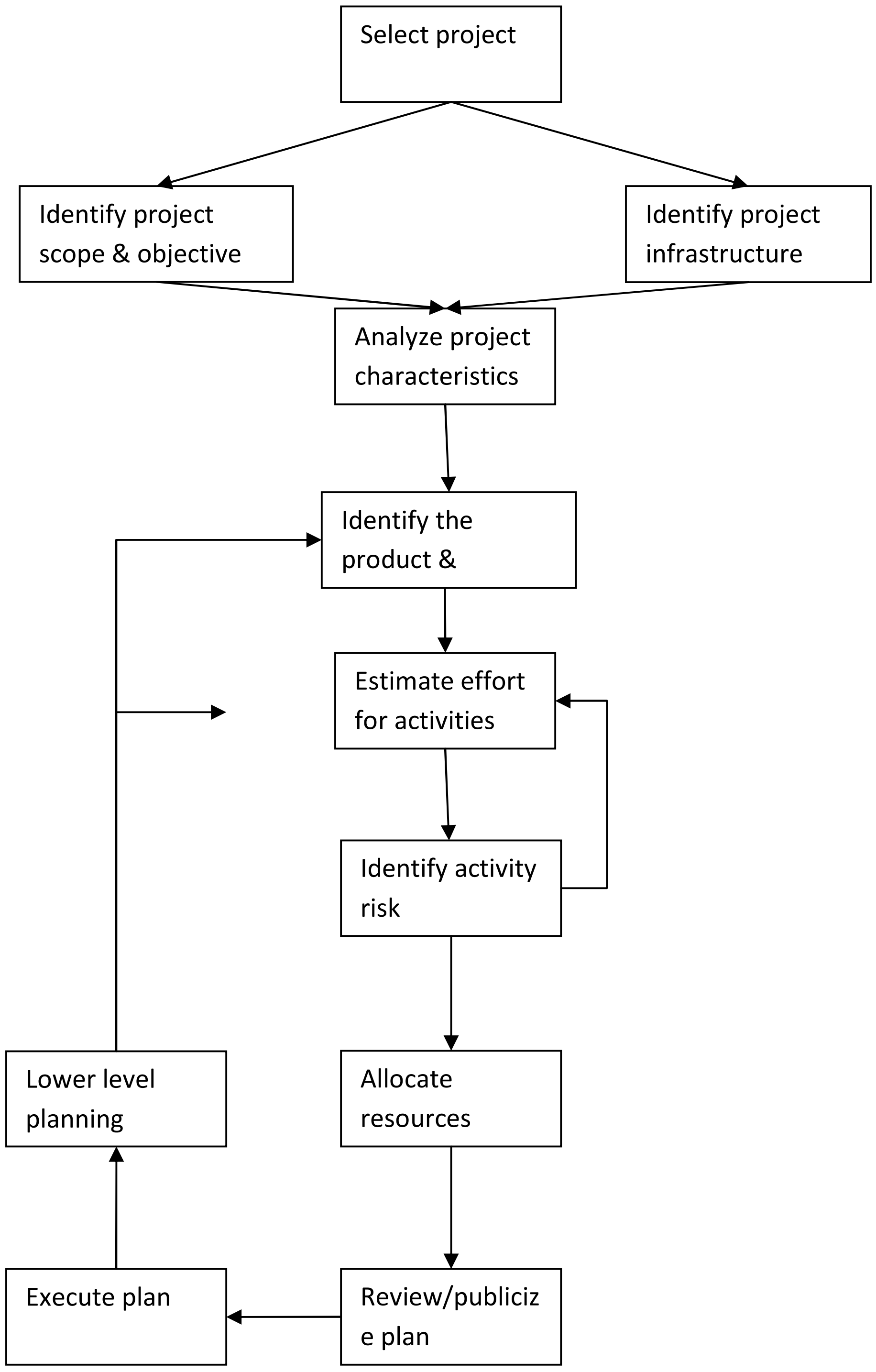
-Estimate effort for each activitiy:

-Identify activity risks:

-Allocate resources:

-Execute plan:

-Lower levels of planning:


Select project
Identify project scope & objective
Identify project infrastructure
Analyze project characteristics
Identify the product &
Estimate effort for activities
Identify activity risk
Allocate resources
Review/publiciz e plan
Execute plan
Lower level planning

Lecture 7: Planning Objectives, Project Plan, Types of project plan

Planning Objectives:

Guideline 1: The project definition, its scope must be bounded, accurate & ambiguous.

Guideline 2: Develop a conceptual design

Guideline 3: Refinement is done

Guideline 4: Detail design is developed

Guideline 5: Documentation is done

Project Plan:

- Plan is prepared during the conception/definition phase i.e. requirements,tasks,budgets,schedule,responsibility etc
- The plan is compared with "actual" during execution phase
- Corrective action is taken where appropriate i.e. preplanning is done

Types of project plan:

- Scope management plan
- Schedule management plan
- Cost management plan
- Quality management plan
- Staffing management plan
- Risk management plan
- Communication management plan
- Procurement management plan

Structure of a SPM Plan:

Have four major sections……

- The project summery
- The project planning
- The project tracking
- The project team

The project summery:

- Start date and end date of project
- Information related to project leader
- Project objectives
- Comment made to the customer on milestone and deliverables
- Explicitly listed assumptions as they save as a source of risks
- Details of billing so that business manager can track them

The project planning:

- The development process being used
- The requirements trace ability plans
- The effort and schedule estimates
- The man power requirement based on their skills, roles
- The development environment needed
- The tool employed
- The quality plan
- The risk management

The project tracking:

This section defines the measurements to be taken and the system required for data storage , various project activities to be undertaken, the frequency and nature of the progress reporting and calculation procedures .

The project team:

This section defines the project team and its structure. It also gives the roles and responsibilities of various people.

Lecture 8: **Software project estimation, Estimation methods:**

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.

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)

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 whiteboard 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.

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.

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.

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., longdistance 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.

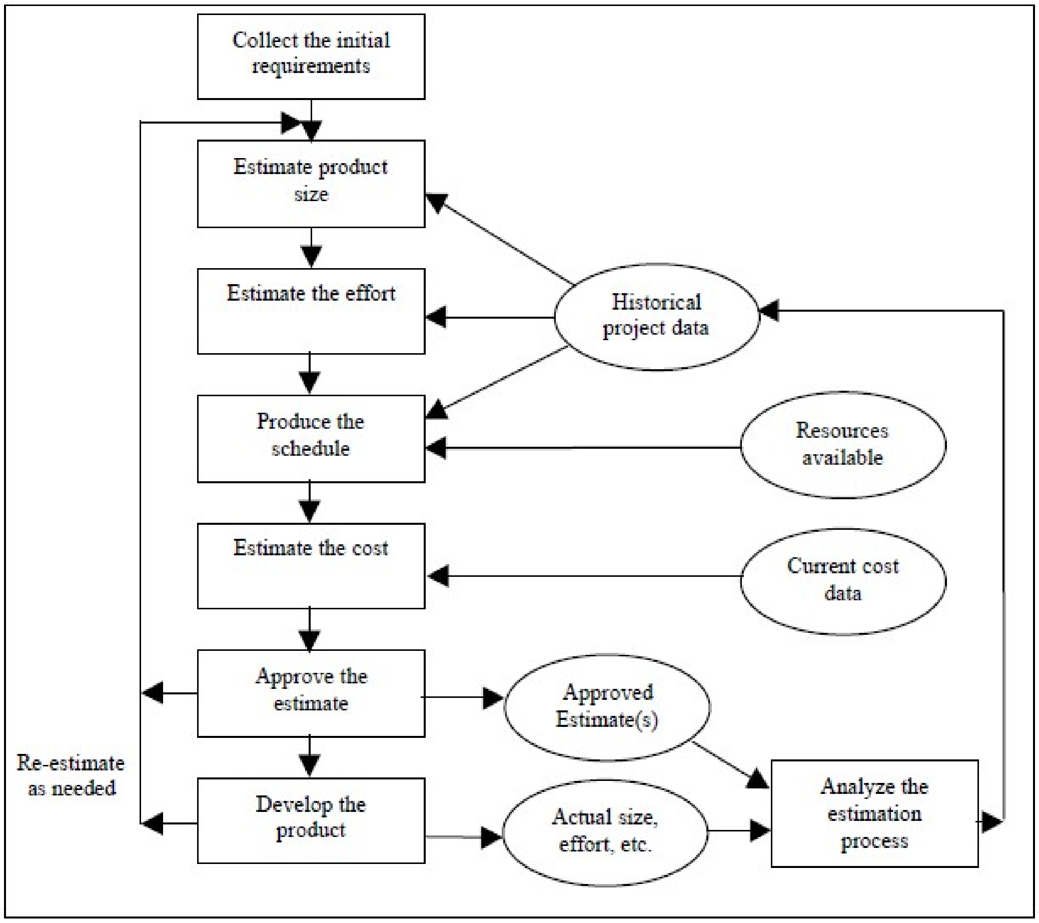


Figure 1 – The Basic Project Estimation Process

Estimation technique:
Estimation of various project parameters is a basic project planning activity. The important project parameters that are estimated include: project size, effort required to develop the software, project duration, and cost. These estimates not only help in quoting the project cost to the customer, but are also useful in resource planning and scheduling. There are three broad categories of estimation techniques:

- Empirical estimation techniques
- Heuristic techniques
- Analytical estimation techniques

Empirical Estimation Techniques
Empirical estimation techniques are based on making an educated guess of the project parameters. While using this technique, prior experience with development of similar products is helpful. Although

empirical estimation techniques are based on common sense, different activities involved in estimation have been formalized over the years. Two popular empirical estimation techniques are:
Expert judgment technique and Delphi cost estimation.

Expert Judgment Technique

Expert judgment is one of the most widely used estimation techniques. In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly. Usually, the expert estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate. However, this technique is subject to human errors and individual bias. Also, it is possible that the expert may overlook some factors inadvertently. Further, an expert making an estimate may not have experience and knowledge of all aspects of a project. For example, he may be conversant with the database and user interface parts but may not be very knowledgeable about the computer communication part.

A more refined form of expert judgment is the estimation made by group of experts. Estimation by a group of experts minimizes factors such as individual oversight, lack of familiarity with a particular aspect of a project, personal bias, and the desire to win contract through overly optimistic estimates. However, the estimate made by a group of experts may still exhibit bias on issues where the entire group of experts may be biased due to reasons such as political considerations. Also, the decision made by the group may be dominated by overly assertive members.

Delphi cost estimation

Delphi cost estimation approach tries to overcome some of the shortcomings of the expert judgment approach. Delphi estimation is carried out by a team comprising of a group of experts and a coordinator. In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate. Estimators complete their individual estimates anonymously and submit to the coordinator. In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation. The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators. Based on this summary, the estimators re-estimate. This process is iterated for several rounds. However, no discussion among the estimators is allowed during the entire estimation process. The idea behind this is that if any discussion is allowed among the estimators, then many estimators may easily get influenced by the rationale of an estimator who may be more experienced or senior. After the completion of several iterations of estimations, the coordinator takes the responsibility of compiling the results and preparing the final estimate.

Heuristic Techniques

Heuristic techniques assume that the relationships among the different project parameters can be modeled using suitable mathematical expressions. Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the mathematical expression. Different heuristic estimation models can be divided into the following two classes: single variable model and the multi variable model.
Single variable estimation models provide a means to estimate the desired characteristics of a problem, using some previously estimated basic (independent) characteristic of the software product such as its size. A single variable estimation model takes the following form:

$$\textbf{Estimated Parameter} = c_1 * e_1^{d}$$

In the above expression, e is the characteristic of the software which has already been estimated (independent variable). *Estimated Parameter* is the dependent parameter to be estimated. The dependent parameter to be estimated could be effort, project duration, staff size, etc. c_1 and d_1 are constants. The values of the constants c_1 and d_1 are usually determined using data collected from past projects (historical data). The basic COCOMO model is an example of single variable cost estimation model.
A multivariable cost estimation model takes the following form:

$$\textbf{Estimated Resource} = c_1 * e_1^{d_1} + c_2 * e_2^{d_2} + \ldots$$

Where e_1, e_2, … are the basic (independent) characteristics of the software already estimated, and c_1, c_2, d_1, d_2, … are constants. Multivariable estimation models are expected to give more accurate estimates compared to the single variable models, since a project parameter is typically influenced by several independent parameters. The independent parameters influence the dependent parameter to different extents. This is modeled by the constants c_1, c_2, d_1, d_2, … . Values of these constants are usually determined from historical data. The intermediate COCOMO model can be considered to be an example of a multivariable estimation model.

Analytical Estimation Techniques

Analytical estimation techniques derive the required results starting with basic assumptions regarding the project. Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis. Halstead's software science is an example of an analytical technique. Halstead's software science can be used to derive some interesting results starting with a few simple assumptions. Halstead's software science is especially useful for estimating software maintenance efforts. In fact, it outperforms both empirical and heuristic techniques when used for predicting software maintenance efforts.

Estimation model:

Organic, Semidetached and Embedded software projects
Boehm postulated that any software development project can be classified into one of the following three categories based on the development complexity: organic, semidetached, and embedded. In order to classify a product into the identified categories, Boehm not only considered the characteristics of the product but also those of the development team and development environment. Roughly speaking, these three product classes correspond to application, utility and system programs, respectively. Normally, data processing programs are considered to be application programs. Compilers, linkers, etc., are utility programs. Operating systems and real-time system programs, etc. are system programs. System programs interact directly with the hardware and typically involve meeting timing constraints and concurrent processing.

Boehm's [1981] definition of organic, semidetached, and embedded systems are elaborated below.
Organic: A development project can be considered of organic type, if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.
Semidetached: A development project can be considered of semidetached type, if the development consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

Embedded: A development project is considered to be of embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.

COCOMO

COCOMO (Constructive Cost Estimation Model) was proposed by Boehm [1981]. According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

Basic COCOMO Model

The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

Effort = a_1 x (KLOC)^{a_2} PM

Tdev = b_1 x (Effort)^{b_2} Months

Where

- KLOC is the estimated size of the software product expressed in Kilo Lines of Code,
- a_1, a_2, b_1, b_2 are constants for each category of software products,
- Tdev is the estimated time to develop the software, expressed in months,
- Effort is the total effort required to develop the software product, expressed in person months (PMs).

The effort estimation is expressed in units of person-months (PM). It is the area under the person-month plot (as shown in fig. 11.3). It should be carefully noted that an effort of 100 PM does not imply that 100 persons should work for 1 month nor does it imply that 1 person should be employed for 100 months, but it denotes the area under the person-month curve

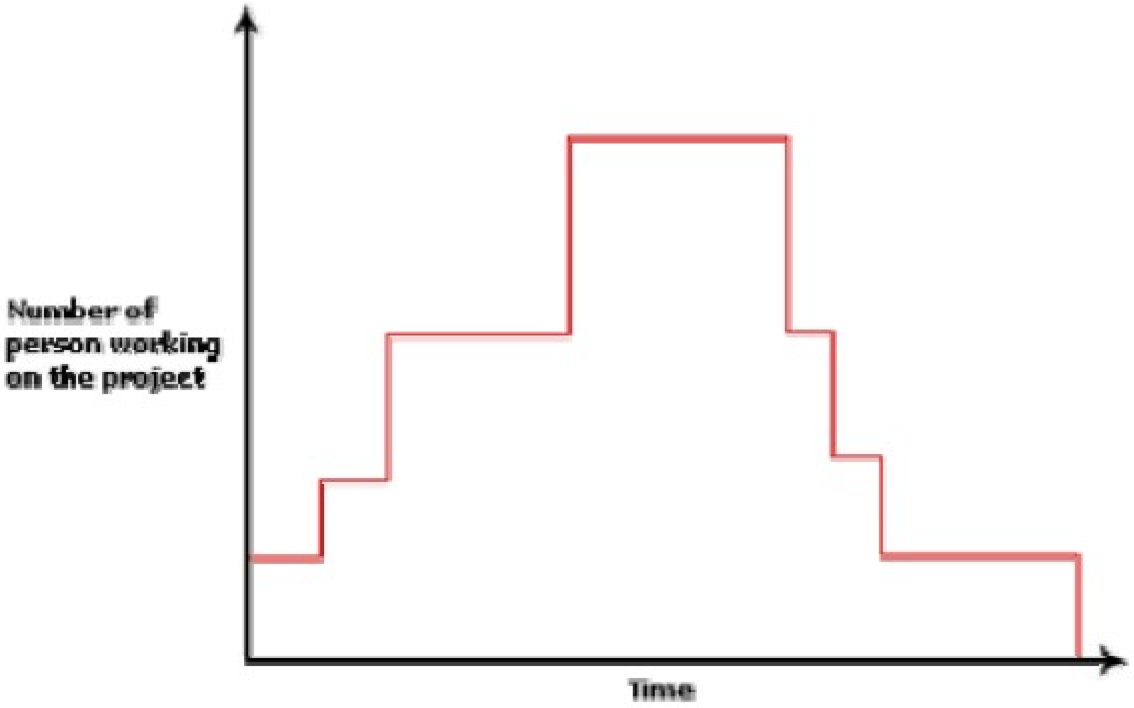


Fig 11.3 Person- month curve

According to Boehm, every line of source text should be calculated as one LOC irrespective of the actual number of instructions on that line. Thus, if a single instruction spans several lines (say n lines), it is considered to be nLOC. The values of a_1, a_2, b_1, b_2 for different categories of products (i.e. organic, semidetached, and embedded) as given by Boehm [1981] are summarized below. He derived the above expressions by examining historical data collected from a large number of actual projects.

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic : $\textbf{Effort} = \mathbf{2.4}(\mathit{KLOC})^{\mathbf{1.05}}$ **PM**

Semi-detached : $\textbf{Effort} = \mathbf{3.0}(\mathit{KLOC})^{\mathbf{1.12}}$ **PM**

Embedded : $\textbf{Effort} = \mathbf{3.6}(\mathit{KLOC})^{\mathbf{1.20}}$ **PM**

Estimation of development time

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic : $\textbf{Tdev} = \mathbf{2.5}(\mathit{Effort})^{\mathbf{0.38}}$ **Months**

Semi-detached : $\textbf{Tdev} = \mathbf{2.5}(\mathit{Effort})^{\mathbf{0.35}}$ **Months**

Embedded : **Tdev = 2.5(*Effort*)$^{0.32}$ Months**

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig. 11.4 shows a plot of estimated effort versus product size. From fig. 11.4, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.

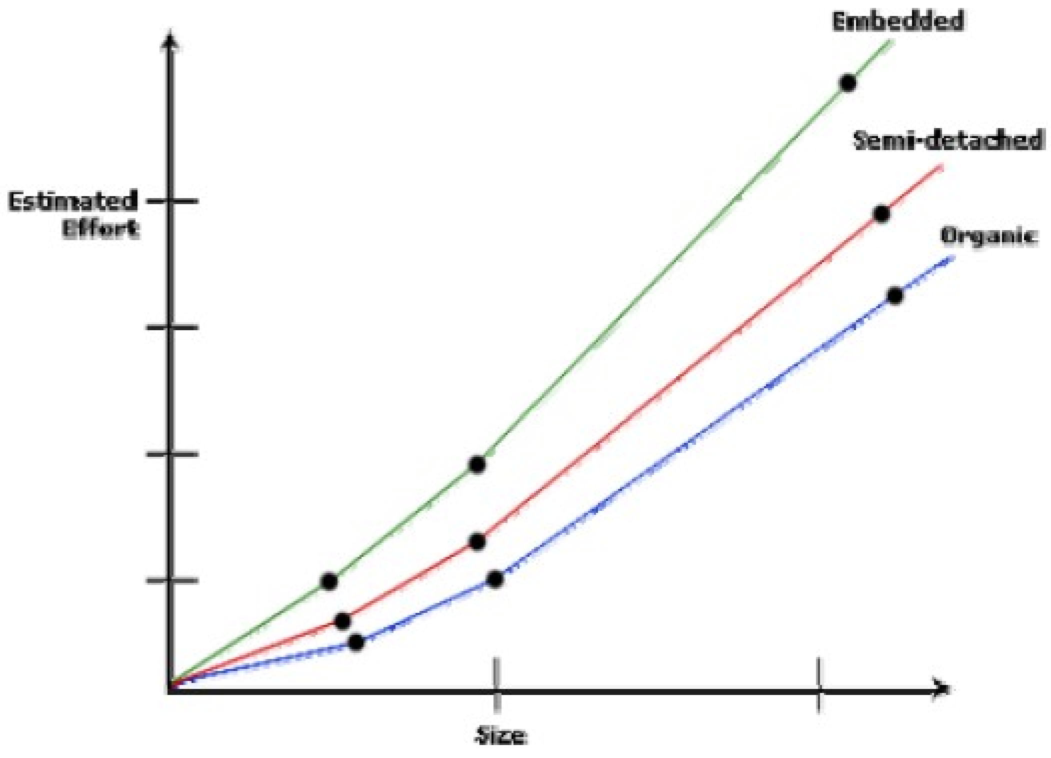


Fig. 11.4: Effort versus product size

The development time versus the product size in KLOC is plotted in fig. 11.5. From fig. 11.5, it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig. 11.5, it can be observed that the development time is roughly the same for all the three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.

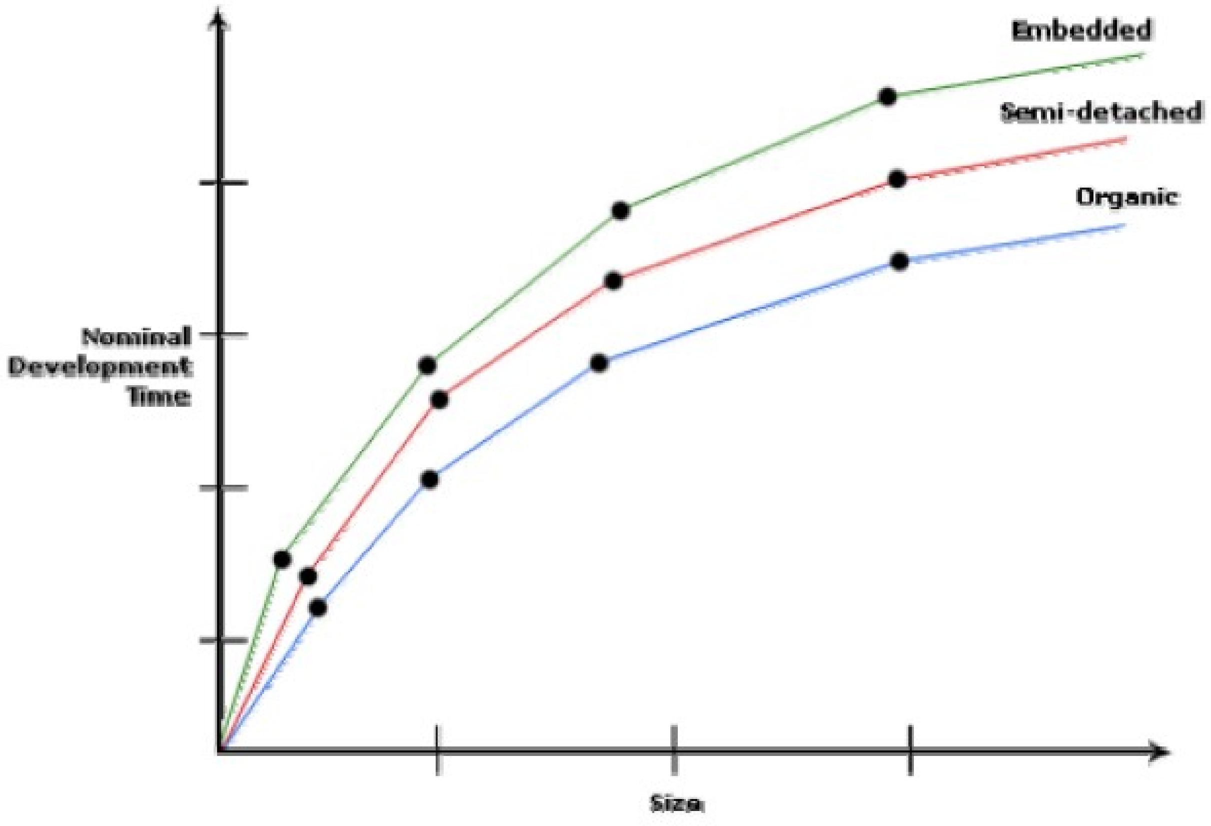


Fig. 11.5: Development time versus size

From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.
It is important to note that the effort and the duration estimations obtained using the COCOMO model are called as nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.
Example:
Assume that the size of an organic type software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software engineers be Rs. 15,000/- per month. Determine the effort required to develop the software product and the nominal development time.
From the basic COCOMO estimation formula for organic software:
Effort = $\mathbf{2.4 \times (32)^{1.05} = 91}$ **PM**
Nominal development time = $\mathbf{2.5 \times (91)^{0.38} = 14}$ **months**

Cost required to develop the product = **14 x 15,000**
= **Rs. 210,000/-**

Intermediate COCOMO model
The basic COCOMO model assumes that effort and development time are functions of the product size alone. However, a host of other project parameters besides the product size affect the effort required to develop the product as well as the development time. Therefore, in order to obtain an accurate estimation of the effort and project duration, the effect of all relevant parameters must be taken into account. The intermediate COCOMO model recognizes this fact and refines the initial estimate obtained using the basic COCOMO expressions by using a set of 15 cost drivers (multipliers) based on various attributes of software development. For example, if modern programming practices are used, the initial estimates are scaled downward by multiplication with a cost driver having a value less than 1. If there are stringent reliability requirements on the software product, this initial estimate is scaled upward. Boehm requires the project manager to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, he suggests appropriate cost driver values which should be multiplied with the initial estimate obtained using the basic COCOMO. In general, the cost drivers can be classified as being attributes of the following items:
Product: The characteristics of the product that are considered include the inherent complexity of the product, reliability requirements of the product, etc.
Computer: Characteristics of the computer that are considered include the execution speed required, storage space required etc.
Personnel: The attributes of development personnel that are considered include the experience level of personnel, programming capability, analysis capability, etc.
Development Environment: Development environment attributes capture the development facilities available to the developers. An important parameter that is considered is the sophistication of the automation (CASE) tools used for software development.
Complete COCOMO model
A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems. These sub-systems may have widely different characteristics. For example, some sub-systems may be considered as organic type, some semidetached, and some embedded. Not only that the inherent development complexity of the subsystems
may be different, but also for some subsystems the reliability requirements may be high, for some the development team might have no previous experience of similar development, and so on. The complete COCOMO model considers these differences in characteristics of the subsystems and estimates the effort and development time as the sum of the estimates for the individual subsystems. The cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.
The following development project can be considered as an example application of the complete COCOMO model. A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:

- Database part
- Graphical User Interface (GUI) part
- Communication part

Of these, the communication part can be considered as embedded software. The database part could be semi-detached software, and the GUI part organic software. The costs for these three components can be estimated separately, and summed up to give the overall cost of the system.