

System Analysis and Design

Systems development is systematic process which includes phases such as planning, analysis, design, deployment, and maintenance. Here, the primarily be focus will be on

- Systems analysis
- Systems design

Systems Analysis

- It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.
- System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.
- Analysis specifies **what the system should do**.

Systems Design

- It is a process of planning a new business system or replacing an existing system by defining its components or modules to satisfy the specific requirements. Before planning, you need to understand the old system thoroughly and determine how computers can best be used in order to operate efficiently.
- System Design focuses on **how to accomplish the objective of the system.**
- System Analysis and Design (SAD) mainly focuses on –
 - Systems
 - Processes
 - Technology

What is a System?

- The word System is derived from Greek word Systema, which means an organized relationship between any set of components to achieve some common cause or objective.
- *A system is “an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal.”*

Constraints of a System

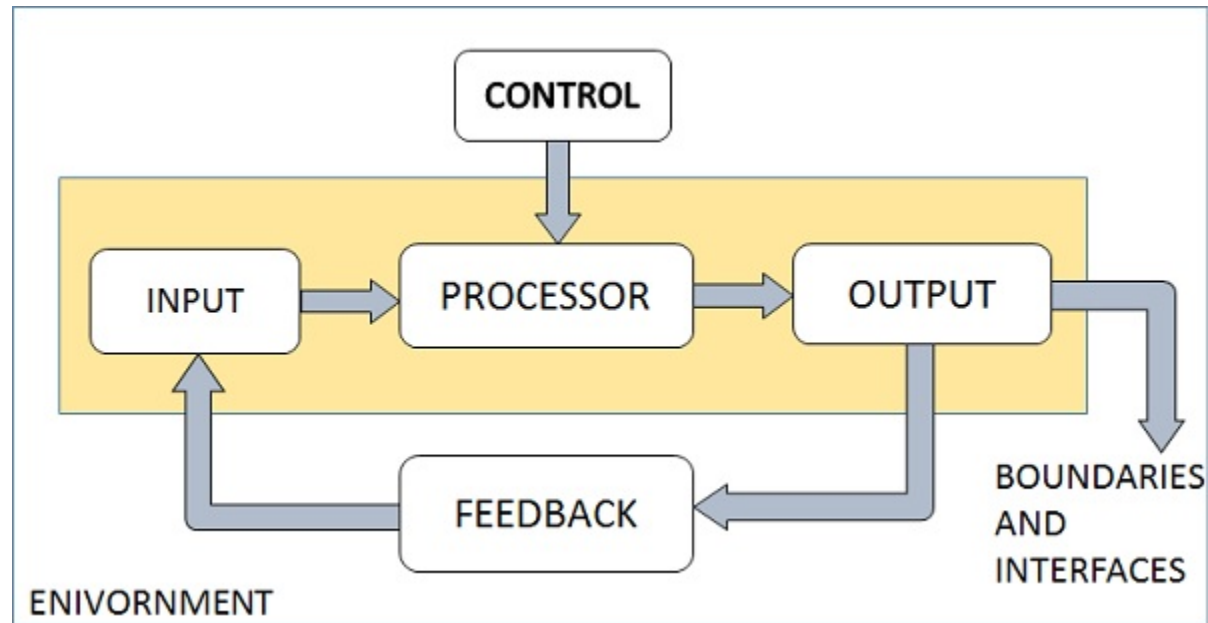
- A system must have three basic constraints –
- A system must have some **structure and behavior** which is designed to achieve a predefined objective.
- **Interconnectivity** and **interdependence** must exist among the system components.
- The **objectives of the organization** have a **higher priority** than the objectives of its subsystems.
- For example, traffic management system, payroll system, automatic library system, human resources information system.

Properties of a System

- A system has the following properties –
- **Organization**
- Organization implies structure and order. It is the arrangement of components that helps to achieve predetermined objectives.
- **Interaction**
- It is defined by the manner in which the components operate with each other.
- For example, in an organization, purchasing department must interact with production department and payroll with personnel department.
- **Interdependence**
- Interdependence means how the components of a system depend on one another. For proper functioning, the components are coordinated and linked together according to a specified plan. The output of one subsystem is the required by other subsystem as input.
- **Integration**
- Integration is concerned with how a system components are connected together. It means that the parts of the system work together within the system even if each part performs a unique function.
- **Central Objective**
- The objective of system must be central. It may be real or stated. It is not uncommon for an organization to state an objective and operate to achieve another.
- The users must know the main objective of a computer application early in the analysis for a successful design and conversion.

Elements of a System

The following diagram shows the elements of a system –



Elements of a System

- **Outputs and Inputs**

- The main aim of a system is to produce an output which is useful for its user.
- Inputs are the information that enters into the system for processing.
- Output is the outcome of processing.

- **Processor(s)**

- The processor is the element of a system that involves the actual transformation of input into output.
- It is the operational component of a system. Processors may modify the input either totally or partially, depending on the output specification.
- As the output specifications change, so does the processing. In some cases, input is also modified to enable the processor for handling the transformation.

- **Control**

- The control element guides the system.
- It is the decision-making subsystem that controls the pattern of activities governing input, processing, and output.
- The behavior of a computer System is controlled by the Operating System and software. In order to keep system in balance, what and how much input is needed is determined by Output Specifications.

Elements of a System

- **Feedback**

- Feedback provides the control in a dynamic system.
- Positive feedback is routine in nature that encourages the performance of the system.
- Negative feedback is informational in nature that provides the controller with information for action.

- **Environment**

- The environment is the “supersystem” within which an organization operates.
- It is the source of external elements that strike on the system.
- It determines how a system must function. For example, vendors and competitors of organization’s environment, may provide constraints that affect the actual performance of the business.

- **Boundaries and Interface**

- A system should be defined by its boundaries. Boundaries are the limits that identify its components, processes, and interrelationship when it interfaces with another system.
- Each system has boundaries that determine its sphere of influence and control.
- The knowledge of the boundaries of a given system is crucial in determining the nature of its interface with other systems for successful design.

Elements of a System

Physical or Abstract Systems

- Physical systems are tangible entities. We can touch and feel them.
- Physical System may be static or dynamic in nature. For example, desks and chairs are the physical parts of computer center which are static. A programmed computer is a dynamic system in which programs, data, and applications can change according to the user's needs.
- Abstract systems are non-physical entities or conceptual that may be formulas, representation or model of a real system.

Open or Closed Systems

- An open system must interact with its environment. It receives inputs from and delivers outputs to the outside of the system. For example, an information system which must adapt to the changing environmental conditions.
- A closed system does not interact with its environment. It is isolated from environmental influences. A completely closed system is rare in reality.

Elements of a System

Adaptive and Non Adaptive System

- Adaptive System responds to the change in the environment in a way to improve their performance and to survive. For example, human beings, animals.
- Non Adaptive System is the system which does not respond to the environment. For example, machines.

Permanent or Temporary System

- Permanent System persists for long time. For example, business policies.
- Temporary System is made for specified time and after that they are demolished. For example, A DJ system is set up for a program and it is disassembled after the program.

Natural and Manufactured System

- Natural systems are created by the nature. For example, Solar system, seasonal system.
- Manufactured System is the man-made system. For example, Rockets, dams, trains.

Elements of a System

Deterministic or Probabilistic System

- Deterministic system operates in a predictable manner and the interaction between system components is known with certainty. For example, two molecules of hydrogen and one molecule of oxygen makes water.
- Probabilistic System shows uncertain behavior. The exact output is not known. For example, Weather forecasting, mail delivery.

Social, Human-Machine, Machine System

- Social System is made up of people. For example, social clubs, societies.
- In Human-Machine System, both human and machines are involved to perform a particular task. For example, Computer programming.
- Machine System is where human interference is neglected. All the tasks are performed by the machine. For example, an autonomous robot.

Elements of a System

Man–Made Information Systems

- It is an interconnected set of information resources to manage data for particular organization, under Direct Management Control (DMC).
- This system includes hardware, software, communication, data, and application for producing information according to the need of an organization.

Man-made information systems are divided into three types –

- **Formal Information System** – It is based on the flow of information in the form of memos, instructions, etc., from top level to lower levels of management.
- **Informal Information System** – This is employee based system which solves the day to day work related problems.
- **Computer Based System** – This system is directly dependent on the computer for managing business applications. For example, automatic library system, railway reservation system, banking system, etc.

Systems Models

Schematic Models

- A schematic model is a 2-D chart that shows system elements and their linkages.
- Different arrows are used to show information flow, material flow, and information feedback.

Flow System Models

- A flow system model shows the orderly flow of the material, energy, and information that hold the system together.
- Program Evaluation and Review Technique (PERT), for example, is used to abstract a real world system in model form.


Static System Models

- They represent one pair of relationships such as *activity–time* or *cost–quantity*.
- The Gantt chart, for example, gives a static picture of an activity-time relationship.

Dynamic System Models

- Business organizations are dynamic systems. A dynamic model approximates the type of organization or application that analysts deal with.
- It shows an ongoing, constantly changing status of the system. It consists of –
 - Inputs that enter the system
 - The processor through which transformation takes place
 - The program(s) required for processing
 - The output(s) that result from processing.

Categories of Information

Volume of Information	Type of Information	Information Level	Management Level	System Support
Low Consensed	Unstructured		Upper	DSS
Medium Moderately Processed	Moderately Structured		Middle	MIS
Large Detail Reports	Highly Structured		Lower	DPS

Categories of Information

Strategic Information

- This information is required by topmost management for long range planning policies for next few years. For example, trends in revenues, financial investment, and human resources, and population growth.
- This type of information is achieved with the aid of Decision Support System (DSS).

Managerial Information

- This type of Information is required by middle management for short and intermediate range planning which is in terms of months. For example, sales analysis, cash flow projection, and annual financial statements.
- It is achieved with the aid of Management Information Systems (MIS).

Operational information

- This type of information is required by low management for daily and short term planning to enforce day-to-day operational activities. For example, keeping employee attendance records, overdue purchase orders, and current stocks available.
- It is achieved with the aid of Data Processing Systems (DPS).

Software Project Management

The main goal of software project management is to enable a group of developers to work effectively towards the successful completion of a project.

Organization of this Lecture:

- Introduction to Project Planning
- Software Cost Estimation
 - Cost Estimation Models
 - Software Size Metrics
 - Empirical Estimation
 - Heuristic Estimation
 - COCOMO
- Staffing Level Estimation
- Effect of Schedule Compression on Cost
- Summary

Introduction

- Many software projects fail:
 - due to faulty project management practices:
 - It is important to learn different aspects of software project management.

Introduction

- Goal of software project management:
 - enable a group of engineers to work efficiently towards successful completion of a software project.

Responsibility of project managers

- Project proposal writing,
- Project cost estimation,
- Scheduling,
- Project staffing,
- Project monitoring and control,
- Software configuration management,
- Risk management,
- Managerial report writing and presentations, etc.

Introduction

- A project manager's activities are varied.
 - can be broadly classified into:
 - project planning,
 - project monitoring and control activities.

Project Planning

- Once a project is found to be feasible,
 - project managers undertake project planning.

Project Planning Activities

- Estimation:
 - Effort, cost, resource, and project duration
- Project scheduling:
- Staff organization:
 - staffing plans
- Risk handling:
 - identification, analysis, and abatement procedures
- Miscellaneous plans:
 - quality assurance plan, configuration management plan, etc.

Project planning

- Requires utmost care and attention --- commitments to unrealistic time and resource estimates result in:
 - irritating delays.
 - customer dissatisfaction
 - adverse affect on team morale
 - poor quality work
 - project failure.

Sliding Window Planning

- Involves project planning over several stages:
 - protects managers from making big commitments too early.
 - More information becomes available as project progresses.
 - Facilitates accurate planning

SPMP Document

- After planning is complete:
 - Document the plans:
 - in a Software Project Management Plan(SPMP) document.

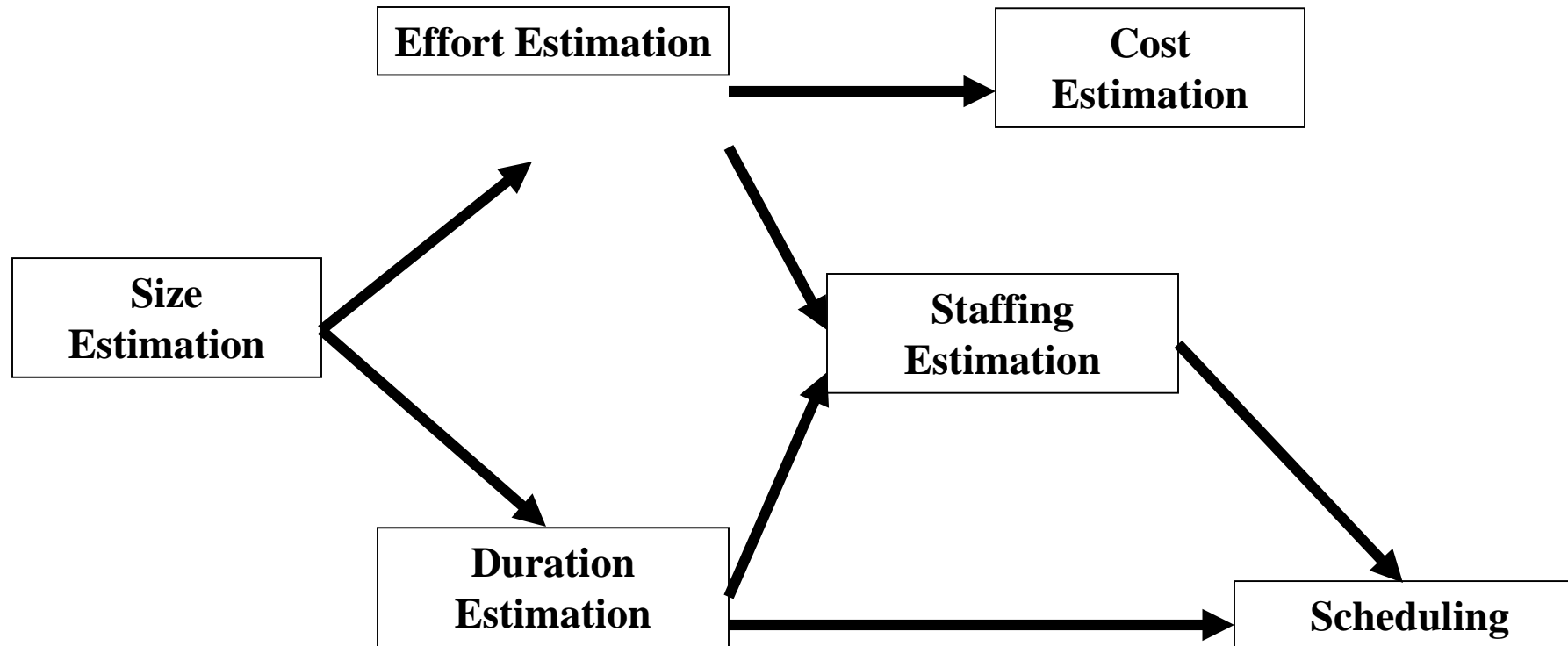
Organization of SPMP Document

- **Introduction** (Objectives, Major Functions, Performance Issues, Management and Technical Constraints)
- **Project Estimates** (Historical Data, Estimation Techniques, Effort, Cost, and Project Duration Estimates)
- **Project Resources Plan** (People, Hardware and Software, Special Resources)
- **Schedules** (Work Breakdown Structure, Task Network, Gantt Chart Representation, PERT Chart Representation)
- **Risk Management Plan** (Risk Analysis, Risk Identification, Risk Estimation, Abatement Procedures)
- **Project Tracking and Control Plan**
- **Miscellaneous Plans** (Process Tailoring, Quality Assurance)

Software Cost Estimation

- Determine [size](#) of the product.
- From the size estimate,
 - determine the [effort](#) needed.
- From the effort estimate,
 - determine [project duration, and cost](#).

Software Cost Estimation



Software Cost Estimation

- Three main approaches to estimation:
 - Empirical
 - Heuristic
 - Analytical

Software Cost Estimation Techniques

- Empirical techniques:
 - an educated guess based on past experience.
- Heuristic techniques:
 - assume that the characteristics to be estimated can be expressed in terms of some mathematical expression.
- Analytical techniques:
 - derive the required results starting from certain simple assumptions.

Software Size Metrics

- LOC (Lines of Code):
 - Simplest and most widely used metric.
 - Comments and blank lines should not be counted.

Disadvantages of Using LOC

- Size can vary with coding style.
- Focuses on coding activity alone.
- Correlates poorly with quality and efficiency of code.
- Penalizes higher level programming languages, code reuse, etc.

Disadvantages of Using LOC (cont...)

- Measures lexical/textual complexity only.
 - does not address the issues of structural or logical complexity.
- Difficult to estimate LOC from problem description.
 - So not useful for project planning

Function Point

- Conceptually, the function point metric is based on the idea that a software product supporting many features would certainly be of larger size than a product with less number of features.

Function Point Metric

- Overcomes some of the shortcomings of the LOC metric
- Proposed by Albrecht in early 80's:
 - $FP = 4 \times \#inputs + 5 \times \#Outputs + 4 \times \#inquiries + 10 \times \#files + 10 \times \#interfaces$
- Input:
 - A set of related inputs is counted as one input.

Function Point Metric

- Output:
 - A set of related outputs is counted as one output.
- Inquiries:
 - Each user query type is counted.
- Files:
 - Files are logically related data and thus can be data structures or physical files.
- Interface:
 - Data transfer to other systems.

Function Point Metric (CONT.)

- Suffers from a major drawback:
 - the size of a function is considered to be independent of its complexity.
- Extend function point metric:
 - Feature Point metric:
 - considers an extra parameter:
 - Algorithm Complexity.

Function Point Metric_(CONT.)

- Proponents claim:
 - FP is language independent.
 - Size can be easily derived from problem description
- Opponents claim:
 - it is subjective --- Different people can come up with different estimates for the same problem.

Software Engineering | Calculation of Function Point (FP)

[Function Point \(FP\)](#) is an element of software development which helps to approximate the cost of development early in the process. It may measures functionality from user's point of view.

Counting Function Point (FP):

•**Step-1:** $F = 14 * \text{scale}$

Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:

0 - No Influence 1 - Incidental 2 - Moderate 3 - Average 4 - Significant 5 - Essential

•**Step-2:** Calculate Complexity Adjustment Factor (CAF). $\text{CAF} = 0.65 + (0.01 * F)$

•**Step-3:** Calculate Unadjusted Function Point (UFP).

TABLE (Required)

•**Step-4:** Calculate Function Point. $\text{FP} = \text{UFP} * \text{CAF}$

Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.
User Input = 50 User Output = 40 User Inquiries = 35 User Files = 6 External Interface = 4

Explanation:

- **Step-1:** As complexity adjustment factor is average (given in question), hence,

$$\begin{aligned}\text{scale} &= 3. \\ F &= 14 * 3 = 42\end{aligned}$$

- **Step-2:**

$$\text{CAF} = 0.65 + (0.01 * 42) = 1.07$$

- **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

$$\text{UFP} = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628$$

- **Step-4:**

$$\text{Function Point} = 628 * 1.07 = 671.96$$

This is the required answer.

Empirical Size Estimation Techniques

- **Expert Judgement:**
 - An euphemism for guess made by an expert.
 - Suffers from individual bias.
- **Delphi Estimation:**
 - overcomes some of the problems of expert judgement.

Expert judgement

- Experts divide a software product into component units:
 - e.g. GUI, database module, data communication module, billing module, etc.
- Add up the guesses for each of the components.

Delphi Estimation:

- Team of Experts and a coordinator.
- Experts carry out estimation independently:
 - mention the rationale behind their estimation.
 - coordinator notes down any extraordinary rationale:
 - circulates among experts using SRS.

Delphi Estimation:

- Experts re-estimate.
- Experts never meet each other
 - to discuss their viewpoints.

Heuristic Estimation Techniques

- Single Variable Model:

- Parameter to be Estimated = $C_1(\text{Estimated Characteristic})^{d_1}$

- Multivariable Model:

- Assumes that the parameter to be estimated depends on more than one characteristic.
 - Parameter to be Estimated = $C_1(\text{Estimated Characteristic})^{d_1} + C_2(\text{Estimated Characteristic})^{d_2} + \dots$
 - Usually more accurate than single variable models.

$$\text{Estimated Resource} = c_1 \times p^{d_1} + c_2 \times p^{d_2} + \dots$$

Basic COCOMO Model

- COCOMO (COnstructive COst MOdel) proposed by Boehm. Basic+ Intermediate + complete
- Divides software product developments into 3 categories:
 - Organic
 - Semidetached
 - Embedded

One person month is the effort an individual can typically put in a month. The person-month estimate implicitly takes into account the productivity losses that normally occur due to time lost in holidays, weekly offs, coffee breaks, etc.

Person-month (PM) is considered to be an appropriate unit for measuring effort, because developers are typically assigned to a project for a certain number of months.

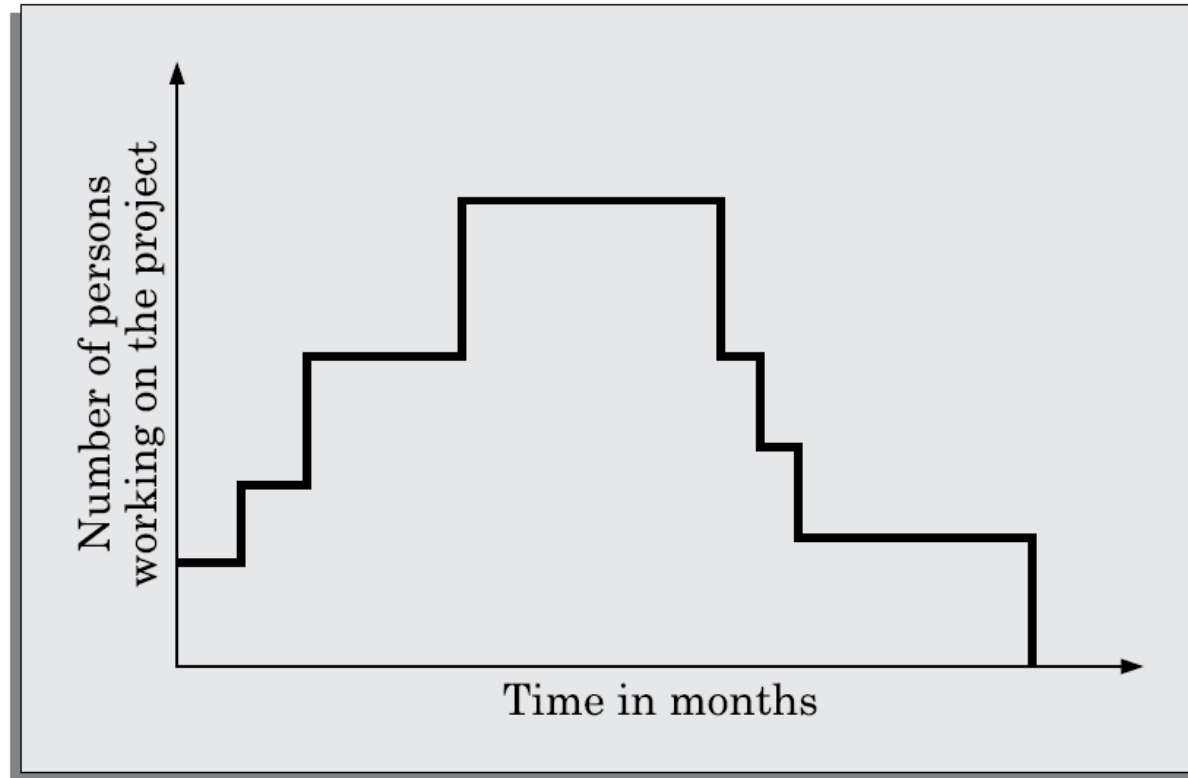


FIGURE 3.3 Person-month curve.

COCOMO Product classes

- Roughly correspond to:
 - application, utility and system programs respectively.
 - Data processing and scientific programs are considered to be **application programs**.
 - Compilers, linkers, editors, etc., are **utility programs**.
 - Operating systems and real-time system programs, etc. are **system programs**.

Organic: We can classify a development project to be 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 classify to be of semidetached type, if the development team 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 hardware, or if stringent regulations on the operational procedures exist. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

Elaboration of Product classes

- Organic:
 - Relatively small groups
 - working to develop well-understood applications.
- Semidetached:
 - Project team consists of a mixture of experienced and inexperienced staff.
- Embedded:
 - The software is strongly coupled to complex hardware, or real-time systems.

COCOMO Model (CONT.)

- For each of the three product categories:
 - From size estimation (in KLOC), Boehm provides equations to predict:
 - project duration in months
 - effort in programmer-months
- Boehm obtained these equations:
 - examined historical data collected from a large number of actual projects.

COCOMO Model (CONT.)

- Software cost estimation is done through three stages:
 - Basic COCOMO,
 - Intermediate COCOMO,
 - Complete COCOMO.

Basic COCOMO Model (CONT.)

- Gives only an approximate estimation:
 - $\text{Effort} = a1 \text{ (KLOC)}^{a2}$
 - $\text{Tdev} = b1 \text{ (Effort)}^{b2}$
 - KLOC is the estimated kilo lines of source code,
 - $a1, a2, b1, b2$ are constants for different categories of software products,
 - Tdev is the estimated time to develop the software in months,
 - Effort estimation is obtained in terms of person months (PMs).

Development Effort Estimation

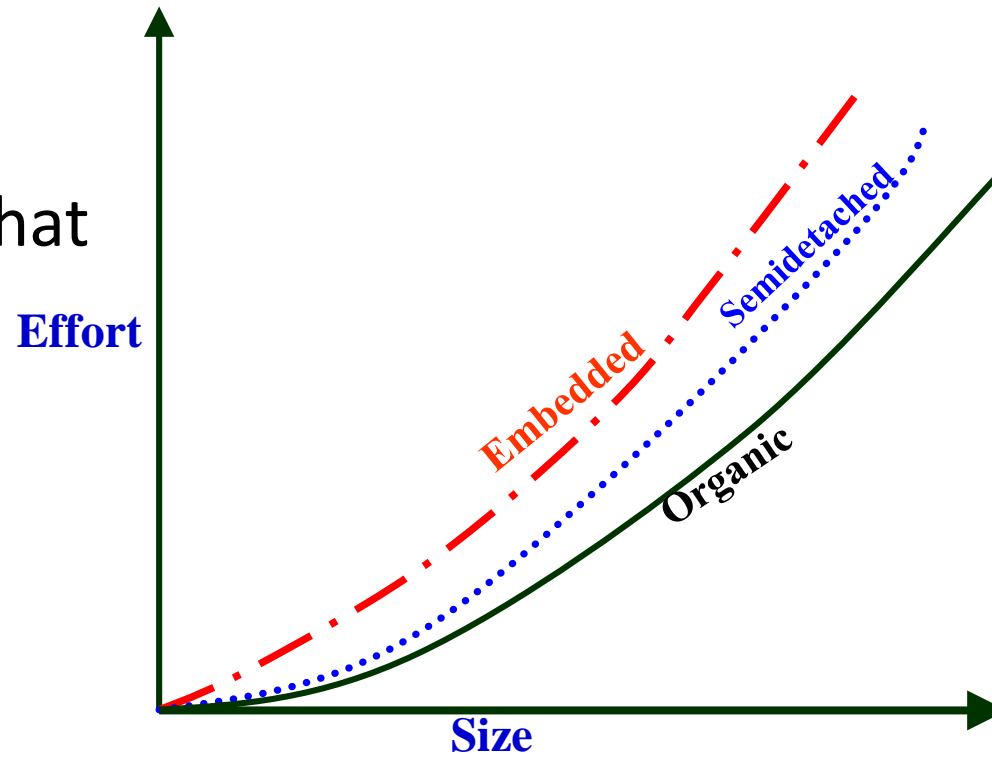
- Organic :
 - $\text{Effort} = 2.4 (\text{KLOC})^{1.05} \text{ PM}$
- Semi-detached:
 - $\text{Effort} = 3.0 (\text{KLOC})^{1.12} \text{ PM}$
- Embedded:
 - $\text{Effort} = 3.6 (\text{KLOC})^{1.20} \text{ PM}$

Development Time Estimation

- Organic:
 - $T_{dev} = 2.5 (\text{Effort})^{0.38}$ Months
- Semi-detached:
 - $T_{dev} = 2.5 (\text{Effort})^{0.35}$ Months
- Embedded:
 - $T_{dev} = 2.5 (\text{Effort})^{0.32}$ Months

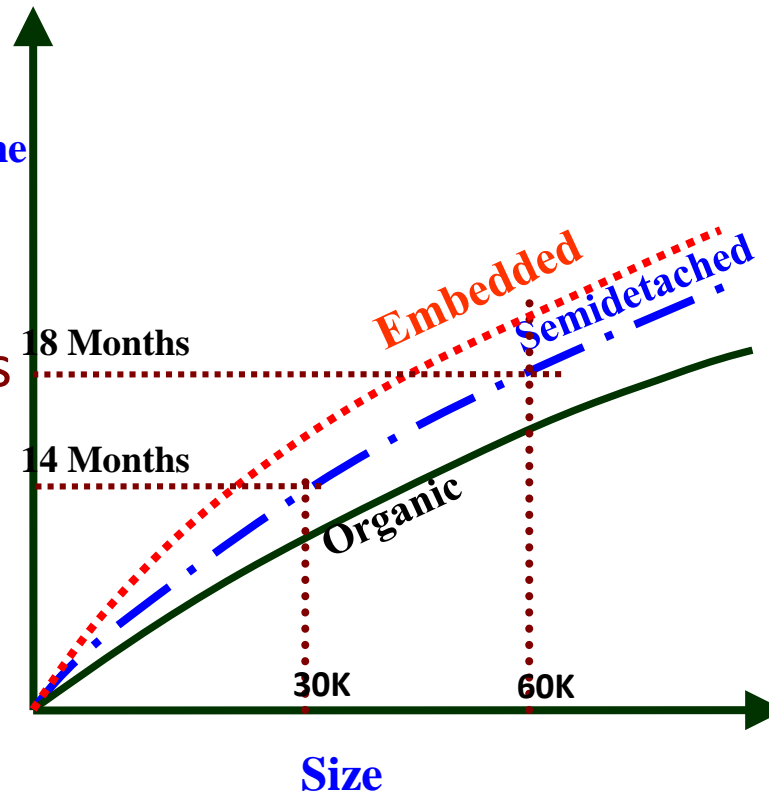
Basic COCOMO Model (CONT.)

- Effort is somewhat super-linear in problem size.



Basic COCOMO Model (CONT.)

- Development time
 - sublinear function of product size.
- When product size increases two times,
 - development time does not double.
- Time taken:
 - almost same for all the three product categories.



Basic COCOMO Model (CONT.)

- Development time does not increase linearly with product size:
 - For larger products more parallel activities can be identified:
 - can be carried out simultaneously by a number of engineers.

Basic COCOMO Model (CONT.)

- 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, semi-detached, or embedded type.
 - There is more scope for parallel activities for system and application programs,
 - than utility programs.

Example

PROBLEM 3.2 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 a software developer is ₹15,000 per month. Determine the effort required to develop the software product, the nominal development time, and the cost to develop the product.

.....

- The size of an organic software product has been estimated to be 32,000 lines of source code.
- Effort = $2.4 \times (32)^{1.05} = 91$ PM
- Nominal development time = $2.5 \times (91)^{0.38} = 14$ months

Staff cost required to develop the product = $91 \times ₹15,000 = ₹1,465,000$

Intermediate COCOMO

- Basic COCOMO model assumes
 - effort and development time depend on product size alone.
- However, several parameters affect effort and development time:
 - Reliability requirements
 - Availability of CASE tools and modern facilities to the developers
 - Size of data to be handled

The intermediate COCOMO model refines the initial estimate obtained using the basic COCOMO expressions by scaling the estimate up or down based on the evaluation of a set of attributes of software development.

Intermediate COCOMO

- For accurate estimation,
 - the effect of all relevant parameters must be considered:
 - **Intermediate COCOMO model** recognizes this fact:
 - refines the initial estimate obtained by the basic COCOMO by using a set of 15 cost drivers (multipliers).

Intermediate COCOMO_(CONT.)

- If modern programming practices are used,
 - initial estimates are scaled downwards.
- If there are stringent reliability requirements on the product :
 - initial estimate is scaled upwards.

Intermediate COCOMO_(CONT.)

- Rate different parameters on a scale of one to three:
 - Depending on these ratings,
 - multiply cost driver values with the estimate obtained using the basic COCOMO.

Intermediate COCOMO_(CONT.)

- Cost driver classes:
 - Product: Inherent complexity of the product, reliability requirements of the product, etc.
 - Computer: Execution time, storage requirements, etc.
 - Personnel: Experience of personnel, etc.
 - Development Environment: Sophistication of the tools used for software development.

Classification of Cost Drivers and their attributes:

- **(i) Product attributes -**
- Required software reliability extent
- Size of the application database
- The complexity of the product
- **Hardware attributes -**
- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time
- **Personnel attributes -**
- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience
- **Project attributes -**
- Use of software tools
- Application of software engineering methods
- Required development schedule

The cost drivers are divided into four categories:

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	..
DATA	..	0.94	1.00	1.08	1.16	..
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	1.00	1.11	1.30	1.66
STOR	1.00	1.06	1.21	1.56
VIRT	..	0.87	1.00	1.15	1.30	..
TURN	..	0.87	1.00	1.07	1.15	..

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	..
AEXP	1.29	1.13	1.00	0.91	0.82	..
PCAP	1.42	1.17	1.00	0.86	0.70	..
VEXP	1.21	1.10	1.00	0.90	..	**
LEXP	1.14	1.07	1.00	0.95
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	..
TOOL	1.24	1.10	1.00	0.91	0.83	..
SCED	1.23	1.08	1.00	1.04	1.10	..

- **Intermediate COCOMO equation:**

- $$E = a_i (\text{KLOC})^{b_i} \text{EAF}$$
$$D = c_i (E)^{d_i}$$

Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Shortcoming of basic and intermediate COCOMO models

- Both models:
 - consider a software product as a single homogeneous entity:
 - However, most large systems are made up of several smaller sub-systems.
 - Some sub-systems may be considered as organic type, some may be considered embedded, etc.
 - for some the reliability requirements may be high, and so on.

Detailed COCOMO Model:

- **Detailed COCOMO Model:** Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost drivers effect on each method of the software
- engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed COCOMO , the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.
- The Six phases of detailed COCOMO are:
 - Planning and requirements
 - System structure
 - Complete structure
 - Module code and test
 - Integration and test
 - Cost Constructive model
- The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

Complete COCOMO

- Cost of each sub-system is estimated separately.
- Costs of the sub-systems are added to obtain total cost.
- Reduces the margin of error in the final estimate.

Complete COCOMO Example

- A Management Information System (MIS) for an organization having offices at several places across the country:
 - Database part (semi-detached)
 - Graphical User Interface (GUI) part (organic)
 - Communication part (embedded)
- Costs of the components are estimated separately:
 - summed up to give the overall cost of the system.

COCOMO II

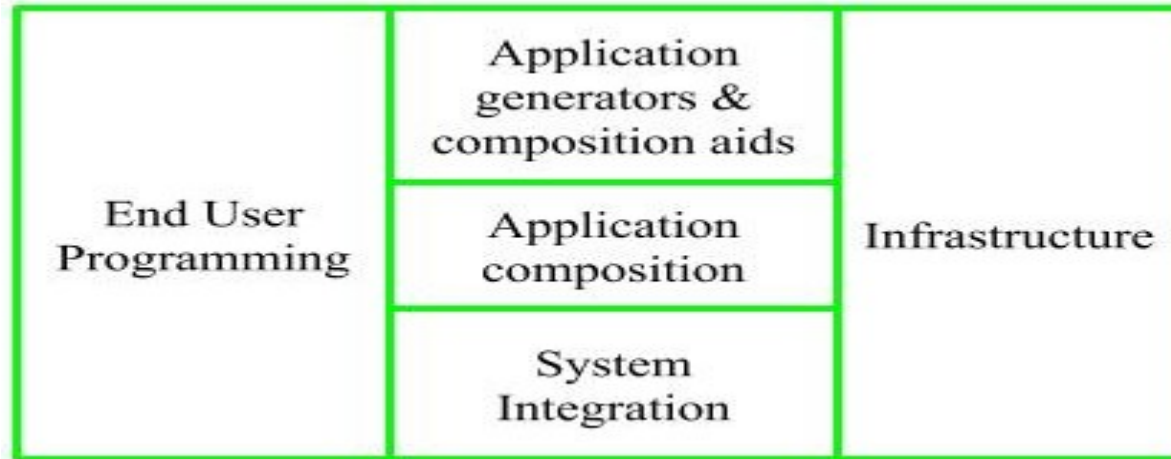
- Difference between COCOMO 1 and COCOMO 2
- **COCOMO 1 Model:**
The Constructive Cost Model was first developed by Barry W. Boehm. The model is for estimating effort, cost, and schedule for software projects. It is also called as Basic COCOMO. This model is used to give an approximate estimate of the various parameters of the project. Example of projects based on this model is business system, payroll management system and inventory management systems.
- **COCOMO 2 Model:**
The COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and is developed at the University of Southern California. This model calculates the development time and effort taken as the total of the estimates of all the individual subsystems. In this model, whole software is divided into different modules. Example of projects based on this model is Spreadsheets and report generator.

Difference between COCOMO 1 and COCOMO 2:

COCOMO I	COCOMO II
COCOMO I is useful in the waterfall models of the software development cycle.	COCOMO II is useful in non-sequential, rapid development and reuse models of software.
It provides estimates of effort and schedule.	It provides estimates that represent one standard deviation around the most likely estimate.
This model is based upon the linear reuse formula.	This model is based upon the non linear reuse formula
This model is also based upon the assumption of reasonably stable requirements.	This model is also based upon reuse model which looks at effort needed to understand and estimate.
Effort equation's exponent is determined by 3 development modes.	Effort equation's exponent is determined by 5 scale factors.
Development begins with the requirements assigned to the software.	It follows a spiral type of development.
Number of submodels in COCOMO I is 3 and 15 cost drivers are assigned	In COCOMO II, Number of submodel are 4 and 17 cost drivers are assigned
Size of software stated in terms of Lines of code	Size of software stated in terms of Object points, function points and lines of code

COCOMO II Model

COCOMO-II is the revised version of the **original Cocomo (Constructive Cost Model)** and is developed at University of Southern California. It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity. It consists of three sub-models:



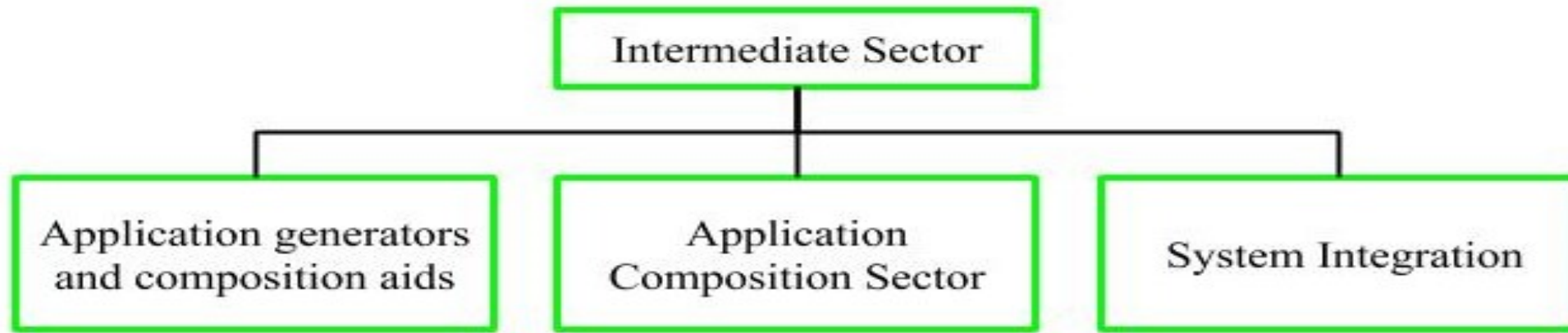
1. End User Programming:

Application generators are used in this sub-model. End user write the code by using these application generators.

Example – Spreadsheets, report generator, etc.

2. Intermediate Sector:

COCOMO II



- **(a). Application Generators and Composition Aids –**
This category will create largely prepackaged capabilities for user programming. Their product will have many reusable components. Typical firms operating in this sector are Microsoft, Lotus, Oracle, IBM, Borland, Novell.
- **(b). Application Composition Sector –**
This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, domain specific components such as financial, medical or industrial process control packages.

COCOMO II

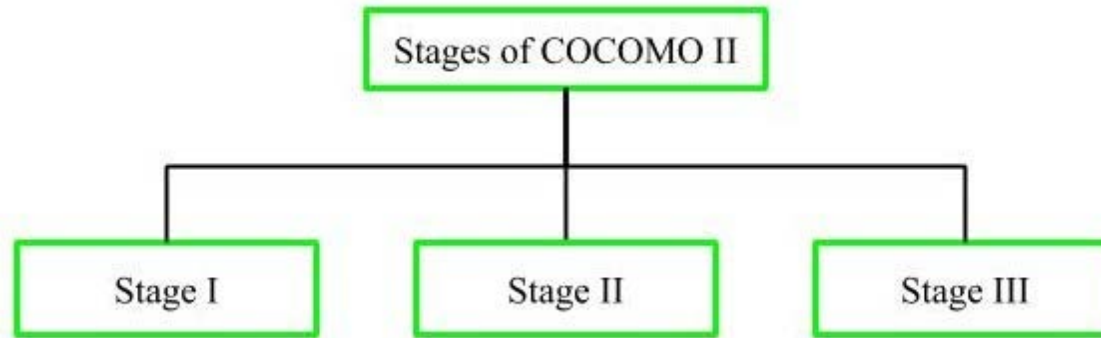
- **(c). System Integration –**

This category deals with large scale and highly embedded systems.

3. Infrastructure Sector:

This category provides infrastructure for the software development like Operating System, Database Management System, User Interface Management System, Networking System, etc.

Stages of COCOMO II:



1. **Stage-I:**

It supports estimation of prototyping. For this it uses *Application Composition Estimation Model*. This model is used for the prototyping stage of application generator and system integration.

2. **Stage-II:**

It supports estimation in the early design stage of the project, when we less know about it. For this it uses *Early Design Estimation Model*. This model is used in early design stage of application generators, infrastructure, system integration.

3. **Stage-III:**

It supports estimation in the post architecture stage of a project. For this it uses *Post Architecture Estimation Model*. This model is used after the completion of the detailed architecture of application generator, infrastructure, system integration.

Attention reader! Don't stop learning now. Get hold of all the important CS Theory concepts for SDE interviews with the [CS Theory Course](#) at a student-friendly price and become industry ready.

Staffing Level Estimation

- Number of personnel required during any development project:
 - not constant.
- Norden in 1958 analyzed many R&D projects, and observed:
 - Rayleigh curve represents the number of full-time personnel required at any time.

Norden concluded that the staffing pattern for any R&D project starting from a low level, increases until it reaches a peak value. It then starts to diminish. This pattern can be approximated by the Rayleigh distribution curve (see Fig. 2.7).

Example3: Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

- **Solution:** The basic COCOMO equation takes the form:

- $$\begin{aligned} \text{Effort} &= a_1 * (\text{KLOC})^{a_2} \text{ PM} \\ \text{Tdev} &= b_1 * (\text{efforts})^{b_2} \text{ Months} \\ \text{Estimated Size of project} &= 400 \text{ KLOC} \end{aligned}$$

- **(i)Organic Mode**

- $$\begin{aligned} E &= 2.4 * (400)^{1.05} = 1295.31 \text{ PM} \\ D &= 2.5 * (1295.31)^{0.38} = 38.07 \text{ PM} \end{aligned}$$

- **(ii)Semidetached Mode**

- $$\begin{aligned} E &= 3.0 * (400)^{1.12} = 2462.79 \text{ PM} \\ D &= 2.5 * (2462.79)^{0.35} = 38.45 \text{ PM} \end{aligned}$$

- **(iii) Embedded Mode**

- $$\begin{aligned} E &= 3.6 * (400)^{1.20} = 4772.81 \text{ PM} \\ D &= 2.5 * (4772.8)^{0.32} = 38 \text{ PM} \end{aligned}$$

Example4: A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

Solution: The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence $E = 3.0(200)1.12 = 1133.12\text{PM}$

$D = 2.5(1133.12)0.35 = 29.3\text{PM}$

Average Staff Size (SS) = $\frac{E}{D}$ Persons

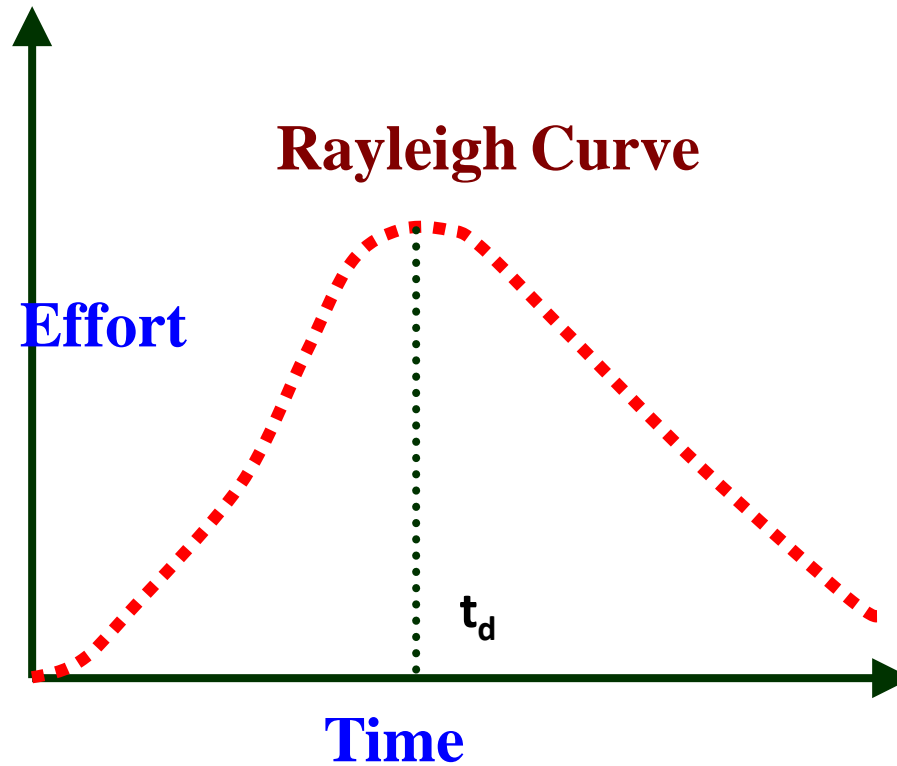
$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{\text{KLOC}}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

$$P = 176 \text{ LOC/PM}$$

Rayleigh Curve

- Rayleigh curve is specified by two parameters:
 - t_d the time at which the curve reaches its maximum
 - K the total area under the curve.
- $L=f(K, t_d)$



Norden represented the Rayleigh curve by the following equation:

$$E = \frac{K}{t_d^2} \times t \times e^{\frac{-t^2}{2t_d^2}}$$

Putnam's Work:

- In 1976, Putnam studied the problem of staffing of software projects:
 - observed that the level of effort required in software development efforts has a similar envelope.
 - found that the Rayleigh-Norden curve
 - relates the number of delivered lines of code to effort and development time.

Putnam's Work_(CONT.):

- Putnam analyzed a large number of army projects, and derived the expression:

$$L = C_k K^{1/3} t_d^{4/3}$$

- K is the effort expended and L is the size in KLOC.
- t_d is the time to develop the software.
- C_k is the state of technology constant
 - reflects factors that affect programmer productivity.

Putnam's Work_(CONT.):

- $C_k=2$ for poor development environment
 - no methodology, poor documentation, and review, etc.
- $C_k=8$ for good software development environment
 - software engineering principles used
- $C_k=11$ for an excellent environment

Rayleigh Curve

- Very small number of engineers are needed at the beginning of a project
 - carry out planning and specification.
- As the project progresses:
 - more detailed work is required,
 - number of engineers slowly increases and reaches a peak.

Rayleigh Curve

- Putnam observed that:
 - the time at which the Rayleigh curve reaches its maximum value
 - corresponds to system testing and product release.
 - After system testing,
 - the number of project staff falls till product installation and delivery.

Rayleigh Curve

- From the Rayleigh curve observe that:
 - approximately 40% of the area under the Rayleigh curve is to the left of t_d
 - and 60% to the right.

Effect of Schedule Change on Cost

- Using the Putnam's expression for L,

$$K = L^3 / C_k^3 t_d^4$$

$$\text{Or, } K = C / t_d^4$$

- For the same product size, $C = L^3 / C_k^3$ is a constant.
- Or, $K_1 / K_2 = t_{d2}^4 / t_{d1}^4$

Effect of Schedule Change on Cost_(CONT.)

- Observe:
 - a relatively small compression in delivery schedule
 - can result in substantial penalty on human effort.
- Also, observe:
 - benefits can be gained by using fewer people over a somewhat longer time span.

Example

- If the estimated development time is 1 year, then in order to develop the product in 6 months,
 - the total effort and hence the cost increases 16 times.
 - In other words,
 - the relationship between effort and the chronological delivery time is highly nonlinear.

Effect of Schedule Change on Cost_(CONT.)

- Putnam model indicates extreme penalty for schedule compression
 - and extreme reward for expanding the schedule.
- Putnam estimation model works reasonably well for very large systems,
 - but seriously overestimates the effort for medium and small systems.

Effect of Schedule Change on Cost_(CONT.)

- Boehm observed:
 - “There is a limit beyond which the schedule of a software project cannot be reduced by buying any more personnel or equipment.”
 - This limit occurs roughly at 75% of the nominal time estimate.

Effect of Schedule Change on Cost

(CONT.)

- If a project manager accepts a customer demand to compress the development time by more than 25%
 - very unlikely to succeed.
 - every project has only a limited amount of parallel activities
 - sequential activities cannot be speeded up by hiring any number of additional engineers.
 - many engineers have to sit idle.

Jensen Model

- Jensen model is very similar to Putnam model.
 - attempts to soften the effect of schedule compression on effort
 - makes it applicable to smaller and medium sized projects.

Jensen Model

- Jensen proposed the equation:

- $L = C_{te} t_d K^{1/2}$

- Where,

- C_{te} is the effective technology constant,
- t_d is the time to develop the software, and
- K is the effort needed to develop the software.

$$\frac{K_1}{K_2} = \frac{t_{d_2}^2}{t_{d_1}^2}$$

Organization Structure

- Functional Organization:
 - Engineers are organized into functional groups, e.g.
 - specification, design, coding, testing, maintenance, etc.
 - Engineers from functional groups get assigned to different projects

Advantages of Functional Organization

- Specialization
- Ease of staffing
- Good documentation is produced
 - different phases are carried out by different teams of engineers.
- Helps identify errors earlier.

Project Organization

- Engineers get assigned to a project for the entire duration of the project
 - Same set of engineers carry out all the phases
- Advantages:
 - Engineers save time on learning details of every project.
 - Leads to job rotation

Team Structure

- Problems of different complexities and sizes require different team structures:
 - Chief-programmer team
 - Democratic team
 - Mixed organization

Democratic Teams

- Suitable for:
 - small projects requiring less than five or six engineers
 - research-oriented projects
- A manager provides administrative leadership:
 - at different times different members of the group provide technical leadership.

Democratic Teams

- Democratic organization provides
 - higher morale and job satisfaction to the engineers
 - therefore leads to less employee turnover.
- Suitable for less understood problems,
 - a group of engineers can invent better solutions than a single individual.

Democratic Teams

- Disadvantage:
 - team members may waste a lot of time arguing about trivial points:
 - absence of any authority in the team.

Chief Programmer Team

- A senior engineer provides technical leadership:
 - partitions the task among the team members.
 - verifies and integrates the products developed by the members.

Chief Programmer Team

- Works well when
 - the task is well understood
 - also within the intellectual grasp of a single individual,
 - importance of early completion outweighs other factors
 - team morale, personal development, etc.

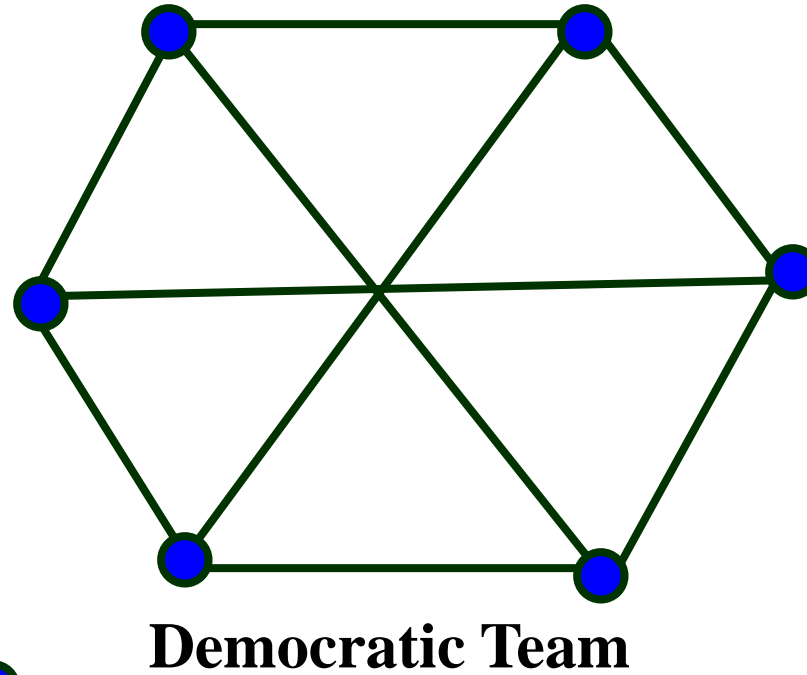
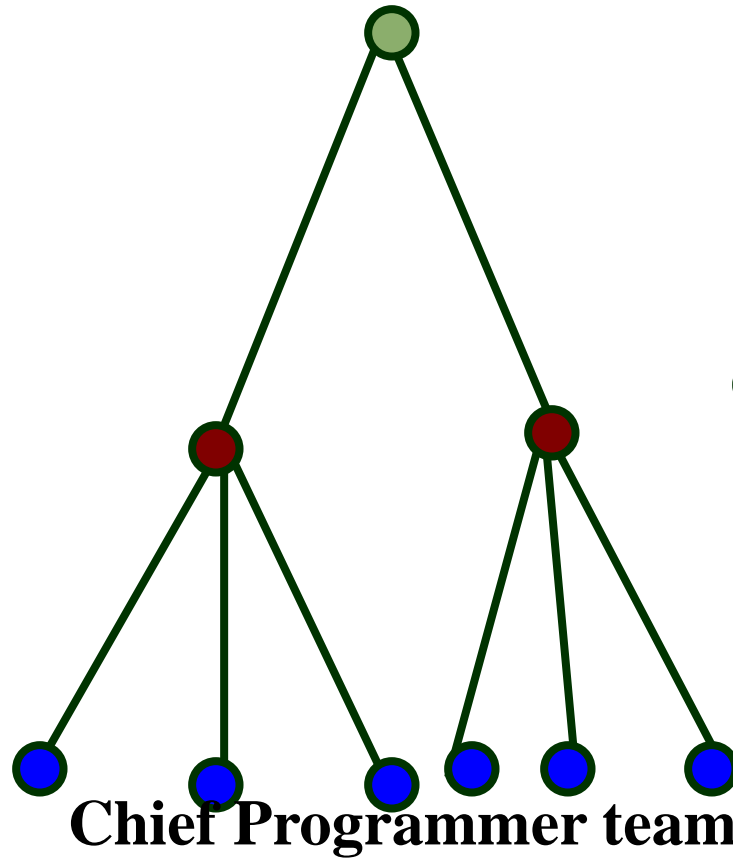
Chief Programmer Team

- Chief programmer team is subject to **single point failure**:
 - too much responsibility and authority is assigned to the chief programmer.

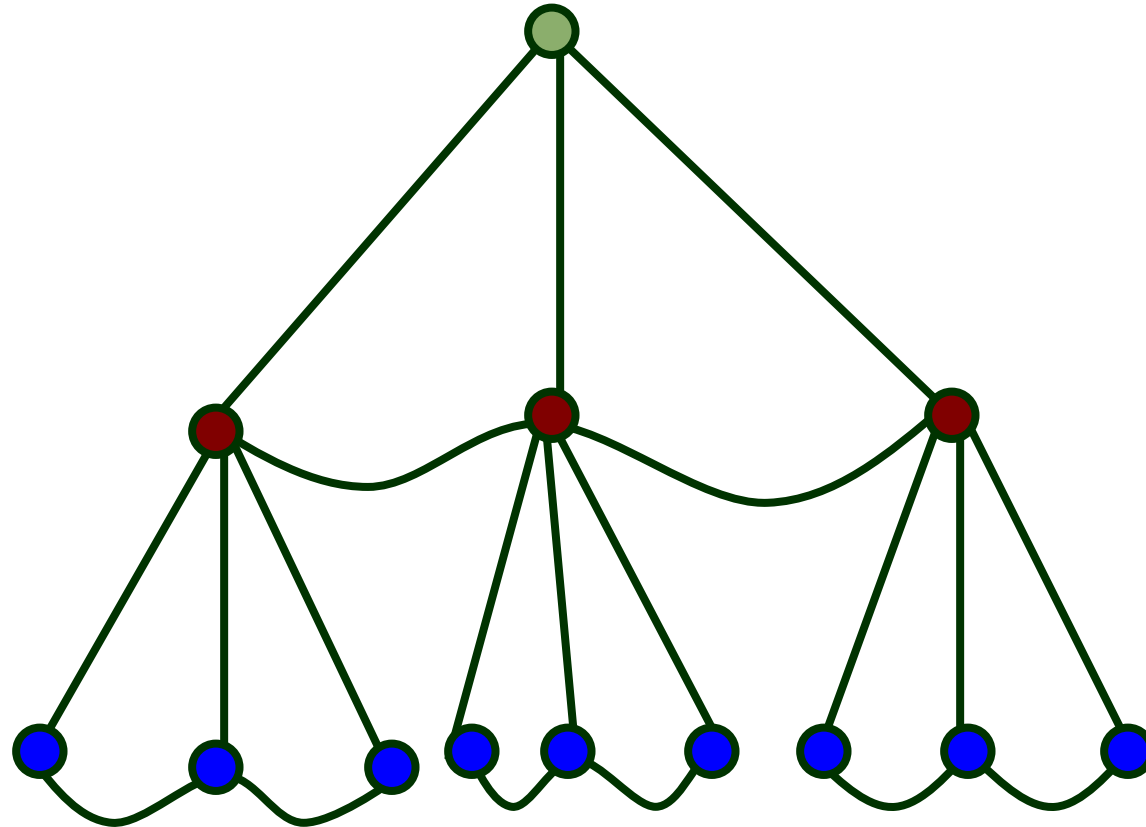
Mixed Control Team Organization

- Draws upon ideas from both:
 - democratic organization and
 - chief-programmer team organization.
- Communication is limited
 - to a small group that is most likely to benefit from it.
- Suitable for large organizations.

Team Organization



Mixed team organization



Summary

- We discussed the broad responsibilities of the project manager:
 - Project planning
 - Project Monitoring and Control

Summary

- To estimate software cost:
 - Determine size of the product.
 - Using size estimate,
 - determine effort needed.
 - From the effort estimate,
 - determine project duration, and cost.

Summary (CONT.)

- Cost estimation techniques:
 - Empirical Techniques
 - Heuristic Techniques
 - Analytical Techniques
- Empirical techniques:
 - based on systematic guesses by experts.
 - Expert Judgement
 - Delphi Estimation

Summary (CONT.)

- Heuristic techniques:
 - assume that characteristics of a software product can be modeled by a mathematical expression.
 - COCOMO
- Analytical techniques:
 - **derive** the estimates starting with some basic assumptions:
 - Halstead's Software Science

Summary (CONT.)

- The staffing level during the life cycle of a software product development:
 - follows Rayleigh curve
 - maximum number of engineers required during testing.

Summary (CONT.)

- Relationship between schedule change and effort:
 - highly nonlinear.
- Software organizations are usually organized in:
 - functional format
 - project format

Summary (CONT.)

- Project teams can be organized in following ways:
 - Chief programmer: suitable for routine work.
 - Democratic: Small teams doing R&D type work
 - Mixed: Large projects