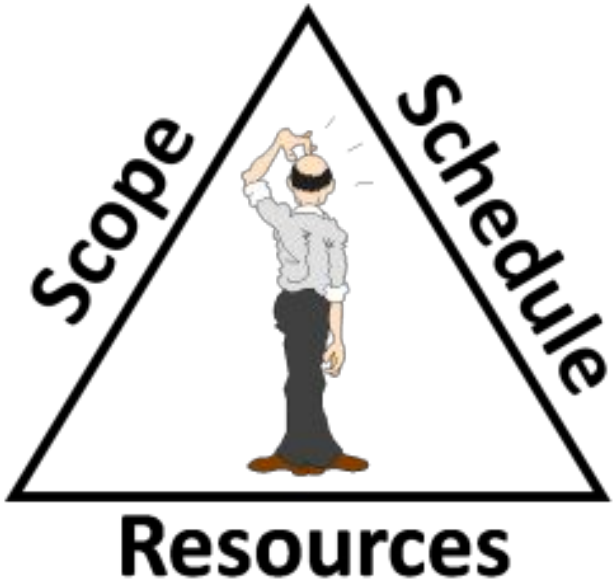
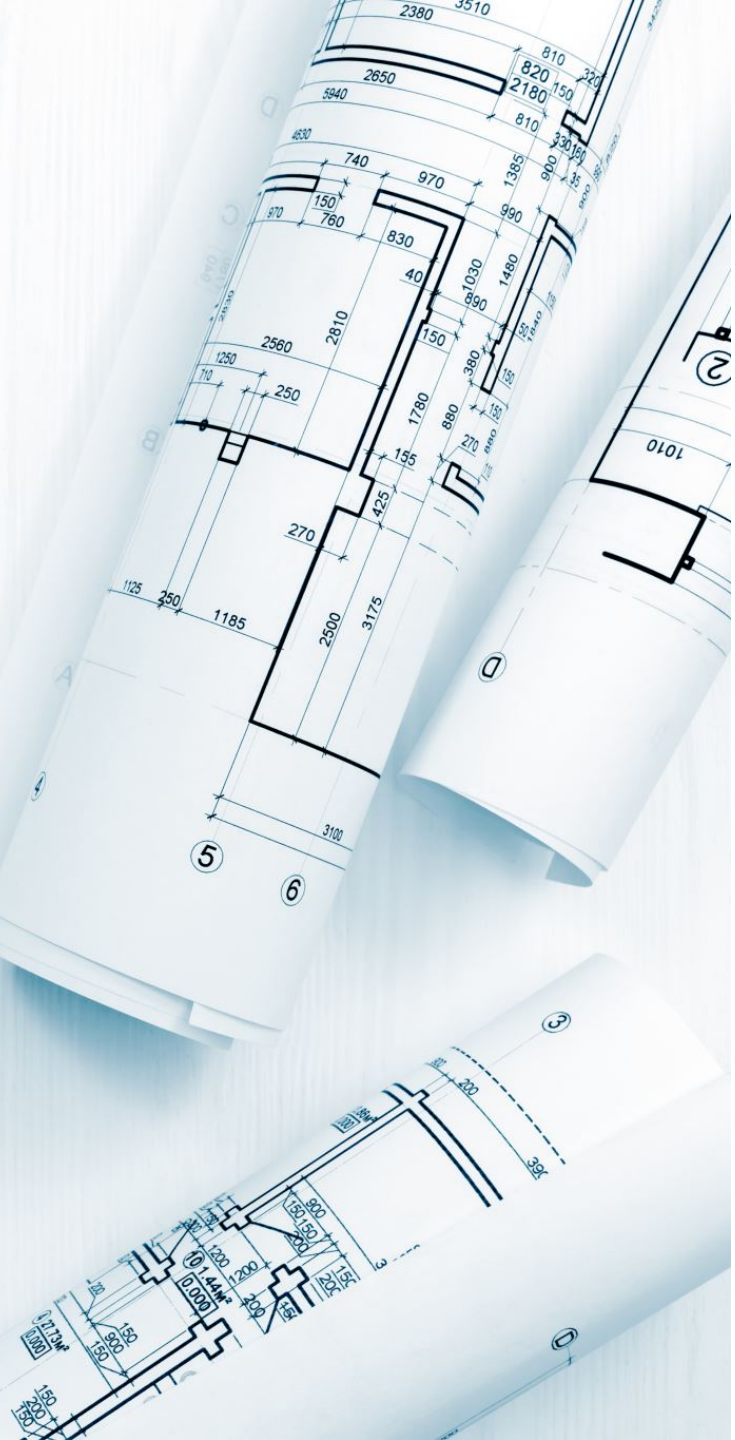
Session-4
Project size and Cost Estimation techniques



Dr. Bharati Wukkadada



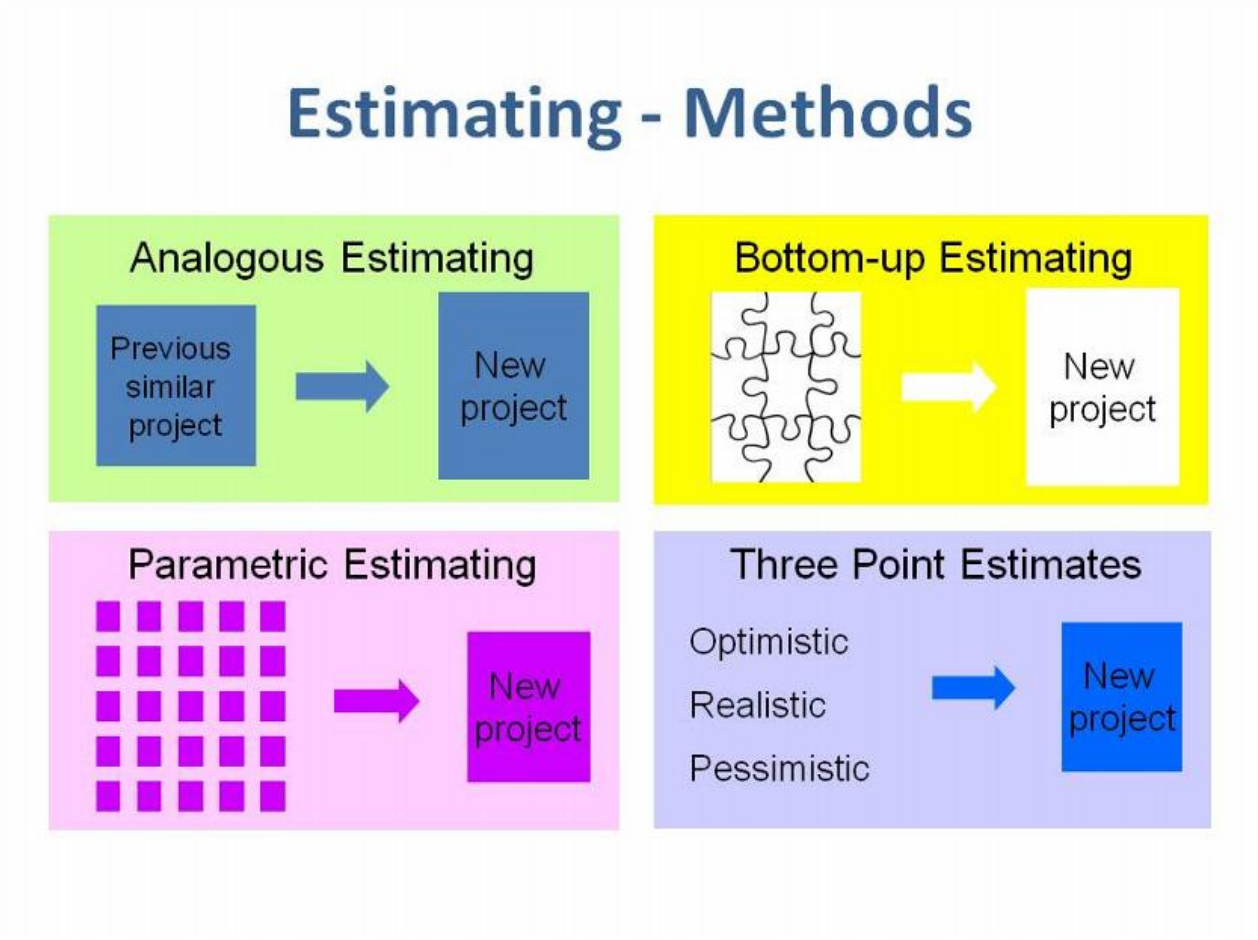
index

- ✔ Project size
- ✔ Project Estimating methods
- ✔ Cost estimation
- ✔ Cocomo model


Resources
Times
Human Skill
Cost
Estimate of what?

Software project estimation

Estimating methods



Optimistic means "taking a favorable view of events or conditions and expecting the most favorable outcome."

Pessimistic means "taking an unfavorable view of events or conditions and expecting an unfavorable outcome."

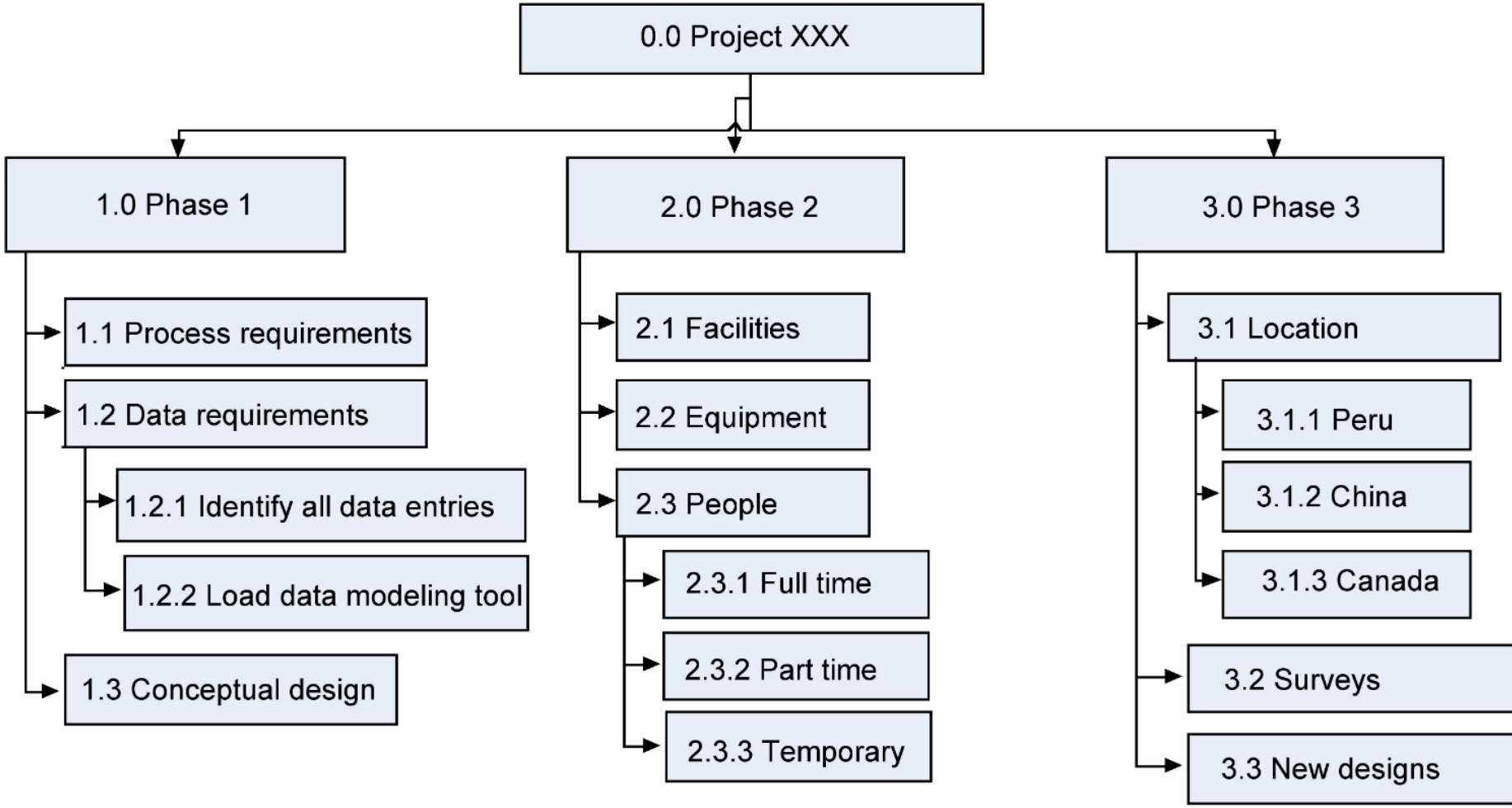
- **Analogous estimation** is a method of estimation that relies on the similarity between projects. The previous past project is known as the analog, and it serves as a basis for estimation on the new project. By contrast, parametric estimation calculates the expected cost based on known variables

Parametric estimating is a **project management technique that uses statistics to calculate the required cost and time to complete a project, a portion of a project, or a task**[12345]. It uses a formula to evaluate historical and statistical data to deliver precise results based on the relationship between variables, such as a parameter, cost, or time[1]. It is an established method in several project management frameworks such as the Project Management Institute's PMI Project Management Body of Knowledge (PMBOK)[3].

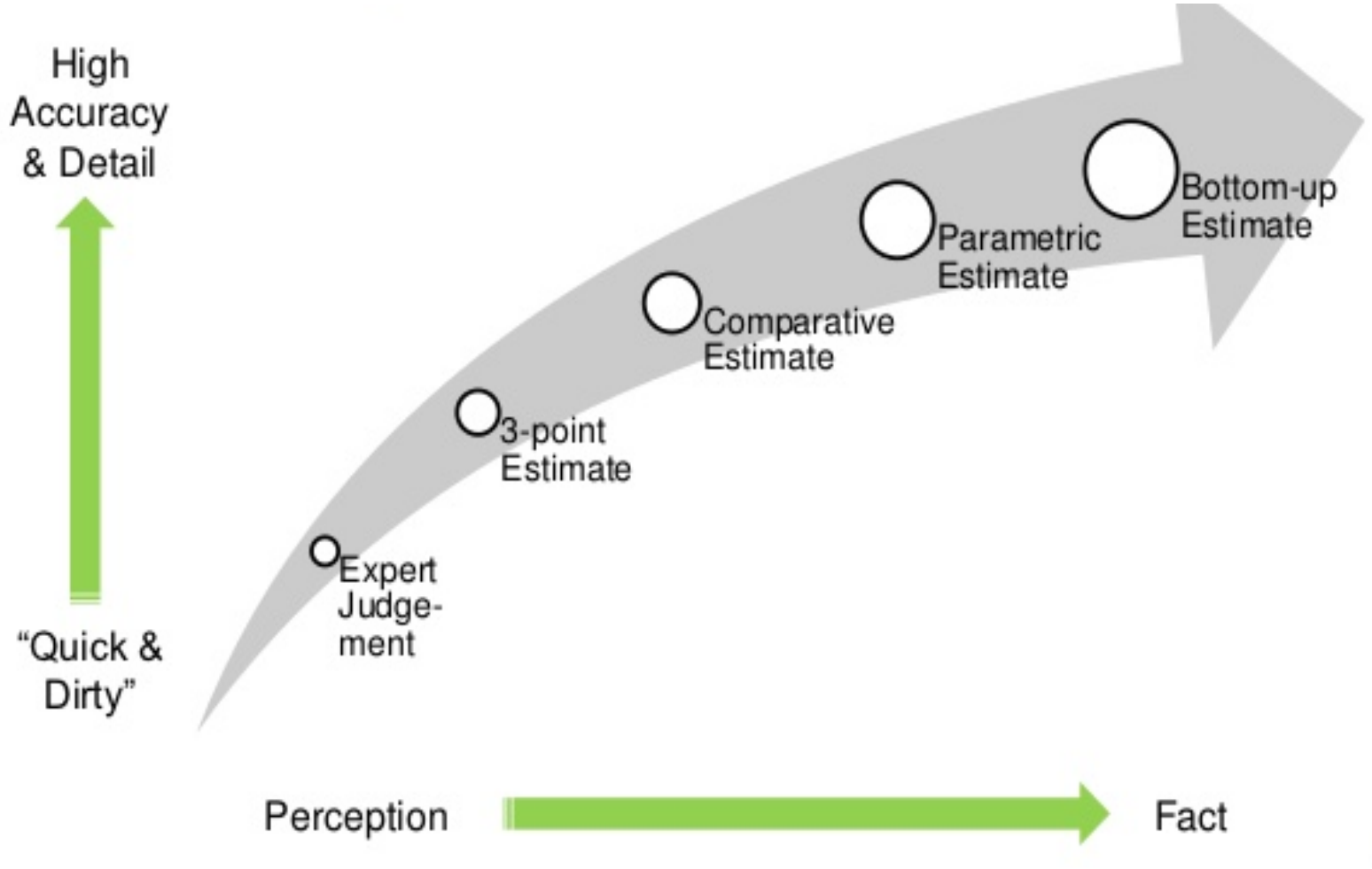
Bottom-up estimating is a **project management technique for estimating a project's overall cost, timeline, and resources by breaking down the project into smaller, detailed tasks**[12345]. Each task is then individually estimated, which allows for more accurate and detailed cost and time estimates[24]. The estimates from the lowest level of the Work Breakdown Structure (WBS) are then aggregated to arrive at an overall forecast[45]

Three-point estimating is a **technique project managers use to estimate the costs and schedules of their projects**[12]. It involves creating **three different estimates for each activity: the optimistic, the pessimistic, and the most likely**[12]. The three estimates are then averaged to reduce the risks, biases, and uncertainties that may affect the project outcomes[1]

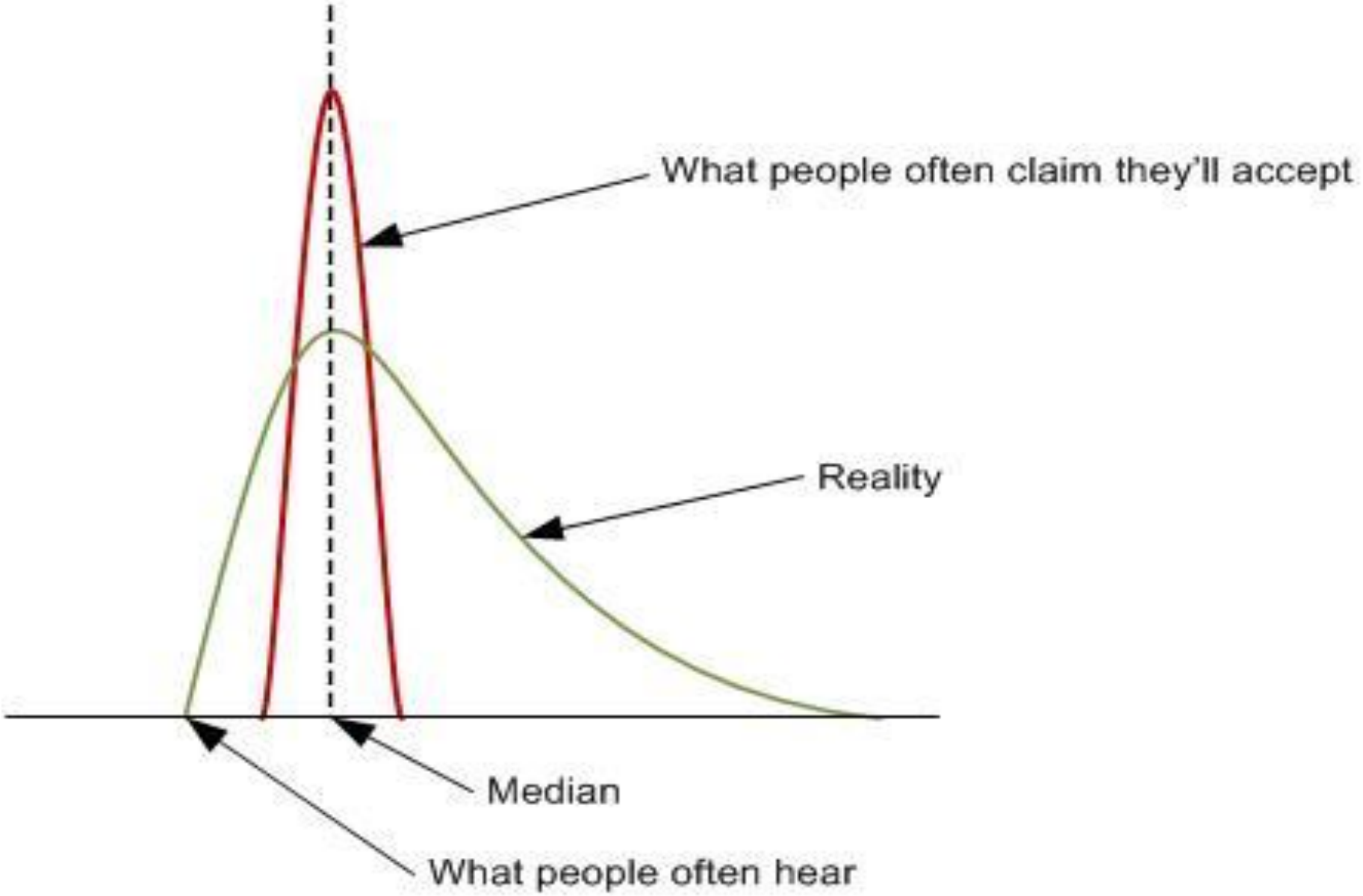
WBS EXAMPLE


0.0 Project XXX
1.0 Phase 1
2.0 Phase 2
3.0 Phase 3
1.1 Process requirements
1.2 Data requirements
1.2.1 Identify all data entries
1.2.2 Load data modeling tool
1.3 Conceptual design
2.1 Facilities
2.2 Equipment
2.3 People
2.3.1 Full time
2.3.2 Part time
2.3.3 Temporary
3.1 Location
3.1.1 Peru
3.1.2 China
3.1.3 Canada
3.2 Surveys
3.3 New designs

Estimation Methods



Practical Realities of Software Guesstimation



This diagram humorously represents the challenges of software project estimation, often referred to as "Software Guesstimation." It compares three different perspectives on project timelines:

1.What People Often Claim They'll Accept (Red Curve)

1. This curve is sharply peaked, indicating that stakeholders, managers, or clients claim they are willing to accept a highly optimistic and narrow time estimate.
2. They expect the project to be completed within a very short and precise timeframe.

2.Reality (Green Curve)

1. This curve is skewed to the right (right-tailed), showing that real-world software projects take much longer than the initial optimistic estimate.
2. Delays and unforeseen complexities extend the actual completion time beyond expectations.

3.What People Often Hear

1. This refers to the communication gap between developers and stakeholders.
2. Stakeholders might hear an overly optimistic estimate (closer to the median) rather than the realistic one, leading to misunderstandings and frustration when delays occur.

Key Takeaways

- There is a mismatch between expectations and reality in software project estimation.
- Clients/stakeholders want highly precise and optimistic estimates.
- Real project timelines are often longer due to unexpected complexities.
- Communication gaps contribute to unrealistic expectations.

This illustrates the importance of setting realistic expectations, using probabilistic estimation methods, and communicating uncertainty effectively in software development projects.

What is Project Estimating ?

- **Estimate** : A quantitative assessment of the likely amount or outcome.

- **Project Estimating** : The act of creating a quantitative assessment of the likely amount or outcome.

- **Estimation Process** :
  - Estimate Activity Resources
  - Estimate Activity Durations
  - Estimate Costs

- **Key Roles in Project Estimating**
  - Project Manager
  - Estimators
  - Program Management
  - Senior Management/Project Sponsor
  - Customer

Estimating Techniques

- **Pure Expert Judgment**
- **Historical Data**
- **Wideband Delphi**
- **Source line of code(SLOC)**
- **Function point(FP)**
- **Use case point**
- **Story point**
- **Monte Carlo**
- **Constructive Cost Model(COCOMO)**
- **Test case point(TCP), etc**

Pure Expert Judgement

- It is **mainly base on the expert knowledge**
- **One or more expert in both software development and the application domain** use their experience to predict software cost.
- **Process iterates until some consensus is reached.**

Historical data

- **Estimation by analogy**
  - **Applicable when other projects in the same application domain have been completed.**

- **PORBE**(Proxy Based Estimating) : PROBE is base on ***the idea that if an engineer is building a component similar to one he built previously, then it will take about the same effort as it did in the past.***

  - **A formula based on linear regression is used to calculate the estimate for each task** from this history.

DELPHI METHOD

What is the Delphi technique?

The Delphi technique is a method of forecasting that leverages the collective intelligence of an expert panel through a structured process of rounds of questionnaires and anonymous feedback. This iterative process helps in tackling complex issues by obtaining a reliable consensus from a group of experts. Originating during the Cold War era, the Delphi method has evolved to find applications across a wide range of fields, including public policy, social sciences, and health care.

> **"The Delphi technique is a method of group decision-making and iterative forecasting that involves consulting a panel of experts and implementing systematic feedback rounds."**

A brief history of the Delphi technique

Invented as a Cold War strategy, it's named after the Ancient Greek Oracle of Delphi. Olaf Helmer and Norman Dalkey of Rand Corporation developed it at the beginning of the Cold War after General Henry Arnold ordered a report on future technologies that could be useful in future military endeavors.

The project team first tried traditional forecasting methods and quantitative methods of data analysis, but it soon became obvious that these methods were redundant in areas that lacked established scientific theories. The Delphi protocol turned out to be effective for overcoming these obstacles. By consulting experts' views as a method of data collection, it was possible to arrive at accurate predictions about possible enemy attacks.

Important characteristics of the Delphi technique

There are four features that distinguish the Delphi technique from other group decision strategies: anonymity, iterative feedback, group response, and consulting experts' opinions. Let's walk through these characteristics in a little more detail.

- **Anonymity:** The benefit of anonymity is that it encourages group members to express their opinions freely. It prevents the collection of dishonest thoughts by removing the potential effects of peer pressure.
- **Iterative feedback:** This is achieved by implementing controlled feedback rounds so members can get a bird's-eye view of what the rest of the panel members are thinking. This gives them insight into how they might adapt their response.
- **Group response:** This gives participants the opportunity to adapt and build upon the information in the feedback round. It's done multiple times until the experts reach a consensus.
- **Using experts:** Rather than extracting participants from a random sample, inventors of the Delphi technique advocate consulting experts in the field in which the prediction is being made.

Steps in the Delphi process

To forecast the outcome of future trends and events, you can use the Delphi technique by following these steps:

1. **Choose a facilitator:** You can choose to take on this role yourself or select a neutral person who's knowledgeable about the Delphi protocol. It also helps if the facilitator has some experience in data collection.
2. **Identify experts:** First, you'll need to identify the specific field of interest. Once you've done this, find multiple experts in that field to ensure quality of results.
3. **Define the research question:** Try to narrow down the problem to exact terms. This will help you obtain more accurate results and make it easier for the experts to formulate their thoughts and answers.
4. **Carry out three response and feedback rounds:** Start by asking for anonymous answers to a specific question. Then, present the information to the experts and allow them to adapt the response in light of the new information. Do this three times.
5. **Act on your findings:** After the third round, take your results and act on them. What you essentially have in your hands is distilled expert knowledge, so it should be accurate.

The Delphi technique in project development and management

In project management, the technique is often used for scope and risk management. It's useful for scope management since it can aid stakeholders in settling on the project's scope. This can remove major causes of project failure, including unclear requirements and poor planning.
It can also help project teams with risk management by predicting potential hazards and better preparing for them. That's why the approach was developed in the first place: to aid in future forecasting and head off any problems.

Benefits and drawbacks of the Delphi technique

With the help of the Delphi approach, you can compile expert opinions and essentially distill expert knowledge. Without individual panelists worrying about consequences and peer pressure, they're free to express their opinions openly. The approach has proven to be relatively accurate in many cases, and it leverages the beneficial phenomenon of collective intelligence.

While the advantages of the Delphi technique seem enough to start using it immediately, there are also some drawbacks. One major disadvantage is that conducting Delphi studies can be extremely time-consuming. This makes it an inappropriate method for daily or weekly decisions, so the method is reserved only for the most consequential situations.

Another drawback is that while using experts might sound like a foolproof method, it's possible for experts to have utterly opposing views. In cases like this, the accuracy of results is compromised and a great deal of time wasted.

Practical examples of the Delphi technique

Here are examples of how the Delphi method works in real life:

Example 1

The U.S. government wanted to learn how many adolescent pregnancies there were in a certain town. The government hired a project facilitator to gather the data. It then chose a team of specialists who were familiar with the city. The professionals did a great job of interacting with the locals.

After conducting research, the facilitator created a poll that made it apparent to the experts what the issue was. The first round of the questionnaire was completed by the experts, and the facilitator then reviewed the responses to come up with a summary report, which he then presented to the experts.

Example 2

A pharmaceutical company wanted to figure out whether a particular drug it was developing would be effective and safe for use. Having no existing data on this, it consulted a team of medical experts who were familiar with the chemical compounds used in the drug. Each of the experts read research papers and systematic reviews on topics surrounding the potential drug.

A facilitator then created a poll asking the experts to indicate an estimate for the safety of the drug on a scale from 1 to 100. They repeated this for three rounds and then included the results in a summary report about the drug's potential development.

Source line of code(SLOC)

- Used to measure the size of a software program by counting the number of lines in the text of the program's source code.

- We need to divide the problem into modules, and each module into sub modules, and so on **until the sizes of the different leaf-level modules can be approximately predicted.**

- Two major types of SLOC measures :
  - **Physical SLOC** is a count of lines in the text of the **program's source code including comment lines**.

  - **Logical SLOC** attempts **to measure the number of executable "statements"(**e.g. in C-like programming languages is the number of statement-terminating semicolons).
    E.g. KLOC i.e. 1,000 lines of code.

Example

```
for (i = 0; i < 100; i++) printf("hello"); /* How many lines of code is this? */
```

```
/* Now how many lines of code is this? */
for (i = 0; i < 100; i++)
{
      printf("hello");
}
```

In this example we have:

- 1 | 5 Physical Lines of Code (LOC)
- 2 | 2 Logical Line of Code (LLOC) (*for* statement and *printf* statement)

6

Function point

A **Function Point** (FP) is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user. FPs measure software size. They are widely accepted as an industry standard for functional sizing.

For sizing software based on FP, several recognized standards and/or public specifications have come into existence. As of 2013, these are –

ISO Standards

- **COSMIC** – ISO/IEC 19761:2011 Software engineering. A functional size measurement method.
- **FiSMA** – ISO/IEC 29881:2008 Information technology - Software and systems engineering - FiSMA 1.1 functional size measurement method.
- **IFPUG** – ISO/IEC 20926:2009 Software and systems engineering - Software measurement - IFPUG functional size measurement method.
- **Mark-II** – ISO/IEC 20968:2002 Software engineering - Ml II Function Point Analysis - Counting Practices Manual.
- **NESMA** – ISO/IEC 24570:2005 Software engineering - NESMA function size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis.

- Function count measure functionality from user point of view.

- The base of the function count is
  what the user requests and
  what he/she receives in return from the system.

- Quantitative(Objective) measure and industry data is available as basis for comparison.

Functions

There are two types of functions –

- Data Functions
- Transaction Functions

Data Functions

There are two types of data functions –

- Internal Logical Files
- External Interface Files

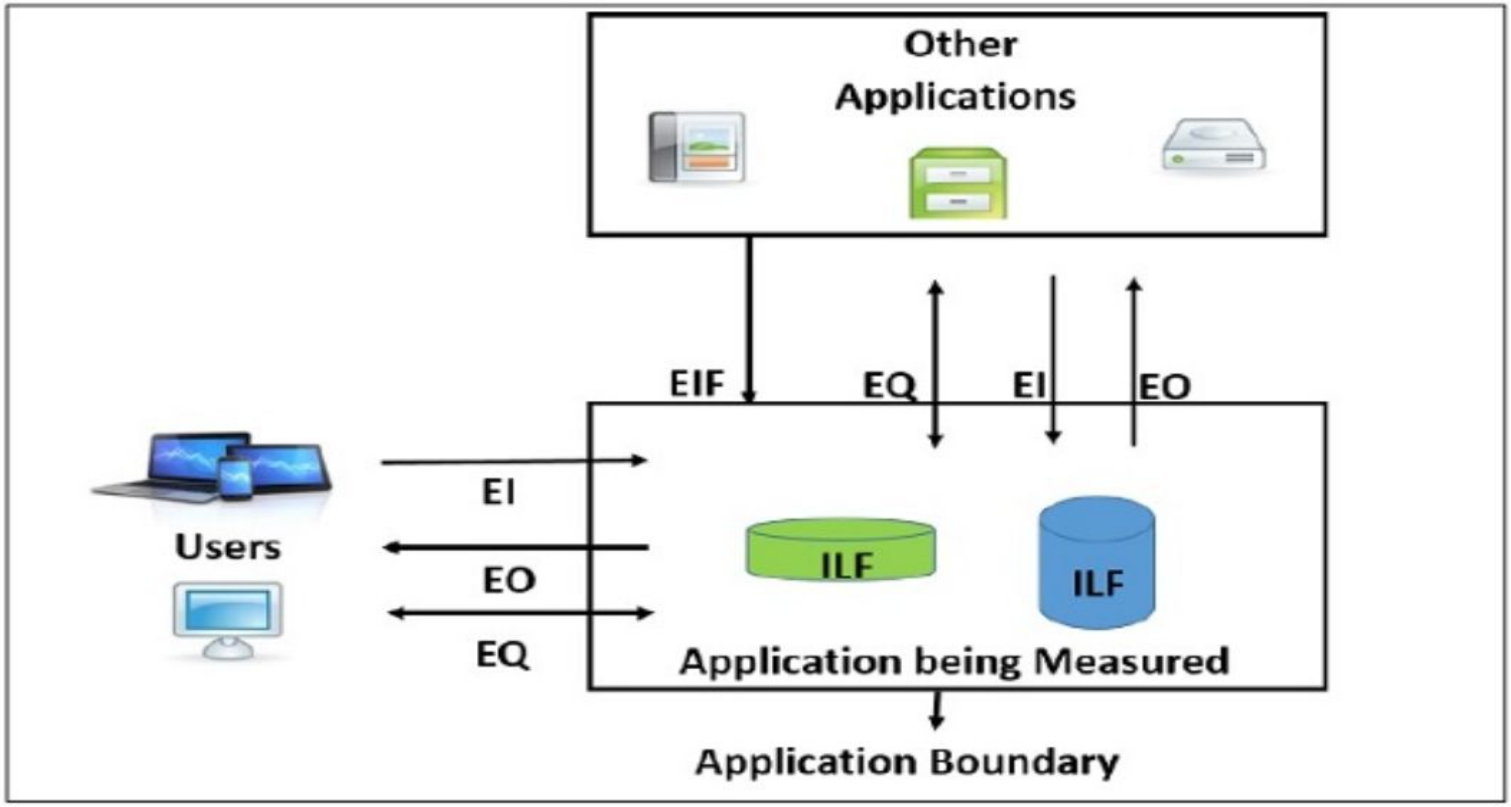


Figure 1: Application Boundary, Data Functions, Transaction Functions

EIF External Interface Files
EI External Input
EO External Output
EQ External Inquiry

External Interface Files

External Interface File (EIF) is a user identifiable group of logically related data or control information that is used by the application for reference purposes only. The data resides entirely outside the application boundary and is maintained in an ILF by another application. An EIF has the inherent meaning that it is externally maintained, an interface has to be developed to get the data from the file. (Refer Figure 1)

Transaction Functions

There are three types of transaction functions.

- External Inputs
- External Outputs
- External Inquiries

Transaction functions are made up of the processes that are exchanged between the user, the external applications and the application being measured.

External Inputs

External Input (EI) is a transaction function in which Data goes "into" the application from outside the boundary to inside. This data is coming external to the application.

- Data may come from a data input screen or another application.
- An EI is how an application gets information.
- Data can be either control information or business information.
- Data may be used to maintain one or more Internal Logical Files.
- If the data is control information, it does not have to update an Internal Logical File. (Refer Figure 1)

External Outputs

External Output (EO) is a transaction function in which data comes "out" of the system. Additionally, an EO may update an ILF. The data creates reports or output files sent to other applications. (Refer Figure 1)

External Inquiries

External Inquiry (EQ) is a transaction function with both input and output components that result in data retrieval. (Refer Figure 1)

Three-Point Estimating/PERT

- More sophisticated form of the range estimation technique.
- 3 separate values of the estimation are derived: optimistic, pessimistic and most likely.
- Helps to normalize the subjective data.

- Program Evaluation and Review Technique (PERT):
- Uses statistical probability outcome, based on weighted average
- E = (O + 4M + P)/6
  - E= Effort, O=Optimistic, ML=Most Likely, P=Pessimistic

Other techniques

- **Monte Carlo**: It is based on the generation of multiple trials to determine the expected value of a random variable.
- **Parkinson Law:** Work expands to fill the time available. Cost determined by available resources.
  - E.g., if the software has to be delivered in 12 months and 5 people are available, the effort required is estimated to be 60 person-months.
- **Test case point**: Estimates the size of testing projects using test cases as input.
- **Pricing to win:** The price is fixed according to what the client is prepared to pay.
- **COCOMO**
  - Hierarchy of software cost estimation models, which include Basic, Intermediate and Detailed sub models,
  - $E = a(KLOC)b * EAF$
  - $D = c(E)d$
  - E is Effort, D is Development time, a, b, c, d are coefficients

Cont…

- **COCOMO**(Constructive Cost Model) :
  - The Constructive Cost **Model** (**COCOMO**) is an procedural software cost estimation **model** developed by Barry W. Boehm.
  - COCOMO applies to:
    - Organic mode(2-50 KLOC)
    - Semi-detached mode(50-300 KLOC)
    - Embedded mode(over 300 KLOC)

Project Size Estimation Metric

- Measures
- Metrics
- Indicators
- Line of Code (LOC)
- Function Point Metric
- Features Point Metric

1. ***Measures***

- A Measure provides a
- quantitative indication of the
  - extent,
  - amount,
  - dimension,
  - capacity
  - or a size of some attributes of a product or process."

- "Measurement" is the act or a process of determining a measure

2.Metric

- "A quantitative measure of the degree to which
  - the system,
  - component
  - or a process possesses a given attribute."

- Software engineer collects measures and develop metrics so that indicators will be obtained
- Eg: there were only two user-discovered errors in the first 18 months of operation of software, then it will be a quality software

3. Indicator

- "An Indicator *is a metric or combination of metrics that provides insight into the software process, a software project or the product itself."*

- It enables the project manager or software engineers to adjust the process, the project or the product to make things better.

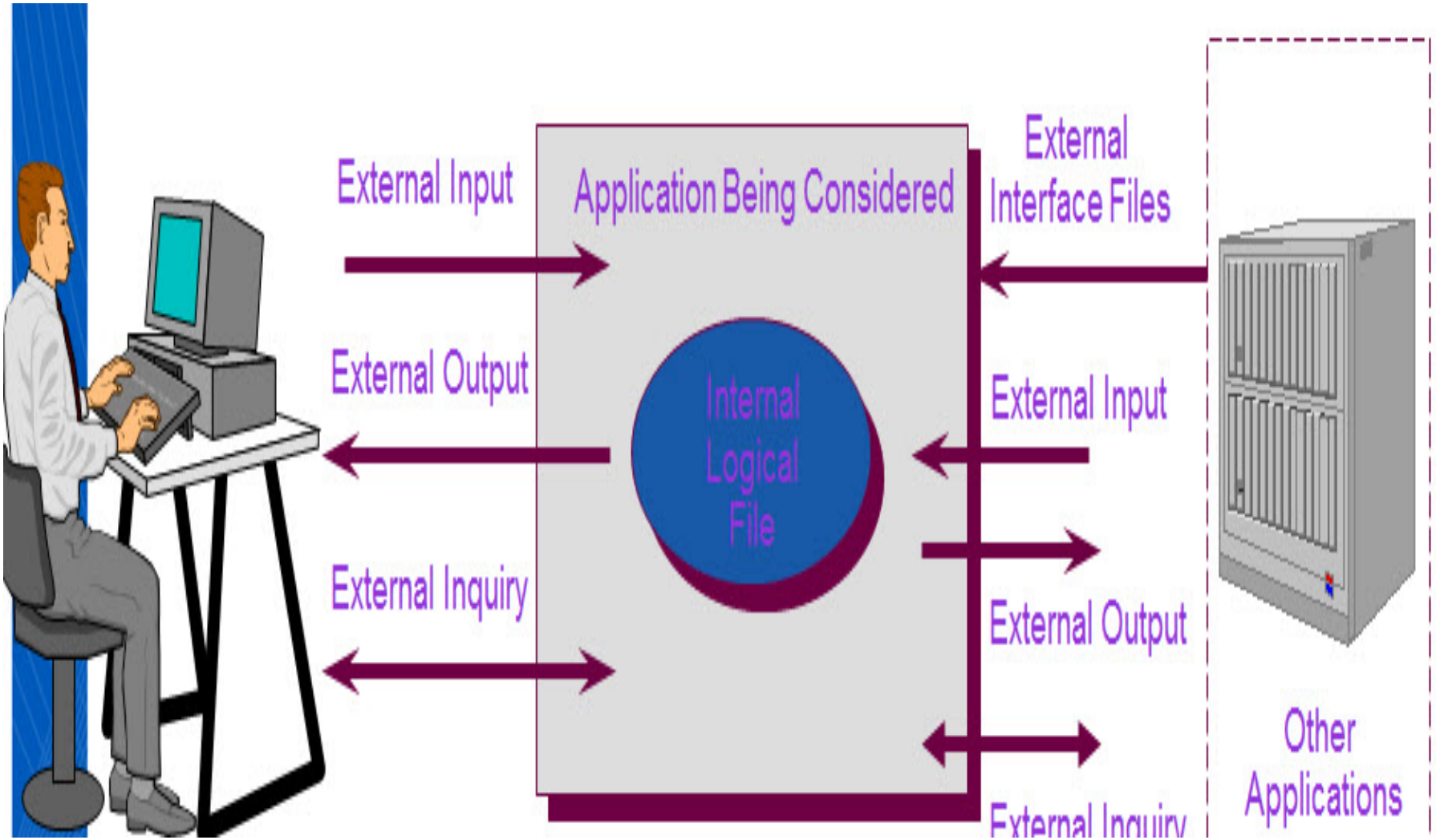
This **allows the decision makers to make a quick comparison** that can provide a perspective as to the "health" of a particular aspect of the project.

5, Function Point Metric

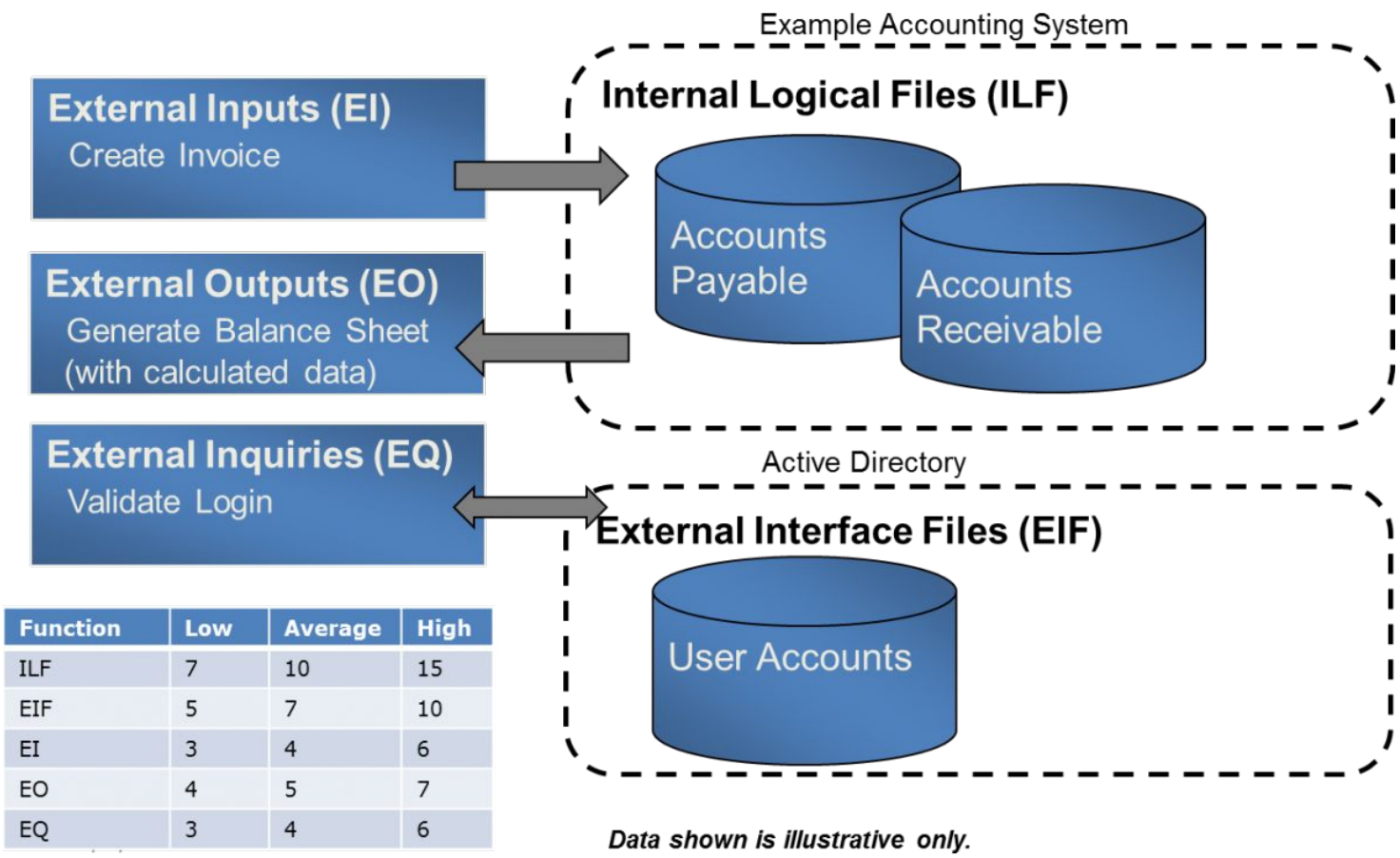
- It estimate the size of the software product directly from the problem specification
- It focuses on program “Functionality”
- the size of the software product is directly dependent on the number of different functions it support
- A product with many functions will be larger than a product with less number of functions
- In addition size also depends on the number of files and number of interfaces
- Size of a product in function point (FP)can be expressed as the weighted sum of five important characteristics

Parameters

1. Number of inputs
2. Number of outputs
3. Number of User Inquiries
4. Number of Files
5. Number of External Interfaces


External Input
Application Being Considered
External Interface Files
External Output
Internal Logical File
External Input
External Inquiry
External Output
Other Applications
External Inquiry

Parameters



Function	Low	Average	High
ILF	7	10	15
EIF	5	7	10
EI	3	4	6
EO	4	5	7
EQ	3	4	6

Data shown is illustrative only.

Function Point Metric

The information is entered in the following table to get the total count.

Measurement parameter	Count	Weighting factor				
		Simple	Average	Complex		
Number of user inputs		x 3	4	6	=	
Number of user outputs		x 4	5	7	=	
Number of user inquiries		x 3	4	6	=	
Number of files		x 7	10	15	=	
Number of external interfaces		x 5	7	10	=	
Count-total	→					

Fig. 3.2
Computing function-point metrics

1. **Number of Inputs :**
   - Each user input that provides distinct application oriented data to the software is counted.
   - Eg: In employee payroll system, data items such as name, age, sex. Address , phone number of an employee is considered as a single input

2. **Number of user outputs :**
   - Each user output that provides application oriented information to the user is counted.
   - Eg. – Reports, Screens, Error Messages etc.

3. Number of User Inquiries :

- It is an on-line input that results in the generation of some immediate s/w response in the form of on-line output.
- Eg**:** interactive queries made by users, user commands

5. Number of Files :

- Each Logical Master File (i.e. a logical grouping of data that may be one part of a large database or a separate file) is counted.
- Eg: data structures or physical files

6. Number of External Interfaces :

- All machine readable interfaces that are used to transmit information to another system are counted.
- Eg: data files on tapes, disks etc

- After collecting data, a complexity value associated with each count is determined.
- Organizations should develop methods for determining whether a particular entry is simple, average or complex.

- After calculating the total count,

  FP = count total * [0.65 +0.01 * Sum(F_i)],

  where

  count total = sum of all FP entries obtained from above table

  F_i = " Complexity Adjustment Values" based on responses to the following questions

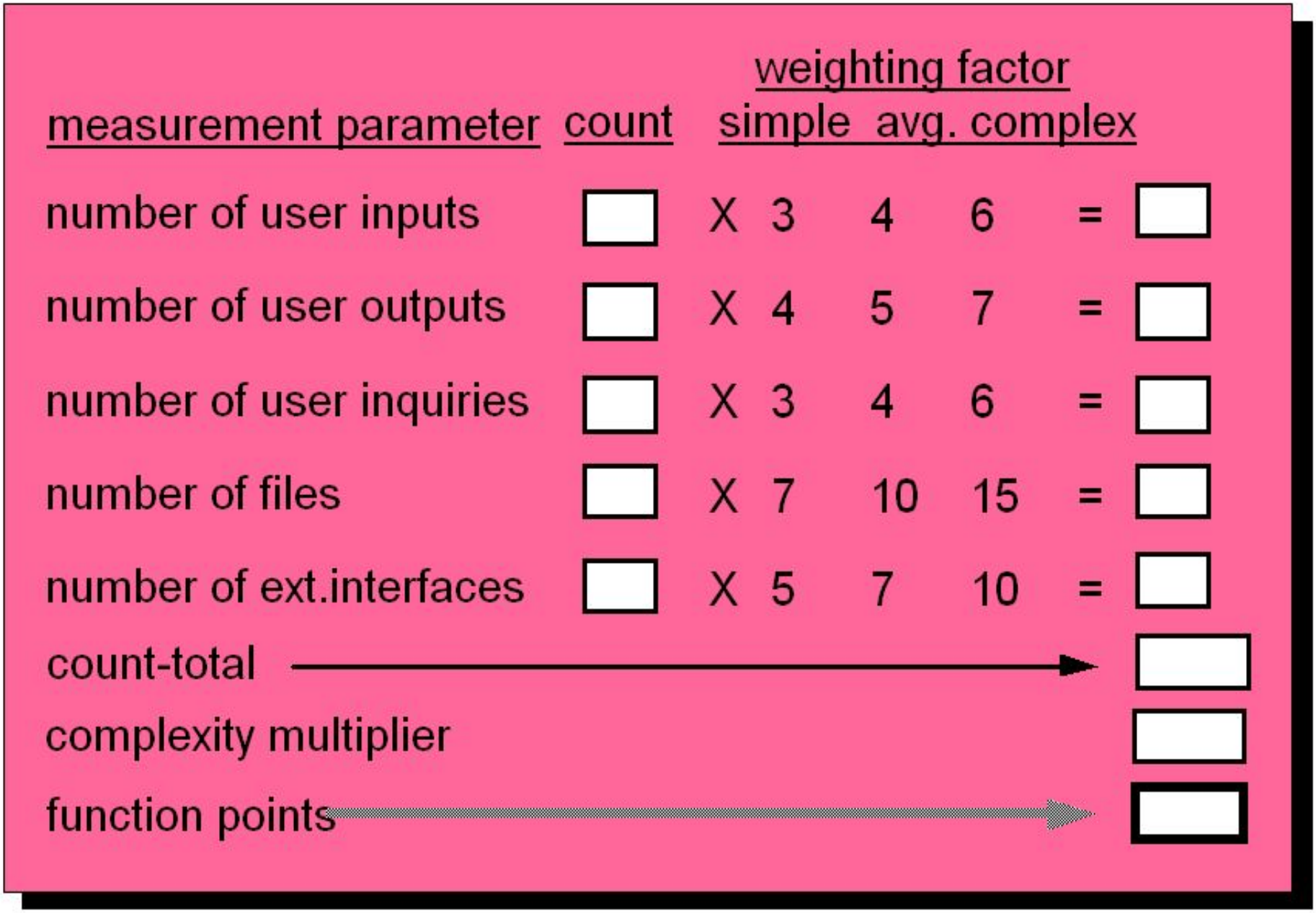
Complexity Adjustment Factor drivers

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require on-line entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the inputs, outputs, files, or inquiries complex
9. Is the internal processing complex?
10. Is the code designed to be reusable?
11. Are master files updated on-line?
12. Are conversion and installations included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

- Each of these questions is answered using a scale that ranges from 0 to 5

0 - No influence

1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential


weighting factor
measurement parameter
count
simple
avg.
complex
number of user inputs
X 3 4 6 =
number of user outputs
X 4 5 7 =
number of user inquiries
X 3 4 6 =
number of files
X 7 10 15 =
number of ext.interfaces
X 5 7 10 =
count-total
complexity multiplier
function points

Example

Consider a project with following functional units:

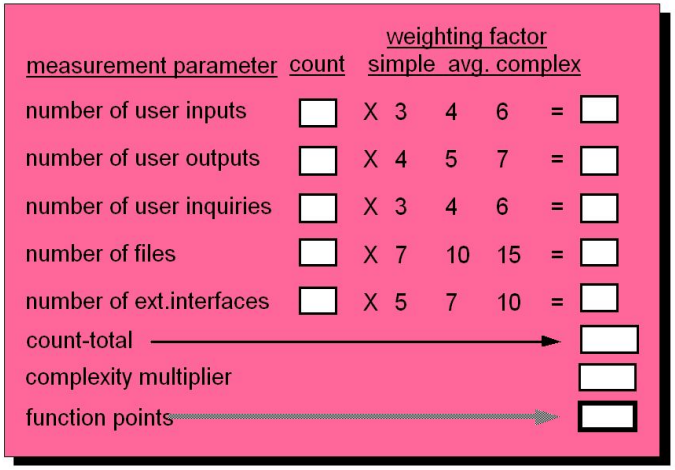
- Number of User Inputs = 50
- Number of User Outputs = 40
- Number of user inquiries = 35
- Number of user files = 6
- Number of external interfaces = 4

Assume all complexity adjustment factors & weighting factors are average.

Compute the function point for the project.

The key missing detail is where **Sum(Fi) =**.
It appears that **Fi values (factors) are summed up and multiplied by a weighting factor (3)**. If these values represent different influencing factors, so14 will be their sum. **14 × 3** comes from[assuming and using the sum]

Functional Point(FP) Complexity Adjustment Factor (CAF)



- Formula : using average

  FP = count total * [0.65 +0.01 * Sum(Fi)]

count total = 50*4 + 40*5+ 35 *4 + 6*10 + 4*7

= 200 + 200 + 140 + 60 + 28

= 628

CAF = 0.65 + 0.01* (14*3)

= 0.65+0.42 = 1.07

FP = 628 * 1.07

= 671.96 = 672

Drawback of Function Point Metric

- It does not take into account the algorithmic complexity of a software
- It assumes that **the effort required to design & develop any two functionalities of the system is the same.**
- It does not distinguish between the difficulty levels of developing the various functionalities.

Example

- Consider a project with the following parameters

1. External Inputs
   a) 10 with low complexity
   b) 15 with average complexity
   c) 17 with high complexity
2. External outputs
   a) 6 with low complexity
   b) 13 with high complexity

3. External inquiries
   a) 3 with low complexity
   b) 4 with average complexity
   c) 2 with high complexity

4. Internal logical files
   a) 2 with average complexity
   b) 1 with high complexity

5. External interface files
   a) 9 with low complexity

In addition, system requires

1. Significant data communication
2. Performance is very critical
3. Designed code may be moderately reusable
4. System is not designed for multiple installations in different organizations

- Other complexity adjustment factors are treated as average
- Compute the function points for the project

6. Features Point Metric

- algorithm complexity as an extra parameter
  - eg: inverting a matrix, decoding a bit string, handling an interrupt
- It overcomes the drawbacks of Function Point Metric
- reflects the fact that the more the complexity of the function, the greater the effort required to develop it
- therefore it's size should be larger compared to simpler functions
- Both the Function Point & Feature Point Metric are Language independent & can be easily computed from SRS Document.

Decomposition Techniques

- Form of problem solving

- decompose it, as a set of smaller problems

- Decomposition consists of decomposition
  - **problem**
  - **process**

- Planner should also consider the scope and size of the software

A. Software Sizing

- Accuracy of s/w project estimate is predicted based on a number of things

1. The degree to which the planner has properly estimated the size of the product to be built
2. The ability to translate the size estimate into human effort, calendar time & dollars
3. The degree to which the project plan reflects the abilities of the software team
4. The stability of Product Requirements & Environment that supports the s/w engineering efforts.

- Size can be measured either in LOC or as FP
- four different approaches are there to the sizing problem
- They are

1. “Fuzzy Logic” Sizing
2. Function Point Sizing
3. Standard Component Sizing
4. Change Sizing

“Fuzzy Logic” Sizing

- uses the approximate reasoning techniques

- In this the planner must
  - identify the type of application
  - establish it’s magnitude on a qualitative scale &
  - then refine the magnitude within the original range

- He should have access to historical database of projects to compare estimate to actual experience

- He can use personal experience also

Function Point Sizing

- The planner develops estimates of the information domain characteristics
- It includes inputs, outputs, external interfaces, files and inquiries.

Standard Component Sizing

- s/w is composed of a number of different "Standard Components"

- Eg- the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files & LOC

- estimates the number of occurrences of each standard component

- then uses historical project data to determine the delivered size per standard component.

- For eg – if planner estimate that 18 reports will be generated. Historical data indicates that 967 lines of COBOL are required per report. This enables the planner to estimate that 17, 406 LOC will be required for the reports components.

Change Sizing

- This approach is used when a project encompasses the use of existing software that must be modified in some way as a part of a project.
- The planner estimates
  - the number & type of modifications that must be accomplished. (e.g. reuse, adding code, changing code, deleting code)
- Then the size of change is estimated

B. Problem based estimation

- prepare a statement of software scope
- decompose the software in to problem functions
- the problem functions are estimated individually
- LOC or FP is estimated is estimated for each problem function
- He may also choose another components for sizing such as classes or changes

C. Process based estimation

- Process is decomposed into small set of tasks
- Then the effort required to accomplish each task is estimated
- This method will first define the software scope
- From the scope software functions are obtained
- Each function perform a series of software process activities
- From this planner will estimate the effort required to accomplish each process activity

Software Estimation

- Basic activity in planning is the estimation of various parameters of the software

- Three categories of estimation techniques

    1. Empirical estimation techniques
    2. Heuristic estimation techniques
    3. Analytical estimation techniques

Estimation of various projects parameters is an important project planning activity. The different parameters of a project that need to be estimated include–

- Project Size,
- Effort required to complete the project,
- Project Duration and
- Cost

Accurate estimation of these parameters is important for resource planning and scheduling. Estimation Techniques can be classified as :

- **Empirical Estimation Techniques**
- **Heuristic Estimation Techniques**
- **Analytical Estimation Techniques**

Empirical Estimation Techniques

- Empirical: derived from experiment, experience and observation rather than theory

- based on making an educated guess of the project parameters.

- Using prior experience you make an *educated* guess

- Two Empirical estimation techniques are:

1. Expert Judgment Technique
2. Delphi Cost Estimation

Heuristic Estimation Technique

- This technique assumes that the characteristics to be estimated can be expressed in terms of some mathematical expression

- If the basic independent parameters are known then the other dependent parameters can be determined by substituting the values in mathematical expression

- Different estimation models are available
- These models are divided in to two classes

1. single variable model
2. multivariable model

Single variable model

- It provides a means to estimate the desired characteristics of a problem using some previously estimated basic characteristics of the software product
- It takes the form:

$$\text{Estimated Parameter} = c_1 * e^{d1}$$

where

e is a characteristics of the software that is already been estimated (independent variable)

Estimated parameter is the dependent parameter to be estimated (effort, project duration, staff size etc)

c_1 and d_1 are constants (data collected from past projects)

E.g.: basic COCOMO model

Multivariable model

- Assumes that parameter to be estimated depends on more than one characteristics
- It takes the form:

  Estimated Parameter = $c_1 * e^{d1} + c_2 * e^{d2} + ...$

where

  e_1, e_2 … are the basic characteristics of software

  c_1, c_2, d_1, d_2 ,….. are constants (historical data)

- This model provides more accurate estimation

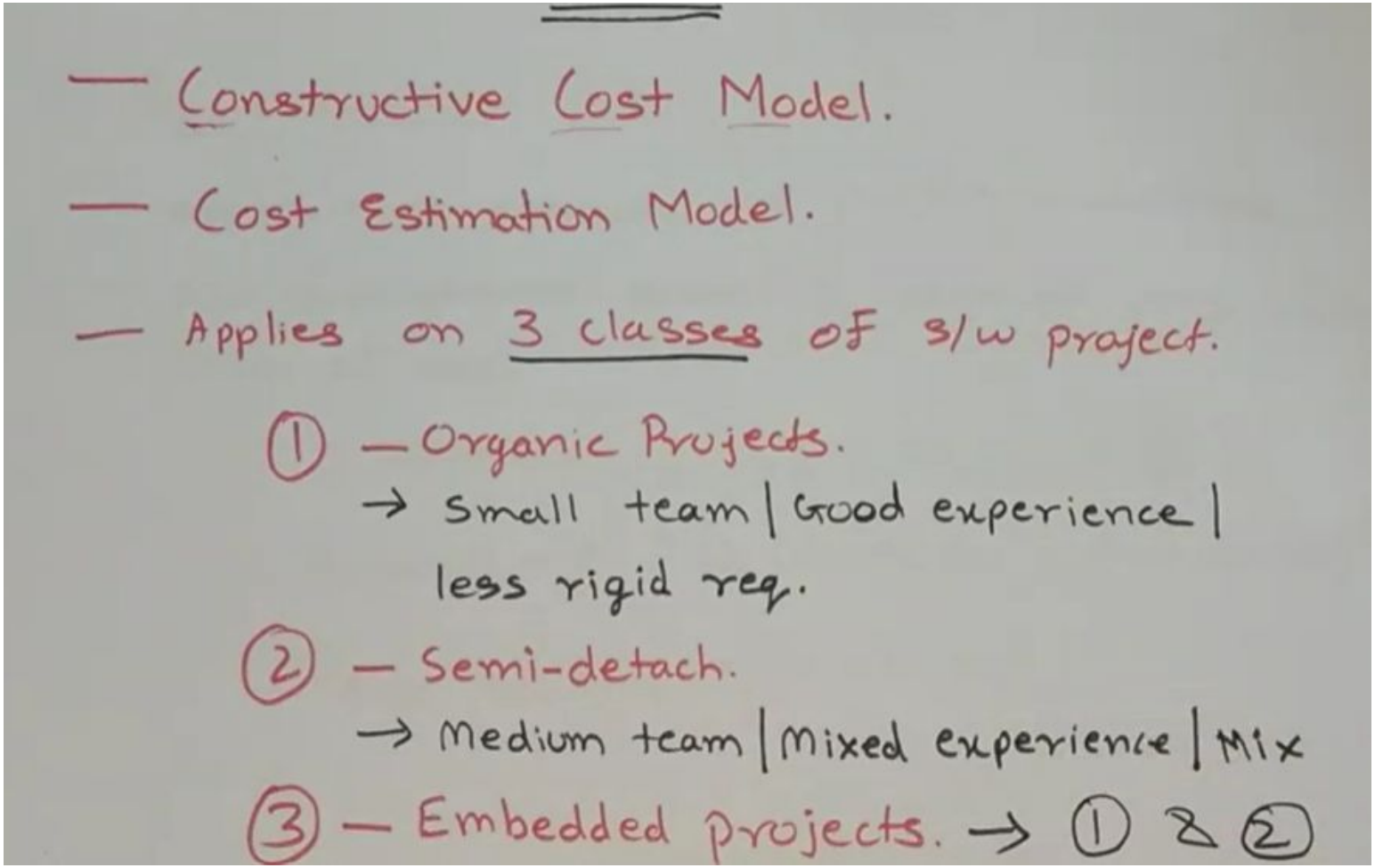
- Eg: intermediate COCOMO model

4. COCOMO – (Constructive Cost Model)

COCOMO – (Constructive Cost Model)

- Proposed by Boehm
- He postulated that any s/w development project can be classified into one of the following 3 categories based on the development complexity (divides s/w product development in to 3 categories)

1. Organic
2. Semidetached
3. Embedded



[Basic COCOMO & Intermediate COCOMO with Numerical]

COCOMO – (Constructive Cost Model)

- Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**. It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models. The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

- In this we have to consider the characteristics of the product, development team and development environment

- Three product classes correspond to

  application (data processing programs)

  utility (compilers and linkers)

  system program (operating systems ,real time systems)

The necessary steps in this model are:

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$E_i = a*(KDLOC)b$$

- The value of the constant a and b are depends on the project type.

3 TYPES / categories

- **Organic –** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
- **Semi-detached –** A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environment lie in between that of organic and Embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience and better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered of Semi-Detached types.
- **Embedded –** A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

3 TYPES / categories

- **1.Organic:** A development project can be treated of the 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 methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**
- **2. Semidetached:** A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**
- **3. Embedded:** A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

Organic

If the project deals with

- developing a well understood application program
- the size of the team is small and
- team members are experienced in developing similar types of project

Semidetached

- If the team
  - consists of both experienced and inexperienced staff
  - members have limited experience on related systems but may be unfamiliar with some aspects of the system

Embedded

- If the software being developed is strongly coupled to complex hardware, or if severe regulations on the operational procedures exist

Mode	Project Size	Nature Of Project	Deadline of the proj.	Development Envir.
Organic	2-50 KLOC	Small size project, small size team, experienced developers in the familiar environment. Eg – Application Programs such as Payroll, Inventory projects.	Not tight	Familiar & In House
Semi-detached	50-300 KLOC	Medium size comments, medium size team, Average previous experience on similar projects. Eg – Utility Programs : Compilers, linkers	Medium	Medium
Embedded	Over 300 KLOC	Large Projects, Real time systems, complex interfaces, very little previous experience. Eg – ATM, Air traffic control, O.S etc.	Tight	Complex h/w, customer interfaces required.

Software cost estimation is done through three stages

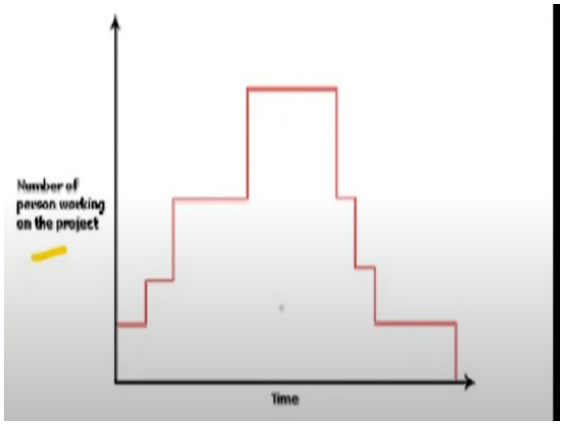
For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

- According to Boehm, software cost estimation should be done through three stages:

- Basic Model
- Intermediate Model
- Detailed Model

PERSON MONTH(PM)

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



Person month is a measurement unit for effort in software engineering. 1 person month means effort put by a person in one month. But 100 person does not mean, work effort put by 100 person in one month or 1 person in 100 months. As requirement of staff varies time to time in the development so there is not constant no of people is there to work. It is calculated from the graph(calculate area under time axis) between no of people working and time(in month).

PM. →

3D-4M
12 PM

4D-3M
12PM

Basic COCOMO Model:

The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

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

- The values for a_1, a_2, b_1, b_2 are :

Software Project	a_1 KLOC	A_2 PM	B_1 EFFORTS	B_2 MONTHS
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Estimation of development effort

- For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:
- **Organic:** Effort = 2.4(KLOC) 1.05 PM
- **Semi-detached:** Effort = 3.0(KLOC) 1.12 PM
- **Embedded:** Effort = 3.6(KLOC) 1.20 PM

Estimation of development time

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

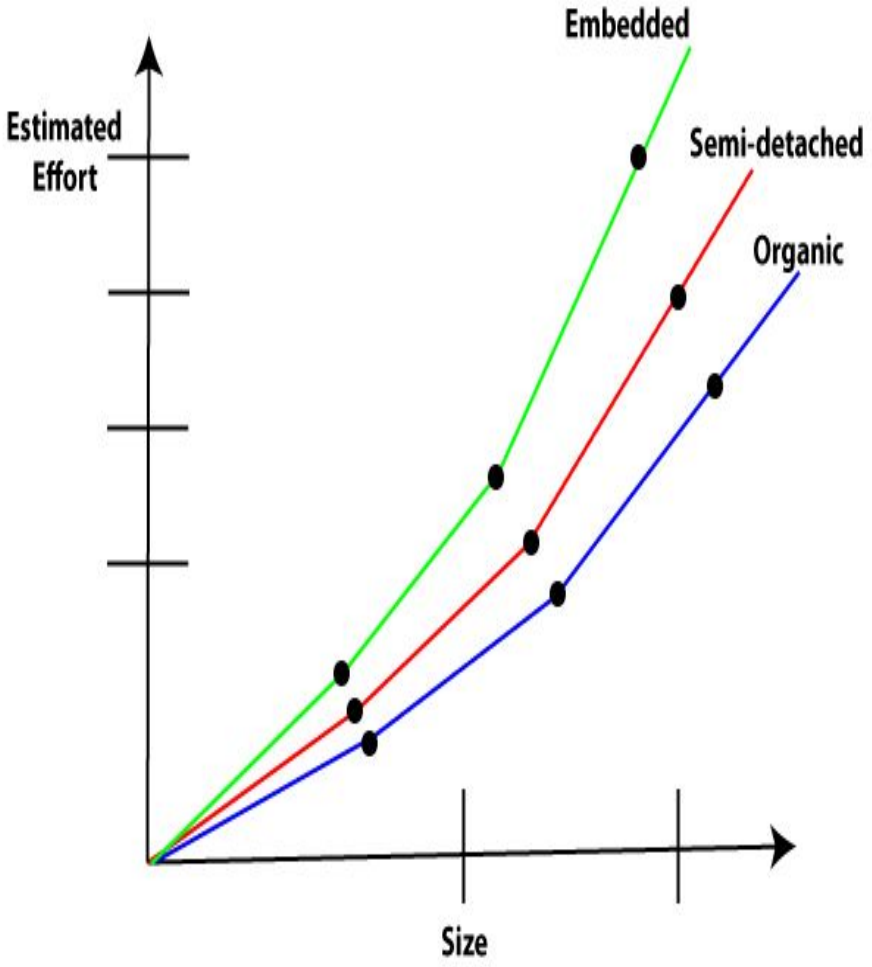
- **Organic:** Tdev = 2.5(Effort) 0.38 Months
- **Semi-detached:** Tdev = 2.5(Effort) 0.35 Months
- **Embedded:** Tdev = 2.5(Effort) 0.32 Months

- When Effort & Development Time are known, the Average Staff Size to complete the project may be calculated as :

Average Staff Size (SS)= Effort/ T_{dev} persons

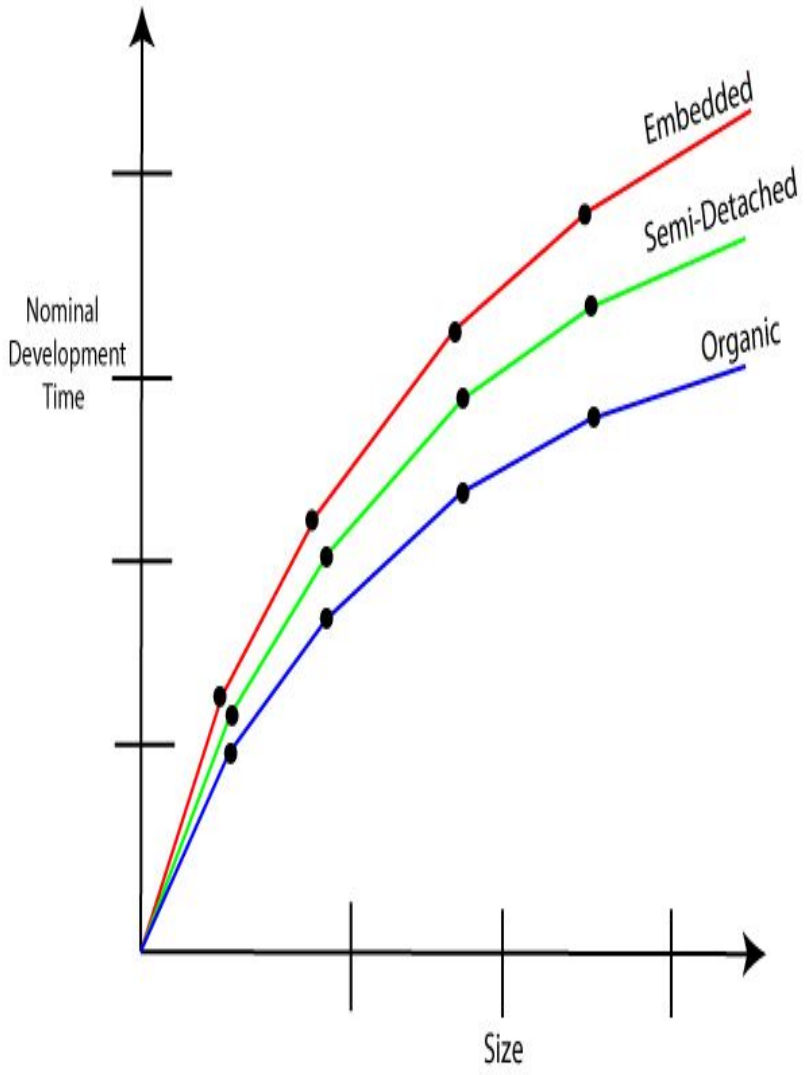
Productivity (P) = KLOC/ Effort KLOC/PM

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



Effort versus product size

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



Development time versus size

- From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.
- It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

ORGANIC MODEL

Example1

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.

EXAMPLE-1
Solution:

Software Projects	a_1	a_2	b_1	b_2
Organic	2.4	1.05	2.5	0.38
Semi - Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

- The basic COCOMO equation takes the form:
- Effort=a_1*(KLOC)a_2 PM
- Tdev=b_1*(efforts)b_2 Months

Estimated Size of project= 400 KLOC

- **(i)Organic Mode**
- E = 2.4 * (400)^1.05 = 1295.31 PM

D = 2.5 * (1295.31)^0.38=38.07 M

- **SS 34.01**
- **P 0.3088**

- **(ii)Semidetached Mode**
- E = 3.0 * (400)1.12=2462.79 PM
  D = 2.5 * (2462.79)0.35=38.45 M
  - SS 64.05 p
  - P 0.16 kloc/pm

- **(iii) Embedded Mode**
- E = 3.6 * (400)1.20 = 4772.81 PM
  D = 2.5 * (4772.8)0.32 = 38 M
- SS 98.6
- P 0.083

Example2:

- 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.

EXAMPLE-2 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=1133.12PM

D=2.5(1133.12)0.35=29.3PM

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

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

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

P = 176 LC/PM

Example-3

:

Assume that the size of organic type s/w project has been estimated to be 32,000 LOC. The Avg. Salary of S/w engineers is Rs. 15,000 per month. Determine the effort required to develop the s/w product & development time.

SOLUTION

- From the basic COCOMO estimation formula for organic s/w :

$$\text{Effort} = a_1 * (\text{KLOC})^{a_2}$$
$$= 2.4\ (32)^{1.05}$$
$$= 91\ \text{PM}$$

$$T_{dev} = b_1 * (\text{Effort})^{b_2}$$
$$= 2.5 * (91)^{0.38}$$
$$= 14\ \text{months}$$

HOME WORK

- Assume that the size of organic type s/w project has been estimated to be 45 KLOC. The Avg. Determine the effort required to develop the s/w product & development time, average staff size and productivity for all three modes.

Intermediate COCOMO

- Basic COCOMO model considers only the Effort & time to determine the product size.
- To obtain an accurate estimation of effort & project duration, the effect of all relevant parameters must be taken into account.
- Intermediate COCOMO model refines the initial estimate obtained through basic COCOMO expression by using a set of 15 cost drivers based on various attributes of s/w development.

- Cost drivers are used to adjust the nominal cost of a project to the actual environment

- It will increase the accuracy of the estimate

- Cost drivers are grouped into four categories
    1. Product attributes
    2. Computer attributes
    3. Personnel attributes
    4. Project attributes

- **Product attributes**
  a) Required software reliability (RELY)
  b) Data base size(DATA)
  c) Product complexity(CPLX)

- **Hardware attributes**
  a) Execution time constraints(TIME)
  b) Main storage constraint(STOR)
  c) Computer turnaround time(TURN)
  d) Virtual machine volatility(VIRT)

- **Personnel attributes**
  a) Analyst capability(ACAP)
  b) Programmer capability(PCAP)
  c) Virtual machine experience(VEXP)
  d) Application experience(AEXP)
  e) Programming language experience(LEXP)

- **Project attributes**
  a) Modern programming practices(MODP)
  b) Use of software tools(TOOL)
  c) Required development schedule(SCED)

- Each cost driver is rated for a given project environment
- For rating a scale is used
- It describes to which extent cost drivers applies to the project being estimated
- Table shows the multiplier values for the 15 cost drivers and their rating

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

- Multiplying factors for cost drivers are multiplied to get effort adjustment factor(EAF)
- EAF = range from 0.9 to 1.4
- Intermediate COCOMO is given by following expressions

$$E_i = a_i * (KLOC)^{b_i} \quad PM$$

$$Effort = EAF * E_i$$

$$T_{dev} = c_i(Effort)^{d_i}$$

Software Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Example

- A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers:

1. Very high application experience with very little experience in programming language.
2. Very low application experience but lot of experience with the programming language.

What is the impact of hiring all developers from one or the other pool?

$$E_i = a_i * (KLOC)^{b_i}$$

$$= 2.8 (400)^{1.20}$$

$$= 3712 \text{ PM}$$

CASE I

$$EAF = .82 * 1.14$$

$$= 0.9348$$

$$\text{Effort} = E_i * EAF = 3712 * 0.9348$$

$$= 3470 \text{ PM}$$

$$T_{dev} = c_i(\text{Effort})^{d_i}$$

$$= 2.5(3470)^{0.32}$$

$$= 33.9 \text{ M}$$

CASE II

EAF = 1.29 * 0.95

= 1.22

$$\text{Effort} = \text{EAF} * E_i$$

= 1.22 * 3712

= 4528 PM

$$T_{dev} = c_i(\text{Effort})^{d_i}$$

= 2.5* (4528)0.32

Complete COCOMO

- Most large systems are made up of several smaller subsystems. These subsystems may have widely different characteristics.
- For E.g. – some subsystems may be considered as organic type, some semi detached & some embedded.
- Even the reliability and team size of these subsystems may be different.
- The COCOMO model considers these differences & estimate the Effort & development time as the sum of the estimates for individual subsystems
- The cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.

Detailed COCOMO

- Offers a means for processing all characteristics of a project

- Introduce two more capabilities

1. Phase-sensitive effort multipliers
2. Three- level product hierarchy

- A set of phase sensitive effort multipliers are there for each cost driver
- Software development is carried out in four phases

1. Plan/requirement
2. Product design
3. Programming
4. Integration/test

- Three product levels are defined

  System

  subsystem

  Module

- Most large systems are made up of several smaller subsystems and modules
- They may have different characteristics also.
- Software development is carried out in different phases
- Effort and development time will be different for different phases
- All these factors are considered in detailed COCOMO

Lifecycle Phase Values of μ_p

Mode & Code Size	Plan & Requirements	System Design	Detailed Design	Module Code & Test	Integration & Test
Organic Small S≈2	0.06	0.16	0.26	0.42	0.16
Organic medium S≈32	0.06	0.16	0.24	0.38	0.22
Semidetached medium S≈32	0.07	0.17	0.25	0.33	0.25
Semidetached large S≈128	0.07	0.17	0.24	0.31	0.28
Embedded large S≈128	0.08	0.18	0.25	0.26	0.31
Embedded extra large S≈320	0.08	0.18	0.24	0.24	0.34

Table 7 : Effort and schedule fractions occurring in each phase of the lifecycle

Lifecycle Phase Values of τ_p

Mode & Code Size	Plan & Requirements	System Design	Detailed Design	Module Code & Test	Integration & Test
Organic Small S≈2	0.10	0.19	0.24	0.39	0.18
Organic medium S≈32	0.12	0.19	0.21	0.34	0.26
Semidetached medium S≈32	0.20	0.26	0.21	0.27	0.26
Semidetached large S≈128	0.22	0.27	0.19	0.25	0.29
Embedded large S≈128	0.36	0.36	0.18	0.18	0.28
Embedded extra large S≈320	0.40	0.38	0.16	0.16	0.30

Table 7 : Effort and schedule fractions occurring in each phase of the lifecycle

Software Engineering (3rd ed.), By K.K Aggarwal & Yogesh Singh, Copyright © New Age International Publishers, 2007

- Phase wise cost and schedule estimate are calculated using the equations

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

Example

- Consider a project to develop a full screen editor. The major components identified are
    1. screen edit
    2. command language interpreter
    3. file input and output
    4. cursor movement
    5. screen movement

  The sizes for these are estimated to be 4k, 2k,1k,2k and 3k delivered source code lines.

- Use COCOMO to determine:
  1. overall cost and schedule estimates

     Assume cost drivers as

     - required s/w reliability is high
     - product complexity is high
     - analyst capability is high
     - programming language experience is low
     - all other drivers are nominal

2. cost and schedule estimate for

- system design phase
- detailed design phase
- module code and test phase
- integration and test phase

solution

- Size of five modules are:

  screen edit = 4 KLOC

  command language interpreter = 2 KLOC

  file input and output = 1 KLOC

  cursor movement = 2 KLOC

  screen movement = 3 KLOC

  total =12 KLOC

- Phase wise effort

system design = 0.16 * 52.91 = 8.465 PM

detailed design = 0.26 * 52.91 = 13.756 PM

module code and test = 0.42 * 52.91= 22.222PM

integration and test = 0.16 * 52.91 = 8.465 PM

- Phase wise development time

system design = 0.19 * 11.29 =2.145 M

detailed design = 0.24 * 11.29 = 2.709 M

module code and test = 0.39 * 11.29 =4.403 M

integration and test = 0.18 * 11.29 = 2.032 M

COCOMO II

- Latest version of COCOMO model
- **Developed by university of southern California under leadership of Dr. Barry Boehm**
- **Provides quantitative analytical framework**
- It has 3 estimation models to estimate effort and cost
- It is actually a hierarchy of Estimation models that address the following areas:

Software Project Planning

COCOMO-II

The following categories of applications / projects are identified by COCOMO-II and are shown in fig. 4 shown below:

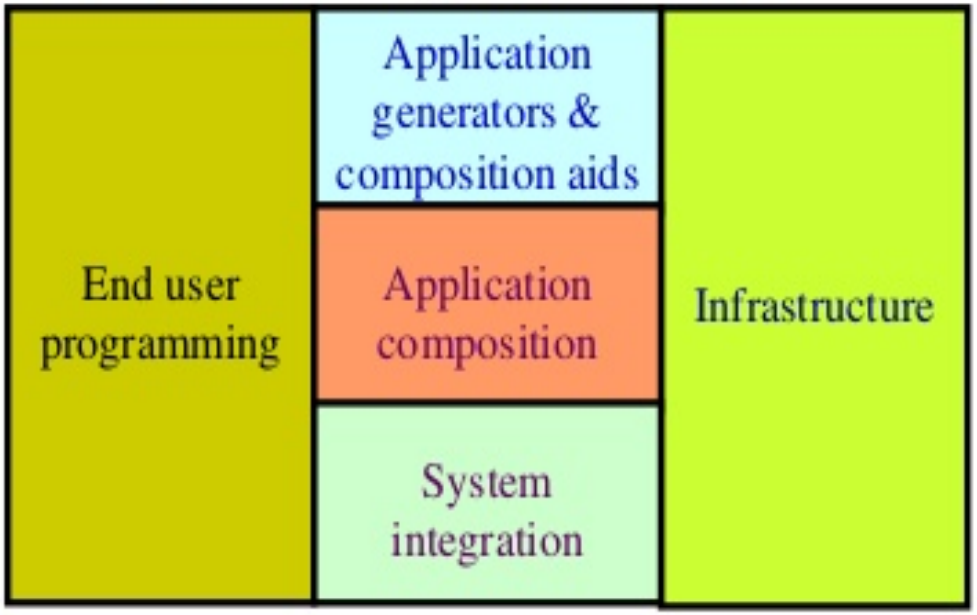


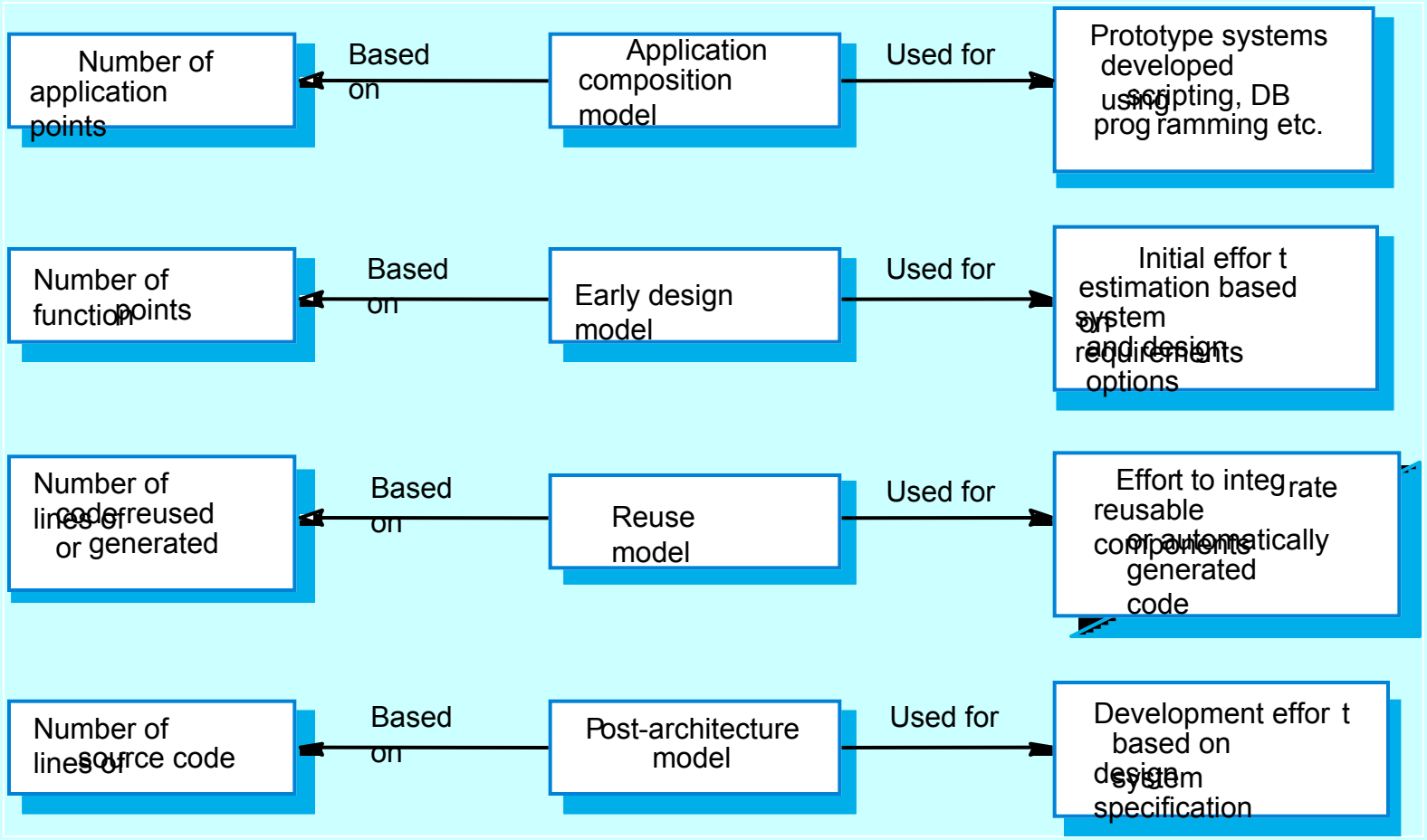
Fig. 4 : Categories of applications / projects

1. Application Composition Model :
   - Used during the early stages of S.E., when prototyping of user interfaces, consideration of s/w & system Interaction, assessment of performance etc

2. Early design Stage Model :
   - Used once requirements have been stabilized & basic s/w architecture has been established.

3. Post-Architecture stage Model :
   - Used during the construction of the s/w.

Use of COCOMO 2 models



Application Composition Model

- This is used when software can be decomposed into several components
- Each component can be described in object points
- Object points are easy to identify and count when system is divided into different sub system components

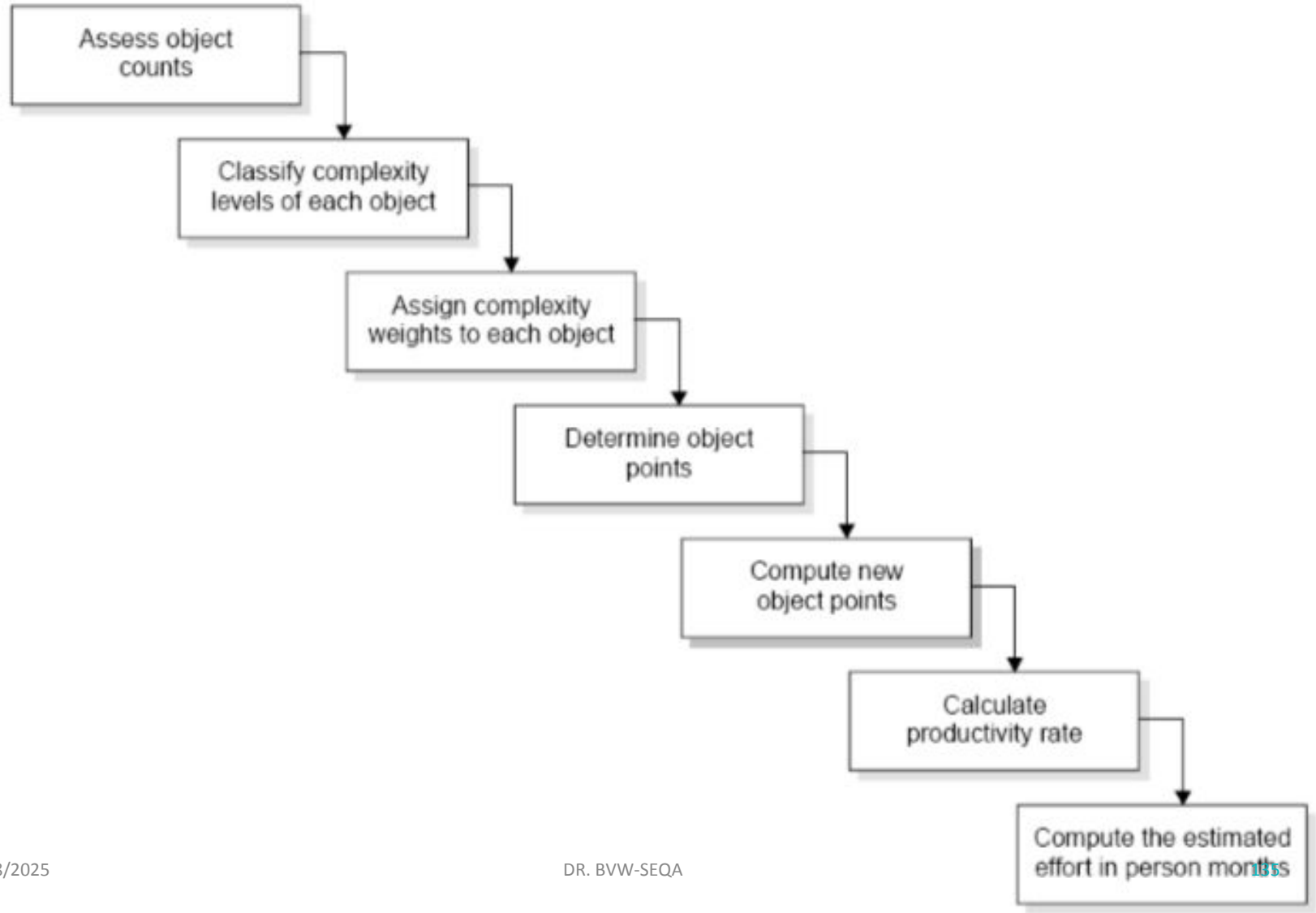
- This model requires sizing information.
- 3 different sizing options are available as part of the model hierarchy:
  - Object Point
  - Function Point
  - LOC

Object point

- It is an indirect s/w measure that is computed using counts of the number of :
  - Screens (user Interfaces)
  - Reports
  - Components

- Each object instance is classified into one of 3 complexity levels (simple, medium, difficult)

Application Composition Estimation Model



For screens

Number of views contained	*# and sources of data tables*		
	Total < 4 (< 2 server < 3 client)	*Total < 8 (2 – 3 server 3 – 5 client)*	*Total 8 + (> 3 server, > 5 client)*
< 3	Simple	Simple	Medium
3 – 7	Simple	Medium	Difficult
> 8	Medium	Difficult	Difficult

Table 9 (a): For screens

Number of sections contained	*# and sources of data tables*		
	Total < 4 (< 2 server < 3 client)	*Total < 8 (2 – 3 server 3 – 5 client)*	*Total 8 + (> 3 server, > 5 client)*
0 or 1	Simple	Simple	Medium
2 or 3	Simple	Medium	Difficult
4 +	Medium	Difficult	Difficult

Table 9 (b): For reports

- Once complexity is determined, the number of screens, reports & components are weighted according to the following tables :

Object Type	Complexity Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Reports	2	5	8
3GL Components			10

Software Project Planning

vi. Calculation of productivity rate: The productivity rate can be calculated as:

Productivity rate (PROD) = NOP/Person month

Developer's experience & capability; ICASE maturity & capability	*PROD (NOP/PM)*
Very low	4
Low	7
Nominal	13
High	25
Very high	50

Table 11: Productivity values

Software Project Planning

iv. Determine object points: Add all the weighted object instances to get one number and this known as object-point count.

v. Compute new object points: We have to estimate the percentage of reuse to be achieved in a project. Depending on the percentage reuse, the new object points (NOP) are computed.

$$NOP = \frac{(\text{object points}) * (100 - \%\text{reuse})}{100}$$

NOP are the object points that will need to be developed and differ from the object point count because there may be reuse.

- The **object point count** = **multiplying the original number of object instances by the weighting factor & summing to obtain a total object point count.**

- When component based development or general s/w reuse is to be applied, the %reuse is estimated & object point count is adjusted.

Software Project Planning

vii. Compute the effort in Persons-Months: When PROD is known, we may estimate effort in Person-Months as:

$$\textbf{Effort in PM} = \frac{NOP}{PROD}$$

Example: 4.9

Consider a database application project with the following characteristics:

I. The application has 4 screens with 4 views each and 7 data tables for 3 servers and 4 clients.

II. The application may generate two report of 6 sections each from 07 data tables for two server and 3 clients. There is 10% reuse of object points.

The developer's experience and capability in the similar environment is low. The maturity of organization in terms of capability is also low. Calculate the object point count, New object points and effort to develop such a project.

Solution

This project comes under the category of application composition estimation model.

Number of screens = 4 with 4 views each

Number of reports = 2 with 6 sections each

From Table 9 we know that each screen will be of medium complexity and each report will be difficult complexity.

Using Table 10 of complexity weights, we may calculate object point count.

$$= 4 \times 2 + 2 \times 8 = 24$$

$$\mathbf{NOP} = \frac{24 * (100 - 10)}{100} = 21.6$$

Table 11 gives the low value of productivity (PROD) i.e. 7.

$$\text{Efforts in PM} = \frac{\text{NOP}}{\text{PROD}}$$

$$\text{Efforts} = \frac{21.6}{7} = 3.086 \text{ PM}$$

Example 2:

Consider the database application with following information:

It has 5 screens with 5 views from 6 data table for 3 servers and 4 clients, it may generate 3 reports of 5 sections from 5 data tables for 2 servers and 3 clients and 4 3GL components for the same application.

There is 20 % reuse of object points. Developers experience and capability is low.

Calculate object point count and new object point counts and efforts.

The Early Design Model

The COCOMO-II models use the base equation of the form

$$PM_{nominal} = A * (size)^{B}$$

where

$PM_{nominal}$ = Effort of the project in person months

A = Constant representing the nominal productivity, provisionally set to 2.5

B = Scale factor

Size = Software size

Scale factors

- This model uses five factors that affects B value
- They are

1. Precedentness (PREC)
2. Development Flexibility (FLEX)
3. Risk Resolution (RESL)
4. Team Cohesion (TEAM)
5. Process Maturity (PMAT)

Software Project Planning

Scale factor	Explanation	Remarks
Precedentness	Reflects the previous experience on similar projects. This is applicable to individuals & organization both in terms of expertise & experience	Very low means no previous experiences, Extra high means that organization is completely familiar with this application domain.
Development flexibility	Reflect the degree of flexibility in the development process.	Very low means a well defined process is used. Extra high means that the client gives only general goals.
Architecture/ Risk resolution	Reflect the degree of risk analysis carried out.	Very low means very little analysis and Extra high means complete and through risk analysis.

Cont...

Table 12: Scaling factors required for the calculation of the value of B

Software Project Planning

Scale factor	Explanation	Remarks
Team cohesion	Reflects the team management skills.	Very low means no previous experiences, Extra high means that organization is completely familiar with this application domain.
Process maturity	Reflects the process maturity of the organization. Thus it is dependent on SEI-CMM level of the organization.	Very low means organization has no level at all and extra high means organization is related as highest level of SEI-CMM.

Table 12: Scaling factors required for the calculation of the value of B

Software Project Planning

Scaling factors	Very low	Low	Nominal	High	Very high	Extra high
Precedent ness	6.20	4.96	3.72	2.48	1.24	0.00
Development flexibility	5.07	4.05	3.04	2.03	1.01	0.00
Architecture/ Risk resolution	7.07	5.65	4.24	2.83	1.41	0.00
Team cohesion	5.48	4.38	3.29	2.19	1.10	0.00
Process maturity	7.80	6.24	4.68	3.12	1.56	0.00

Table 13: Data for the Computation of B

The value of B can be calculated as:

B=0.91 + 0.01 * (Sum of rating on scaling factors for the project)

B= 0.91+0.01($\sum_{1}^{5}$ ratings)

Early design cost drivers

There are seven early design cost drivers and are given below:

i. Product Reliability and Complexity (RCPX)

ii. Required Reuse (RUSE)

iii. Platform Difficulty (PDIF)

iv. Personnel Capability (PERS)

v. Personnel Experience (PREX)

vi. Facilities (FCIL)

vii. Schedule (SCED)

Software Project Planning

The seven early design cost drivers have been converted into numeric values with a Nominal value 1.0. These values are used for the calculation of a factor called "Effort multiplier" which is the product of all seven early design cost drivers. The numeric values are given in Table 15.

Early design Cost drivers	*Extra Low*	*Very Low*	*Low*	*Nominal*	*High*	*Very High*	*Extra High*
RCPX	.73	.81	.98	1.0	1.30	1.74	2.38
RUSE	—	—	0.95	1.0	1.07	1.15	1.24
PDIF	—	—	0.87	1.0	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1.0	0.83	0.63	0.50
PREX	1.59	1.33	1.12	1.0	0.87	0.71	0.62
FCIL	1.43	1.30	1.10	1.0	0.87	0.73	0.62
SCED	—	1.43	1.14	1.0	1.0	1.0	—

Table 15: Early design parameters

Software Project Planning

The early design model adjusts the nominal effort using 7 effort multipliers (EMs). Each effort multiplier (also called drivers) has 7 possible weights as given in Table 15. These factors are used for the calculation of adjusted effort as given below:

$$PM_{adjusted} = PM_{\text{nominal}} \times \left[\prod_{i=7}^{7} EM_i \right]$$

$PM_{adjusted}$ effort may very even up to 400% from $PM_{nominal}$

Hence $PM_{adjusted}$ is the fine tuned value of effort in the early design phase

Software Project Planning

Example: 4.10

A software project of application generator category with estimated 50 KLOC has to be developed. The scale factor (B) has low precedentness, high development flexibility and low team cohesion. Other factors are nominal. The early design cost drivers like platform difficult (PDIF) and Personnel Capability (PERS) are high and others are nominal. Calculate the effort in person months for the development of the project.

Solution

Here $B = 0.91 + 0.01 * (\text{Sum of rating on scaling factors for the project})$

$= 0.91 + 0.01 * (4.96 + 2.03 + 4.24 + 4.38 + 4.68)$

$= 0.91 + 0.01(20.29) = 1.1129$

$PM_{nominal} = A*(size)^B$

$= 2.5 * (50)^{1.1129} = 194.41$ Person months

The 7 cost drivers are

PDIF = high (1.29)

PERS = high (0.83)

RCPX = nominal (1.0)

RUSE = nominal (1.0)

PREX = nominal (1.0)

FCIL = nominal (1.0)

SCEO = nominal (1.0)

$$PM_{adjusted} = PM_{\text{nominal}} \times \left[\prod_{i=7}^{7} EM_i \right]$$

= 194.41 * [1.29 x 0.83)

= 194.41 x 1.07

= 208.155 Person months

Post Architecture Model

The post architecture model is the most detailed estimation model and is intended to be used when a software life cycle architecture has been completed. This model is used in the development and maintenance of software products in the application generators, system integration or infrastructure sectors.

$$PM_{adjusted} = PM_{\text{nominal}} \times \left[\prod_{i=7}^{17} EM_i \right]$$

EM : Effort multiplier which is the product of 17 cost drivers.

The 17 cost drivers of the Post Architecture model are described in the table 16.

Cost Driver	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
RELY	0.75	0.88	1.00	1.15	1.39	
DATA		0.93	1.00	1.09	1.19	
CPLX	0.75	0.88	1.00	1.15	1.30	1.66
RUSE		0.91	1.00	1.14	1.29	1.49
DOCU	0.89	0.95	1.00	1.06	1.13	
TIME			1.00	1.11	1.31	1.67
STOR			1.00	1.06	1.21	1.57
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.50	1.22	1.00	0.83	0.67	
PCAP	1.37	1.16	1.00	0.87	0.74	

Table 18: 17 Cost Drivers

Cont…

Cost Driver	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
PCON	1.24	1.10	1.00	0.92	0.84	
AEXP	1.22	1.10	1.00	0.89	0.81	
PEXP	1.25	1.12	1.00	0.88	0.81	
LTEX	1.22	1.10	1.00	0.91	0.84	
TOOL	1.24	1.12	1.00	0.86	0.72	
SITE	1.25	1.10	1.00	0.92	0.84	0.78
SCED	1.29	1.10	1.00	1.00	1.00	

Table 18: 17 Cost Drivers

Cost Driver	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
RELY	0.75	0.88	1.00	1.15	1.39	
DATA		0.93	1.00	1.09	1.19	
CPLX	0.75	0.88	1.00	1.15	1.30	1.66
RUSE		0.91	1.00	1.14	1.29	1.49
DOCU	0.89	0.95	1.00	1.06	1.13	
TIME			1.00	1.11	1.31	1.67
STOR			1.00	1.06	1.21	1.57
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.50	1.22	1.00	0.83	0.67	
PCAP	1.37	1.16	1.00	0.87	0.74	
PCON	1.24	1.10	1.00	0.92	0.84	
AEXP	1.22	1.10	1.00	0.89	0.81	
PEXP	1.25	1.12	1.00	0.88	0.81	
LTEX	1.22	1.10	1.00	0.91	0.84	
TOOL	1.24	1.12	1.00	0.86	0.72	
SITE	1.25	1.10	1.00	0.92	0.84	0.78
SCED	1.29	1.10	1.00	1.00	1.00	

Schedule estimation

Development time can be calculated using $PM_{adjusted}$ as a key factor and the desired equation is:

$$TDEV_{\text{nominal}} = [\phi \times (PM_{adjusted})^{(0.28+0.2(B-0.091))}] * \frac{SCED\%}{100}$$

where Φ = constant, provisionally set to 3.67

$TDEV_{\text{nominal}}$ = calendar time in months with a scheduled constraint

B = Scaling factor

$PM_{adjusted}$ = Estimated effort in Person months (after adjustment)

Example: 4.11

Consider the software project given in example 4.10. Size and scale factor (B) are the same. The identified 17 Cost drivers are high reliability (RELY), very high database size (DATA), high execution time constraint (TIME), very high analyst capability (ACAP), high programmers capability (PCAP). The other cost drivers are nominal. Calculate the effort in Person-Months for the development of the project.

Solution

Here B = 1.1129

$PM_{nominal}$ = 194.41 Person-months

$$PM_{adjusted} = PM_{nominal} \times \left[\prod_{i=7}^{17} EM_i \right]$$

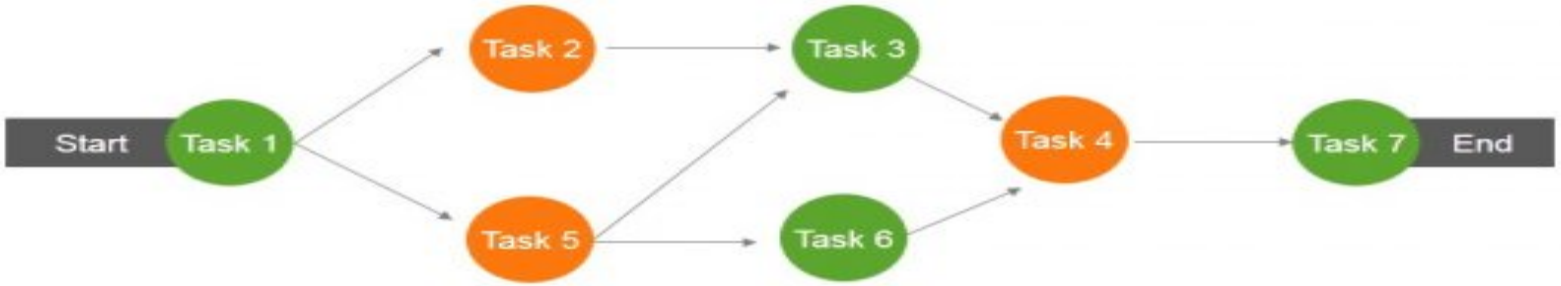
= 194.41 x (1.15 x 1.19 x 1.11 x 0.67 x 0.87)

= 194.41 x 0.885

= 172.05 Person-months

Parameters	COCOMO I	COCOMO II
Development Life Cycle	Useful in waterfall models	Useful in non-sequential, rapid development, reengineering and reuse models of software.
Size	Delivered Source Instructions (thousands) i.e. KDSI as an input.	Object Points or FP or KSLOC.
Equation Exponent	Effort equation's exponent is determined by 3 development modes.	Effort equation's exponent is determined by 5 scale factors.
Cost Driver	15 cost drivers attributes	17cost drivers attributes
Estimation Accuracy	It provides estimates of effort and schedule.	Provides estimates that represent one standard deviation around the most likely estimate.
Data Points	63 Projects Referred	161 Projects Referred
Model Difference	Model based upon 1. Linear reuse formula 2. Assumption of reasonably stable requirements.	Other enhancements : 1. Non Linear reuse formula 2. Reuse model which looks at effort needed to understand & assimilate.

Activity Network Diagram



This slide is 100% editable. Adapt it to your needs and capture your audience's attention.

WWW.COMPANY.COM

- Managers can estimate time duration in several ways

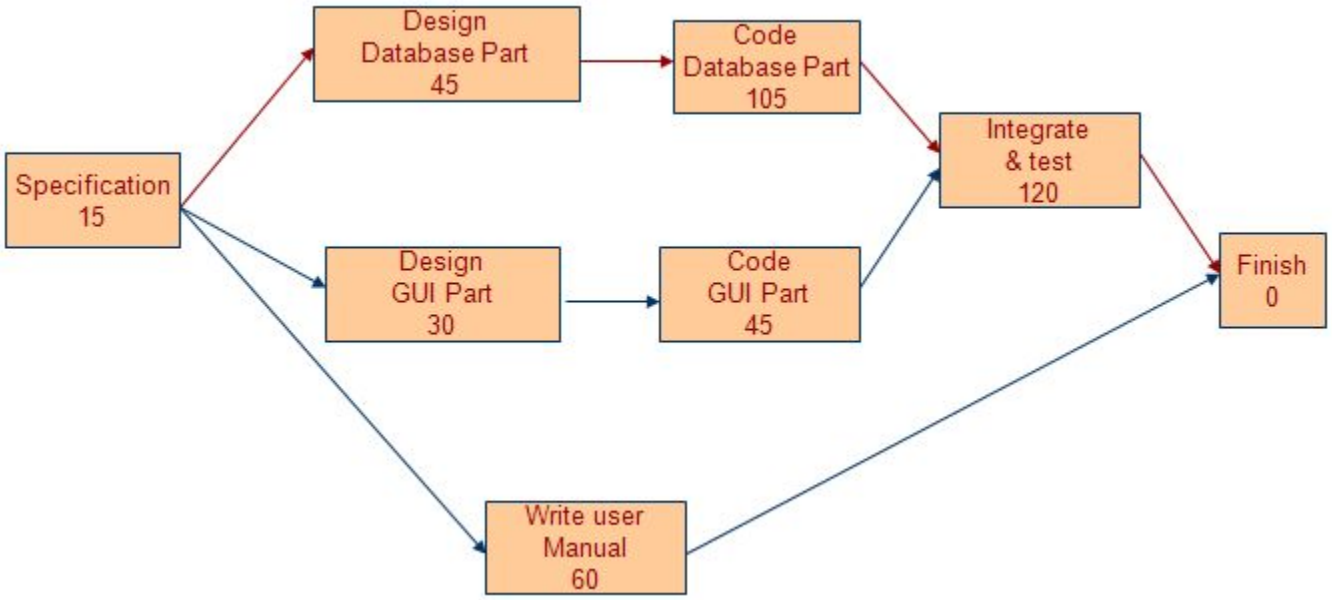
  1. Empirically assign duration to different tasks
  2. Allow each engineer to estimate the time by himself

Critical Path Method (CPM)

- From the activity network representation, following analysis can be made:
  - The minimum time (MT) to complete the project is the maximum of all paths from start to finish
  - The earliest start (ES) time of a task is the maximum of all paths from the start to this task.
  - The latest start (LS) time is the difference between MT & maximum of all paths from this task to the finish.
- The earliest finish (EF) time of the task is the sum of the earliest start time of the task & the duration of the task.
- The latest finish (LF) time of a task can be obtained by subtracting maximum of all paths from the task to finish from MT. (excluding current task)

- The Slack time (ST) is LS – EF OR LF-EF. It is the total time for which a task may be delayed before it would affect the finish time of the project.
- A critical task is one with 0 slack time
- A path from the start node to the finish node containing only critical task is called a critical path

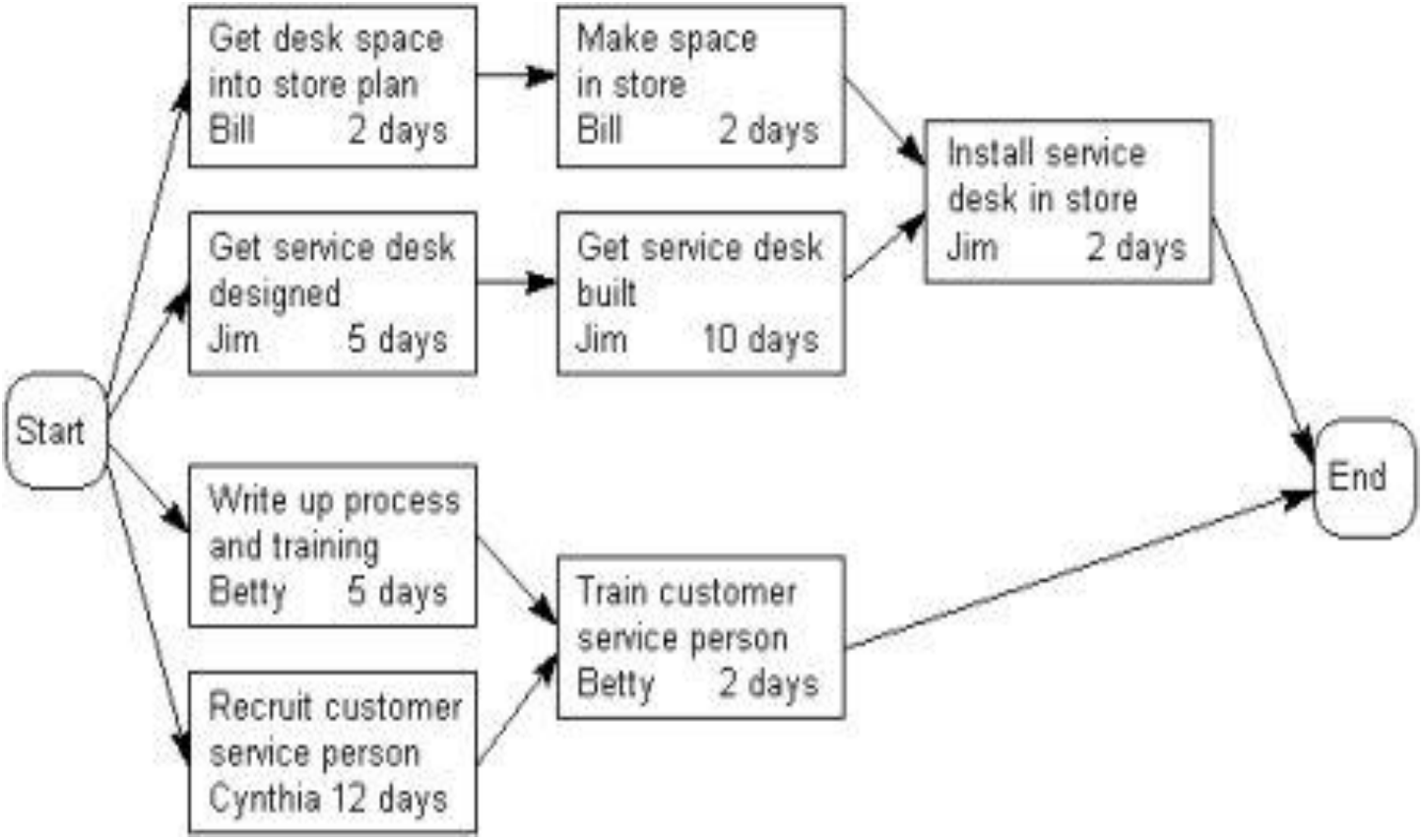
Critical Path Method Numerical



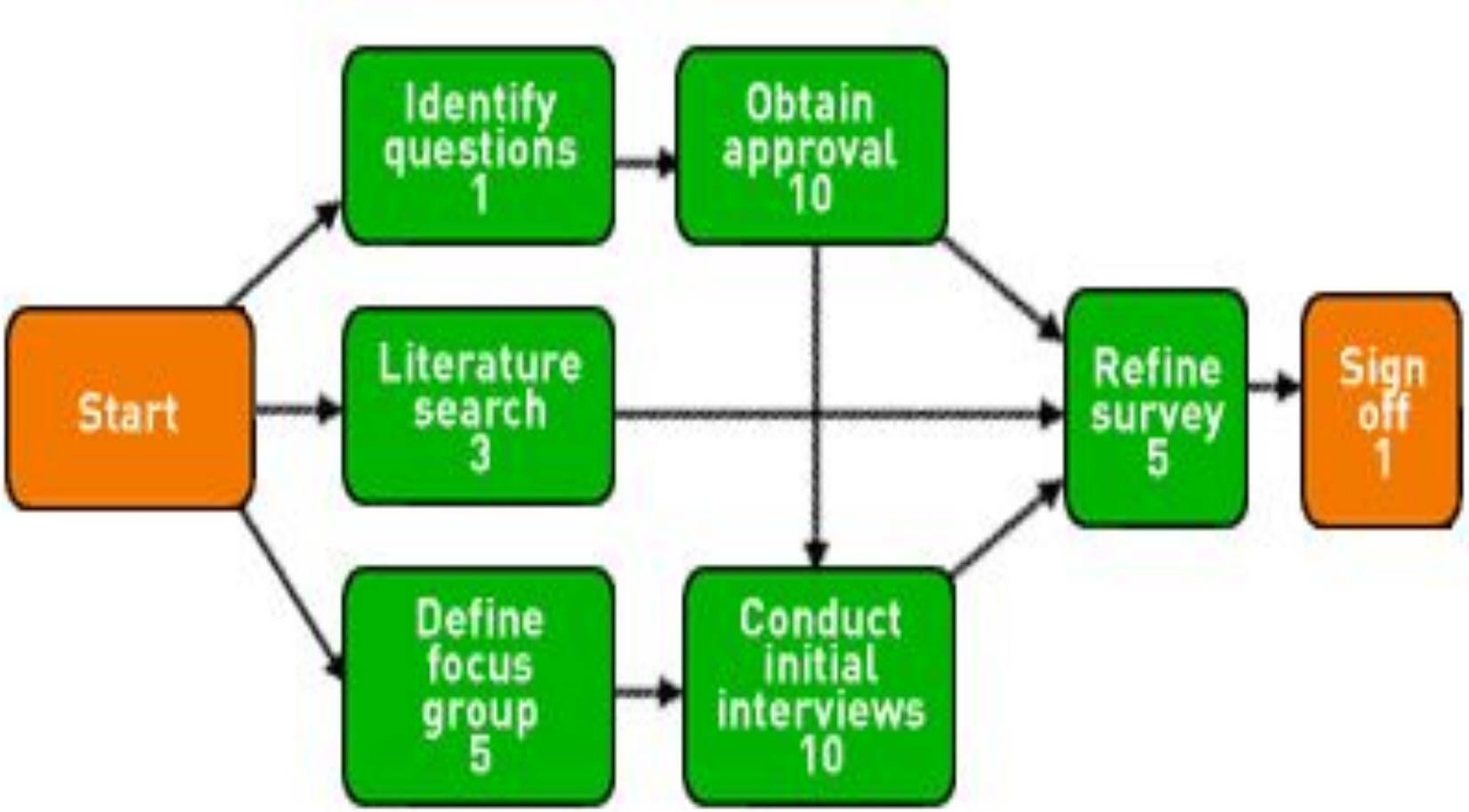
- The above parameters for different tasks for the MIS problem are as follows :

Task	ES	EF	LS	LF	ST
Specification Part	0	15	0	15	0
Design Database Part	15	60	15	60	0
Design GUI Part	15	45	90	120	75
Code Database Part	60	165	60	165	0
Code GUI Part	45	90	120	165	75
Integrate & Test	165	285	165	285	0
Write User Manual	15	75	225	285	210

Find the critical Path:



Find the critical Path:



questions

- What is software project estimation
- What is cocomo model
- Difference between cocomo I and cocomo II

Any questions ??????