UNIT 5

Software Project Management: Project Management Essentials, What is Project management, Software Configuration Management.

Project Planning and Estimation: Project Planning activities, Software Metrics and measurements, Project Size Estimation, Effort Estimation Techniques.

INTRODUCTION

- Software project management is an activity crucial to the success of software projects.
- Software products are more complex entities in which defects are mostly discovered during development.
- Software changes are limited to changes in programs and the volume of change depends on understanding requirements of customers and developers.
- In the current scenario, people working on a project to develop software are geographically distributed; hence, proper coordination and communication management is required among them.
- Due to the lack of domain knowledge certain tools are needed for project size estimation, planning, software development, and so on.
- Quality software can only be produced if a systematic process and appropriate techniques are followed for the development of software.
- It is observed that several software projects run behind schedule, over budget, and fail to meet the customer requirements.
- In a software company, many software projects run concurrently; they share resources and their activities are synchronized for optimal use of resources.

Goal of SPM

- Development of quality software is the primary objective of software project management.
- *The goal of software project management is to enable a group of people to work efficiently on a project, using a systematic process, in order to produce a quality software product.*
- Software project management uses a more established and specialized approach as compared to general project management.

PROJECT MANAGEMENT ESSENTIALS

- Project management is an integrated environment where various entities are interconnected and performed together to produce a quality software product.
- There are four essential elements of effective software project management:
  - Project
  - People
  - Process
  - Product

- Effective software project management focuses on these elements to fulfill the customer needs.

Project

- A *project* is a temporary endeavor with defined starting and ending deadlines, roles and responsibilities, conditions, budgets and plans, undertaken to accomplish aim and objectives.
- Each project has a unique purpose even if there are multiple projects in a single domain.
- Projects are basically temporary in nature due to their flexibility to change.
- Uncertainties are involved in a software project.
- Requires resources (hardware, software, and human resources), typically from different areas. A successful project always satisfies the project constraints expected by the customer.
- Project constraints are defined in the project scope, which defines the work products to be produced in a project using processes.
- The most common constraints that are employed on the project scope are project cost, time, and quality.
- Project stakeholders must agree upon the defined project scope.
- The main focus of project management is to satisfy the customer through production of quality products.

People

- *People (or stakeholders) in a project are those who are either involved in or are affected by the project.*
- Each person in the project has certain roles and responsibilities according to their skill sets.
- They should be motivated, trained, rewarded, deployed, and retained as and when required to improve their capabilities.
- Poor management sometimes leads to project failures.

People Involved in Project

- Senior Managers
- Project Managers

- Programmers
- Support Staff
- Customers
- End Users
- Project Sponsors
- Competitors
- Suppliers

Process

- A software process describes the characteristics and organization of activities in order to produce software.
- The general activities of software processes include
  - Definition
  - Development
  - Implementation
- The selection of an appropriate software process model according to the project is a challenge for the project manager.
- Process models ultimately affect the product quality.

Product

- A product is the final outcome of a project.
- It is produced with an effective project management process.
- The project manager must determine the requirements and the expected outcomes at the beginning of the project.
- Project scope of the product must be clearly defined, which helps us to produce a quality product.
- Project scope is refined into discrete functional units and the development schedule is planned for each functional units.
- The quality of the product also depends on the process being used for product development.

PROJECT MANAGEMENT

- *Project management is the application of knowledge, skills, tools, and techniques for performing project activities in order to satisfy the expectations of the stakeholders from a project.*
- Project management is necessary to find the pitfalls and create outlets to avoid the unintended consequences of the project.

- Software project management is the key to successful delivery of software projects as per customers' expectations.

Project Management Knowledge Areas

- The knowledge areas are the competencies of the process, practice, tools, and techniques that the project managers must have for managing the projects.
- The main knowledge areas for project management are:
  - Scope
  - Time
  - Cost
  - Quality
  - People
  - Communications
  - Risk,
  - Procurement
  - Project Integration Management
- Each of these knowledge areas has some processes for its execution in order to achieve an effective project management program.
- Project management skills include:
  - General Management
  - Human Resource Management
  - Procurement Management
  - Communication Management
  - Risk Management.
- Project management tools and techniques assist in carrying out the activities of project managers.

Project Failures

- Inability to clearly understand customer needs
- Lack of user involvement
- Unrealistic expectations
- Lack of top management support
- Incomplete requirements and specifications
- Lack of resources (hardware, software, and people)
- Lack of good planning
- Technical incompetence
- Changing business needs and requirements

- Changes are managed poorly
- Lack of best practices
- The chosen technology changes
- Ill-defined responsibility of team members
- Lack of communication among team members

Project Success

- Strong support from the top management
- A sound methodology for the project
- Tactical and technical leadership by competent people
- User involvement in each activity
- Experienced project manager
- Clearly defined business goals
- Clearly defined project scope
- Feedback mechanism
- Sufficient resource allocation
- Adequate communication channels
- Troubleshooting capabilities
- Reliable estimates
- Clearly specified deadlines
- Parties sticking to the commitment

SOFTWARE CONFIGURATION MANAGEMENT

- Software configuration management is concerned with identification, tracking, and change of control of software configuration.
- *Software configuration management (SCM)* or *configuration management (CM)* is the discipline of identifying the configuration of a system at any time for the purpose of systematically controlling changes to the configuration throughout the system life cycle.
- The purpose of CM is to establish and maintain the integrity of a software product throughout software the development life cycle.
- The CM process improves product visibility, product protection, product control, customer confidence, and team communication.
- A systematic CM process reduces rework, confusions, and project risks
- The project management process provides a systematic CM plan and monitors the status of the SCM process.

- The *configuration management officer (CMO)* implements and maintains the CM process according to the CM plan.
- The CMO coordinates, supports, and performs the CM activities and reports to the project manager.
- The *Configuration Control Board (CCB)* approves and disapproves the changes to be performed in the product.
- The CCB consists of technical and administrative representatives for configuration planning and execution.
- The approved changes are performed through the development process.

- The CM process consists of the following four activities
  - *Configuration identification*
  - *Configuration change control*
  - *Configuration status accounting*
  - *Configuration auditing*

Configuration Identification

- A *software configuration item (SCI)* or *configuration item (CI)* is a work product that is designated for configuration management.
- The SCI activity identifies items to be controlled, establishes identification schemes for the items and their versions, and establishes the tools and techniques to be used in acquiring and managing controlled items.

- The work products in SCI may include plans, process descriptions, requirements, design data and models, product specifications, source codes, compilers, product data files, etc.
- The status of the CIs at a given point in time is called *baseline.*
- Each new baseline is the sum total of an older baseline plus a series of approved changes done on the CI.
- A baseline together with all approved changes to the baseline represents the *current configuration*.

Configuration Change Control

- The purpose of software configuration change control (CC) is to manage changes during the software life cycle.
- The activities of CC are to identify what changes to make, the authority for approving changes, and support for the implementation of those changes.
- The outcome of CC is useful in measuring change traffic, breakage, and aspects of rework.
- A change request (CR) for changes to software configuration items may be originated by anyone at any point in the software life cycle and may include a suggested solution and requested priority.
- The CRs are usually recorded on the software change request (SCR) form.
- Once an SCR is received, a technical evaluation is performed to determine the extent of modifications that would be necessary should the change request be accepted.
- An established authority, Configuration Control Board (CCB), will evaluate the technical and managerial aspects of the change request and accept, modify, reject, or defer the proposed change.

Configuration Status Accounting

- The software configuration status accounting records and reports the information needed for effective management of the software configuration.
- It is the bookkeeping process of each release for CIs.
- The CI are identified, collected, and maintained for the status accounting.
- The reported information can be used by the development team, the maintenance team and for project management and quality assurance activities.
- A database is maintained for providing the status accounting of configuration requests.
- This procedure involves tracking what is in each version of software and the changes that lead to this version.
- It keeps a record of all the changes made to the previous baseline to reach the new baseline.

Configuration Auditing

- Configuration auditing determines the extent to which an item satisfies the required functionalities.
- Configuration audit is performed to evaluate the conformance of software products and processes to applicable regulations, standards, guidelines, plans, and procedures.

- Auditing is performed according to a well-defined process consisting of various auditor roles and responsibilities.
- Expertise of people is required and some automated tools also support the planning and conduct of an audit process.

PROJECT PLANNING AND ESTIMATION

- A successful project is possible only through good project planning.
- Project planning concentrates on estimating resources, time, budgets, and monitoring and controlling the activities of project management.
- At the beginning of project planning, all the project constraints, such as staff and other requirements, overall budget, starting and ending dates, schedule, etc., are defined and the project manager has their details.
- During project planning, future estimates are planned for an effective project management.
- The project plan provides the basis for performing and managing the activities of a software project and addresses the commitment to stakeholder's resource constraints and capabilities of the software project.
- A bad project management plan leads to project failures and sometimes projects are cancelled.
- The main goal of the project plan is to establish a pragmatic strategy for controlling, tracking, and monitoring a project.
- A typical project plan includes project scope, project estimates, schedule, requirements, risk management plan, control strategy, and various other plans.

PROJECT PLANNING ACTIVITIES

- The project planning activities include both business-level and technical-level planning.
- Business-level planning addresses the relationships with the customer.
- Includes the project and business objectives, proposal writing, analysis and inclusion of functional requirements, product demand and its scope, and legal issues.
- Technical-level planning focuses on performing the technical activities.
- Concentrates on technical issues, such as selection of the development life cycle model, planning documentation methods and tools, risk management planning, estimations, financial planning, resource allocation and management, team structuring and organization, software development life cycle plan, documentation plan, configuration management plan, quality assurance plan, transition plan, and so on.
- A project plan includes various activities related to project management.
- A general project plan includes the following project planning activities, beginning with project initiation to project completion.

- *Defining business objectives and project scope*
- *Project estimations*
- *Project scheduling*
- *Resource planning*
- *Financial planning*
- *Staffing-level planning*
- *Development planning*
- *Project monitoring and control plan*
- *Risk management planning*
- *Quality assurance planning*
- *Configuration management planning*
- *Miscellaneous plans*

Defining business objectives and project scope:

- Business objective gives a clearly defined target that will be achieved through proper planning by the team.
- Project scope defines the parameters of the project with constraints for what the project will attempt to do.
- The need for product development and its demand in the market must be stated.
- A proposal is written that will work as a contract for the project.
- Functional requirements are analysed to prepare a good project plan.
- Legal issues, such as patent, copyright, liability, warranty etc., Must be specified well in advance.

Project estimations:

- Project estimation includes cost, size, duration, project complexity, and resource requirements.
- The cost involves both
    - direct cost (i.e. efforts made directly on the project; for example, time spent on developing the code, testing, etc.) and
    - Indirect cost (i.e. the amount spent on telephone, Internet, etc.).
- Estimation of size is the basis of project delivery and other estimations.
- Some projects are very simple to develop and other projects are so complex that they may take considerable amount of time for development.
- Project complexity determines the project duration and resource requirements.

Project scheduling:

- It emphasizes defining the start and completion time for each task in the project.
- The tasks are decomposed into manageable pieces that can be easily developed within the stipulated time and constraints.
- There are various tools and techniques used for project scheduling, such as:
    - Work Breakdown Structures (WBS),
    - Activity Network Diagram (AND),

- PERT-CPM,
- Gantt charts, and so on.

Resource planning:

- Usually multiple projects run in an organization.
- Each project manager will try to reserve the resources in advance according to the project schedule and functional requirements.
- The resources can be hardware, software, and people. Most of the organizations start recruiting people in advance.

Financial planning:

- Every project is concerned with financial plans.
- These plans provide the assurance that the project has the funds to accomplish its objectives.
- A cost estimation process is used to determine how money will be spent in software project management.
- A cost-benefit analysis is carried out to estimate the budget for personnel expenses, purchase orders, automated items, and purchase of other items.

Staffing-level planning:

- Staffing-level planning organizes the team and accurately estimates the number of personnel required for the project.
- A module-wise estimation is done to avoid resource conflicts.
- Due to the lack of engineers, organizations may use their skills and expertise in multiple projects.

Development planning:

- A software project is developed as a series of technical activities, such as analysis, design, coding, testing, verification and validation, deployment, documentation, training, maintenance, and so on.
- The project plan provides details on who will perform which task, how it will be performed, what support tools are required, and various such activities.
- The software development life cycle is also chosen before proceeding to development planning.

Project monitoring and control plan:

- The project monitoring plan assesses the progress of a project as to whether it is moving in the right direction or not.
- This plan helps in controlling the execution of activities.
- The project is monitored according to the project schedule for cost, time, efforts, and defects.

Risk management planning:

- The risk management plan assures that problems are discovered in early stages and appropriate strategy has been made to handle the risk.
- While managing risks, the cost, quality, and schedule estimates should be within constraints.
- Most organizations put their efforts to minimize risks.

***Quality assurance planning*:**

- Product milestones are rigorously tested to find uncovered defects.
- Quality assurance aims to prevent errors and assures that the delivered product exhibits high quality.
- There are some standards that provide certain guidelines and procedures to produce a quality product.

Configuration management planning:

- The configuration management plan encompasses the discipline that addresses identifying and controlling changes to the work products.
- The configuration management plan includes configuration identification, configuration change control procedure, monitoring and tracking changes, and auditing of the changes.

Miscellaneous plans:

- There are various miscellaneous plans, such as process tailoring, special testing plan, verification and validation plan, tools and environment plan, delivery plan, and maintenance plan.

SOFTWARE METRICS AND MEASUREMENTS

- Software metric is the unit of measurement of software product, project, and process attributes.
- Software metrics are the measures that are used to quantify products, processes, and projects.
- Measuring software process or project provides a foundation to predict the parameters of project planning.
- Effective use of metrics helps to measure product quality, manage projects, and assess processes.
- Metrics provide necessary information for monitoring the progress of the project.
- There are three types of metrics used in software projects:
    - *Product metrics,*
    - *Process metrics,*
    - *Project metrics.*

Product metrics:

- Product metrics are used to measure product characteristics such as product quality factors, such as reliability, maintainability, defects, and product operational characteristics.

Process metrics:

– Process metrics assess the effectiveness of software processes and ensure conformance of standards and guidelines.

– Process metrics provide some attributes that help to improve the process maturity.

Project metrics:

– Project metrics are used to monitor the project plan, track progress, and estimate the project attributes.

– It measures the size, cost, efforts, schedule, and risks.

– Generally, projects are assessed on the basis of past project experiences and data.

Project Size Estimation

- Size measurement is the initial step for estimating the other attributes of software.
- It is a direct measurement, which is based on the problem size.
- Programs written in any programming language have some size, whether the program is an assembly code, a high-level language code, a GUI code, or a component of a programming language.
- There are various units of size measurement, such as

– *Lines of code (LOC),*

– *Function point (FP),*

– *Token count (TC),*

- *Fuzzy logic sizing,*
- *Object point (OC),*
- *Standard component sizing, and many more.*

Lines of Code

- A line of code (LOC) metric is based on the measurement of the source lines of the code in a program.
- It is simply measured by counting the program header, declarations, executable, and non-executable lines in the source program.
- Comments lines, blank lines, and header lines are usually not considered during the LOC measurement.
- LOC is generally counted in kilo (thousand) line of codes (KLOC) per person-month (PM).
- Size measurement of the source lines helps to measure
  - Defects per KLOC,
  - Errors per KLOC,
  - Dollars per KLOC,
  - Pages of documentation per KLOC, and so on.
- Size estimation is performed by decomposing a problem into manageable modules.
- Small-size modules can easily be predicted for LOC counting.
- The overall project size is the sum of the sizes of all modules in the project.

Lines of Code: Disadvantages

– The value of LOC measurement varies with the programmer, programming language, and the project complexity.

– Every programmer has different technical skills, programming styles, and logical ability. Therefore, the LOC value may differ during estimation.

– Different programming languages have their different programming techniques. The actual number of the source lines may vary in programs.

– LOC only considers the source code. But some projects are highly complex and they need much effort at the design and analysis phases.

– LOC is not suitable for component or reuse- based programming technologies where the components are considered as the unit of measurement.

– Also, it is very difficult to accurately estimate the project size from the requirement specification or project nature.

– There is a lack of standard tools for counting the source lines.

– The quality of code is the main focus for quality software, which is poorly considered during size measurement.

Function Point Analysis

– To overcome the limitations of the LOC-based measurement, Alan Albrecht proposed another size estimation technique called the function point (FP) analysis.

– The size of the project is estimated on the basis of functions or services requested by the customer in the requirement specification.

– It is a structured technique that decomposes systems down into smaller components so that they can be better understood and analysed.

– It relies on the product features delivered to the customer.

– The actual number of function points can be verified at the end of each stage of a project.

Function Point Analysis: Advantages

– FP measurement is programming language independent and programmer independent.

– It does not have any constraint specific to the hardware, procedural or non-procedural languages.

– It is very easy to predict the function points in the final product from the requirement specification.

– More accurate estimates are possible in the early stages of software development.

– An important aspect of FP is the consideration of user’s view through requirement specification or problem description along with developers view during requirement decomposition for the FP analysis.

Method:

– FP-based estimations are based on the following five information domain values and their complexities in a particular project.

- Number of inputs
- Number of outputs
- Number of inquiries
- Number of internal logical files
- Number of external interfaces

– The value of each of these five information domains is collected and a subjective evaluation is performed to categorize them as simple, average, and complex.

– There are certain weights assigned at each complexity level to the information domains

–

Function Point Analysis: Method

1. Calculate the unadjusted function point (UFP).

- It is calculated by simply counting the value of each information domain and multiplying it by an appropriate weight at its complexity level.

2. Compute the complexity adjustment attributes (CAA).

- The CAAs are complexity attributes (14) that can vary from project to project. They are computed using the following relationship:

CAA = [0.65 + 0.01 × ∑ CAAi], where CAA is the complexity adjustment attributes.

3. Then, compute FP = UFP × CAA.

Example 4.1

Compute the FP value for the grade calculation of students. Assume that it is an average complexity size project. The information domain values are as follows: number of inputs = 13, number of outputs = 4, number of inquiries = 2, number of external files = 5, number of interfaces = 2. The total value of complexity adjustment attributes is 13.

Solution 4.1:

Calculation of UFP for average complexity size project:

= (Number of inputs) × 4 + (Number of outputs) × 5 + (Number of inquiries) × 4 + (Number of files) × 10 + (Number of interfaces) × 7

= $13 \times 4 + 4 \times 5 + 2 \times 4 + 5 \times 10 + 2 \times 7 = 175$

Compute CAA, which has the value = 13

= $0.65 + 0.01 \times (13 \times 3)$

= $0.65 + 0.01 \times 39$

= 1.04

Compute FP = UFP × CAA

= $175 \times 1.04 = 182$

Thus, the total value of FP is 182.

Function Point Analysis: Benefits

- It accurately estimates the project cost, project duration, and project staffing size.
- It is helpful to monitor the productivity level of a project.
- It is also useful in managing change of scope and in communicating the functional requirements.
- The FP computation is useful to find other metrics, such as project defect per FP, cost per FP, FP per hour productivity, and so on.

Feature Point Metric

- The feature point metric is an extension of the function points that includes the algorithmic complexity of the software.
- An algorithm is a step-wise procedure with defined rules which are designed to solve a significant computational problem.
- For example, a sine routine can be considered as an algorithm. Each algorithm is assigned a weight ranging from 1 (elementary) to 10 (sophisticated algorithms).
- The feature point metric is the weighted sum of algorithms plus the function points.
- It is especially useful for systems with a few inputs/outputs and a high algorithmic complexity, such as mathematical software, CAD, AI, discrete simulations, and military applications.

EFFORT ESTIMATION

- Effort estimation predicts how much time is required to complete a project, how much it costs, and how many engineers are required for completing the project.
- The effort estimation is important for cost-benefit determination, project scheduling, and monitoring and control.
- The effort estimation is a continuing activity that starts with project proposal.
- More accurate estimates are possible as the project proceeds, i.e., during the feasibility study, requirement analysis, design, and subsequent phases.
- There are various factors that affect cost estimation such as, capability of engineers, product complexity, product size, availability of delivery time, required reliability, and the level of technology.

Effort Estimation Approaches

The top-down approach

- The top-down approach estimates the project cost globally by estimating the system level costs of the overall project size.
- The system level cost includes the cost of resources for product development, configuration management, quality assurance, integration, installation, and deployment.
- It requires minimum project details and it is usually faster and easier to adopt.
- This approach does not consider other technical requirements for specific modules.
- In case of underestimation or overestimation, an inaccurate result of effort estimate may be produced that may lead to an under-budget or over-budget project.

The bottom-up approach

- In the bottom-up approach, the cost of individual modules is estimated and then all these costs are integrated to estimate the overall project cost.
- This approach can produce more accurate estimate due to the consideration of discrete parts for estimation.
- It is more stable as estimation errors in modules can be rectified and balanced. But someone may overlook the system-level costs, such as quality assurance cost, configuration management cost, etc.
- Size estimate is the primary need for both top-down and bottom-up approaches.

Effort Estimation Techniques

- There are different techniques for effort estimation. Some of the most popular estimation techniques are
  - *Estimation by analogy,*
  - *Delphi estimation,*
  - *Algorithmic cost modelling, and*
  - *Analytical techniques*

Estimation by analogy

- The estimation-by-analogy technique is based on the experiences of the past projects.
- The estimation-by-analogy method involves an expert for finding the analogous situations so as to give his opinion.
- This technique follows a top-down estimation approach.
- The project to be assessed is characterized on certain factors that become the basis for finding similar or analogous projects which have been completed and their efforts are known.
- Then the proposed project is compared with the previously completed similar projects.
- These effort values are then used with some adjustments to estimate the efforts for the proposed project.

Estimation by analogy: Problems

- The estimators have to determine how best they can characterize the projects in terms of application domain, the number of inputs, the number of distinct entities referenced, the number of screens, and so on.
- Another problem after project characterization is to determine similarity: how much confidence we can place in the analogies and how many projects need to be compared.
- A few analogies might lead to maverick projects being used and too many analogies might lead to dilution of the effect of the closest analogies.
- Lastly, it is very difficult what variables should be considered to estimate the new projects with known effort values from the analogous projects.
- Means and weighted means can give more influence to the closer analogies.

Delphi estimation

- Delphi cost estimation is a group consensus technique that relies on expert judgment and follows the top-down estimation approach with certain features of the bottom-up approach.
- The coordinator interacts with experts for providing necessary information and documents.
- The estimators go through the requirement document and make their estimates anonymously.
- The coordinator compiles the estimation reports and distributes a summary of the estimation responses.
- The estimation report specifies any unusual characteristic noted by the estimators.
- The coordinator can call a group meeting of experts to discuss points if the estimates vary widely.
- The estimation is iterated for as many rounds as appropriate for an accurate estimate.

Delphi estimation: Benefits

- The experts use their experiences of the past projects to assess the factors in the proposed project.
- Each expert applies their distinguished expertise and knowledge to assess the project.
- There is less chance for bias due to group consensus.

- There are rare chances of bias if only a few experts, maybe one or two, are involved in the estimation process.

Delphi estimation: Problems

- The team may take pain in gathering some information that may not be useful for estimation in the proposed project.
- This method is very difficult to quantify.
- It is hard to document the factors used by the experts or expert groups.

Algorithmic cost models

- Algorithmic cost models use size estimation model, which is based upon certain project parameters.
- The value of these parameters depends on the project type:
  - – ***Organic projects*** are very simple and can be developed with a small-size team. The team should have good application experience and should be familiar with the application environments. A simple data processing system is a good example of the organic category.
  - – ***Embedded projects*** are very complex and have stringent constraints, for example, flight control system for aircraft.
  - – ***Semidetached projects*** are intermediate in size and complexity. The team should have mixed experience to meet the mix of rigid and less-than-rigid requirements. A transaction processing system with fixed requirements for terminal hardware and database software is an example of the semi-detached category.

COCOMO Model

- COCOMO (COnstructive COst MOdel) is an algorithmic cost estimation technique proposed by Boehm, which works in a bottom-up manner.
- It is designed to provide some mathematical equations to estimate software projects.
- These mathematical equations are based on historical data and use project size in the form of KLOC.
- The COCOMO model uses a *multivariable* size estimation model for effort estimation.
- A multivariable model depends on several variables, such as development environment, user involvement, memory constraints, technique used, etc.
- A *single variable* model is based only upon the size of the project, which is given as

$$\text{Effort} = a \times \text{Size}^{b}$$

- The constants a and b are derived from the historical data of the past projects in the organizations.
- The values of a and b in COCOMO model vary across the three categories of projects: organic, semidetached, and embedded, as shown in Table 4.2.

Project category	a	b
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

Table 4.2: The values of *a* and *b* constants

- COCOMO estimation is a family of hierarchical models, which includes
  - *Basic,*
  - *Intermediate,* and
  - *Detailed COCOMO* models.
- Each of the models initially estimates efforts based on the total estimated KLOC.

Basic COCOMO Model

- The basic COCOMO model estimates effort in a function of the estimated KLOC in the proposed project.
- The basic COCOMO model is very simple, quick, and applicable to small to medium organic-type projects. It is given as follows:

  Development effort (E) = $a \times (KLOC)^{b}$

  Development time (T) = $c \times (E)^{d}$

- Where *a, b, c,* and *d* are constants and these values are determined from the historical data of the past projects.
- The development time (T) is calculated from the initial development effort (E).
- The values of *c* and *d* for organic-, semidetached-, and embedded-type projects are shown in Table 4.3.

- **Example 4.2**

Assume that a system for simple student registration in a course is planned to be developed and its estimated size is approximately 10,000 lines of code. The organization is proposed to pay Rs. 25000 per month to software engineers. Compute the development effort, development time, and the total cost for product development.

Solution 4.2:

- The project can be considered an organic project. Thus, from the basic COCOMO

  model, Development effort (E) = $3.2 \times (10)^{1.05}$ = 35.90 PM

Development time (T) = 2.5 × $(35.90)^{0.38}$ = 9.747 months

- Total product development cost = Development time × Salaries of engineers

= 9.747 × 25000

= Rs. 2,43,675

Intermediate COCOMO Model

- Boehm has introduced 15 cost drivers, considering the various aspects of product development environment.
- These cost drivers are used to adjust the project complexity for estimation of effort and these are termed as *effort adjustment factors (EAF)*.
- These cost drivers are classified as computer attributes, product attributes, project attributes, and personnel attributes.
- The intermediate COCOMO model computes software development effort as a function of the program size and a set of cost drivers.
- The intermediate COCOMO model estimates the initial effort using the basic COCOMO model. Then the EAF is calculated as the product of 15 cost drivers.

Intermediate COCOMO Model

- Total effort is determined by multiplying the initial effort with the total value of EAF. The computation steps are summarized below.
    - Development effort *(E):*
        - Initial effort $(E_i) = a \times (\text{KLOC})^b$
        - $EAF = EAF_1 \times EAF_2 \times \ldots \times EAF_n$
        - Total development effort $(E) = E_i \times EAF$
    - Development time $(T) = c * (E)^d$

Example 4.3

Suppose a library management system (LMS) is to be designed for an academic institution. From the project proposal, the following five major components are identified:

Online data entry	-	1.0 KLOC
Data update	-	2.0 KLOC
File input and output	-	1.5 KLOC
Library reports	-	2.0 KLOC
Query and search	-	0.5 KLOC

The database size and application experience are very important in this project. The use of the software tool and the main storage is highly considerable. The virtual machine experience and its volatility can be kept low. All other cost drivers have nominal requirements. Use the COCOMO model to estimate the development effort and the development time.

Solution 4.3:

The LMS project can be considered an organic category project. The total size of the modules is 7 KLOC. The development effort and development time can be calculated as follows:

Development effort

Initial effort (E_i) $= 3.2 \times (7)^{1.05}$ = **24.6889 PM**

$EAF = 1.16 \times 0.82 \times 0.91 \times 1.06 \times 1.10 \times 0.87 =$ **0.8780**

Total effort $(E) = E_i * EAF$

$= 24.6889 \times 0.8780$

= **21.6785 PM**

Development time *(T)* = $2.5 \times (E)^{0.38}$ month

= $2.5\ (21.6785)^{0.38}$ month

= **8.0469 month**

Detailed COCOMO Model

- The detailed COCOMO model inherits all the features of the intermediate COCOMO model for the overall estimation of the project cost.
- The detailed COCOMO model uses different effort multipliers (cost drivers) for each phase of the project.
- Phase-wise effort multipliers provide better estimates than the intermediate model.
- The detailed COCOMO model defines five life cycle phases for effort distribution:
  - plan and requirement,
  - system design,
  - detailed design,
  - code and unit test, and
  - Integration and test.
- In the detailed COCOMO model, effort is calculated as a function of size in terms of KLOC and the value of a set of cost drivers according to each phase of the software life cycle.
- If the project size varies majorly from the value taken in the phase-wise distribution, then interpolation formula can be applied to find the more appropriate percentage value.
- The detailed COCOMO model illustrates the importance of recognizing different levels of predictability at each phase of the development cycle.

Example 4.4

Compute the phase-wise development effort for the problem discussed in Example 4.3.

Solution 4.4:

There are five components in the organic project discussed in Example 4.3: online data entry, data update, file input and output, library reports, query and search. The estimated effort *(E)* is 21.6785 PM. The total size is 7 KLOC, which is between 2 KLOC and 32 KLOC. Thus, the actual percentage of effort can be calculated as follows:

Plan and requirement (%) = 6 + (6 - 6) / (32 - 2) × 7 = 6%

Effort = 0.06 × 21.6785 PM = 1.30071 PM

System design = 16 + (16 - 16) / (32 - 2) × 7 = 16%

Effort = 0.16 × 21.6785 PM = 3.46856 PM

Detailed design = 24 + (26 - 24) / (32 - 2) × 7 = 25%

Effort = 0.25 × 21.6785 PM = 5.419625 PM

Code and unit test = 38 + (42 - 38) / (32 - 2) × 7 = 39%

Effort = 0.39 × 21.6785 PM = 8.454615 PM

Integration and test = 22 + (16 - 22) / (32 - 2) × 7 = 24%

= 0.24 × 21.6785 PM = 5.20284 PM

COCOMO I Model

- The basic, intermediate, and detailed COCOMO models are in the category of COCOMO I and these are collectively known as COCOMO 81.
- The COCOMO I models were developed to estimate the effort, schedule, and cost of a software project.
- The three variations of COCOMO I use different types of parameters for estimation.
- These models are helpful to produce repeatable estimations.
- It is unable to deal with exceptional conditions, such as exceptional teamwork, exceptional people involved in the estimation process, etc.
- Rough sizing of projects and inaccurate cost driver rating will result in an inaccurate estimation.

COCOMO II Model

- COCOMO II is an extension to the original COCOMO 81 model.
- Since the development of COCOMO 81, software development techniques have changed.
- These changes have changed the development processes from linear to spiral and one-go to incremental way for developing simple to complex projects.
- Programming techniques and technologies have moved from conventional to component-based software development.
- Thus, a new, enhanced COCOMO II model was proposed to overcome the limitations of COCOMO 81.

- It consists of sub-models, each one providing unique features for project planning and design process. These sub-models are called
  - The *applications composition model,*
  - *Early design model,*
  - *Post-architecture model,* and
  - *Reuse model.*

Application Composition Model

- The application composition model is best suitable for prototyping projects and projects where there is extensive reuse of components.
- It involves prototyping efforts to resolve unexpected risks related to GUI, device interaction, performance, or technology maturity.
- The estimation is based on the standard estimates of the developer productivity in object points/months and the use of CASE tools to support development.
- The effort estimation in an application is computed as follows:

$$\text{Effort } (E) = (\sum AP \times (1 - \%reuse/100))/ \text{ PROD}$$

- ∑AP is the sum of application points in the system to be developed. PROD is the productivity of programmers from the historical data rated on some scale.
- The PROD for the past projects is rated as very low = 4, low = 7, nominal = 13, high = 15, and very high = 50. Effort *(E)* will be measured in person-months (PM). %reuse is the percentage of reusable components that will be reused in the application.

Early Design Model

- The early design model is used when requirements are available but design has not yet commenced.
- It involves exploration of an alternative architecture or concept of operation. The estimation formula based on simple COCOMO is given as follows:

$$\text{Effort (E)} = A \times \text{Size}^{B} \times \text{EAF}$$

- *A* and *B* are constants. The value of *A* is fixed at 2.94 from a large data set.
- The value of *B* varies with the project size and it can range from 1.1 to 1.24, depending on the project complexity, process maturity level, development flexibility, risk resolution processes, and cohesion of the development team in an organization.
- EAF is the effort adjustment factor based on seven characteristics of projects and processes such as, product reliability and complexity (RCPX), reuse required (RUSE), platform difficulty (PDIF), personnel capability (PERS), personnel experience (PREX), schedule (SCED) and support facilities (FCIL).

Post-Architecture Model

- This model is used once the system architecture has been designed and sufficient information is available for the development and maintenance of the system.
- Estimation is determined using the estimation formula of early design model as follows:

  Effort (E) = $A \times \text{Size}^{B} \times \text{EAF}$

- SLOC or FP can be used for determining the project size.
- There are 17 instead of 7 multiplicative cost drivers.

Post-Architecture Model

- The following five factors determine the project's scaling exponent *B*. The value of these factors is added (for example, here it is 16) and their value in percentage (i.e. .16) is added to 1.01 to get a value of 1.17.

Reuse Model

- The reuse model focuses on system development with reuse.
- It helps to monitor the progress of reuse-based development and guides to the investment for producing reusable assets and reusing them in development.
- The cost of system development with reuse involves the cost incurred in design, implementation, generalization, testing, documentation, library support, and integration of reusable assents.
- The level of reuse in a system is defined as the total size of the system divided by the size of the reusable components in the system.

- The level of reuse can be used to monitor reuse and to estimate the cost, time, quality, and productivity.

Reuse Model

- The cost of system development can be assessed by the cost of system development with reuse and without reuse.
- The cost of software can be measured as:

Total cost (C_S) = Cost of the reused software + Cost of the software without reuse

= (Relative cost of the reused software × Reused software) + (Relative cost of the software without reuse × Software without reuse)

$= (C_R \times R_s) + (C_{RW} \times (1 - R_s))$

$= (C_R \times R_s) + (1 \times (1 - R_s))$

$= 1 + R_s (C_R - 1)$

Analytical techniques

- Halstead has proposed analytical estimation technique based upon certain assumptions about the project.
- From the science of Halstead, the length and volume of code can be measured. The length of a code is used to measure the length of the source code program. It is defined as follows:

$N = N1 + N2$

- Where $N1$ is the total number of operator occurrences, and $N2$ is the total number of operand occurrences.
- The volume metric is used to measure the amount of storage space required for the program. It is defined as:

$V = N \log (n1 + n2)$

- Where $n1$ is the number of distinct operators, and $n2$ is the number of distinct operands that occur in a program.
- Analytical estimation is manly useful for estimating software efforts in maintenance, scheduling, and reporting projects.