

Program Evaluation and Review Technique (PERT)

PERT (Program Evaluation and Review Technique) is a probabilistic method, where the activity times are represented by a probability distribution. This distribution of activity times is based on three different estimates made for each activity.

PERT weighted average =

$$\frac{\text{optimistic time} + 4 \times \text{most likely time} + \text{pessimistic time}}{6}$$

6

CPM vs PERT: Comparison Table

Aspect	Critical Path Method (CPM)	Program Evaluation and Review Technique (PERT)
Time Estimation	Single deterministic time estimate	Three time estimates: optimistic (O), most likely (M), and pessimistic (P)
Time Calculation	Fixed duration for each activity	Expected time = $(O + 4M + P) \div 6$
Approach	Deterministic approach	Probabilistic approach
Focus	Activity-oriented; emphasizes time-cost tradeoffs	Event-oriented; emphasizes managing uncertainty
Origin	Developed by DuPont and Remington Rand (1957)	Developed by U.S. Navy for Polaris missile program (1957)
Best For	Routine, repetitive projects with known durations	Complex projects with uncertain activity durations
Risk Management	Limited probabilistic analysis	Calculates standard deviations and completion probabilities
Resource Handling	Includes resource optimization techniques	Less emphasis on resource allocation
Cost Analysis	Strong focus on cost-time relationship	Less emphasis on cost analysis
Typical Applications	Construction, maintenance, manufacturing	Research, development, innovative projects
Schedule Compression	Contains "crashing" concepts for reducing durations	Does not typically include crashing techniques
Visualization	Arrow/Precedence Diagram Method	Similar network diagrams but with probabilistic calculations

Key Elements of PERT

- **Task Identification:** Breaking down the project into specific activities or tasks
- **Task Sequencing:** Determining dependencies between tasks (which tasks must be completed before others can begin)
- **Time Estimation:** PERT uniquely uses three time estimates for each activity:
 - Optimistic time (O): Shortest possible completion time
 - Most likely time (M): Most realistic completion time
 - Pessimistic time (P): Longest possible completion time
- **Expected Time Calculation:** Using the formula: $(O + 4M + P) \div 6$
- **Critical Path Identification:** Determining the sequence of activities that represents the longest path through the network
- **Probability Analysis:** Using statistical methods to determine the likelihood of completing the project by a specific date

Expected Time Calculation in PERT

The formula $(O + 4M + P) \div 6$ is a weighted average used in PERT to calculate the expected time for completing an activity. Here's a breakdown of what this means:

Components of the Formula:

- **O (Optimistic time):** The minimum possible time required to complete the activity, assuming everything goes perfectly (best-case scenario)
- **M (Most likely time):** The realistic estimate of time required under normal conditions (most probable scenario)
- **P (Pessimistic time):** The maximum possible time required, assuming everything goes wrong (worst-case scenario)
- **4M:** The most likely time is weighted four times because it's considered the most reliable estimate

Who Makes the Three Time Estimates in PERT?

Primary Estimators

1. Subject Matter Experts (SMEs)
 - Technical specialists with direct experience performing similar tasks
 - People with the most knowledge about specific activities or work packages
 - Often the individuals who will actually perform the work
2. Project Team Members
 - Team leads responsible for executing the activities
 - Those with hands-on experience with similar tasks in past projects
 - Engineers, developers, or technical staff involved in implementation
3. Project Managers
 - May contribute to or finalize estimates
 - Often coordinate the estimation process
 - Responsible for reviewing estimates for consistency and reasonableness

Who Makes the Three Time Estimates in PERT?

Supporting Contributors

1. Historical Data Analysts
 - People who analyze records from similar past projects
 - Those who maintain organizational process assets and lessons learned
2. External Consultants
 - Specialists brought in for their expertise in similar projects
 - Industry experts with broader experience across multiple organizations
3. Stakeholders
 - In some cases, clients or business representatives with relevant experience
 - Department managers who oversee similar operations

1. Expected Duration Calculation

Given optimistic time = 4 days, most likely time = 7 days, and pessimistic time = 16 days, what is the expected duration of this activity?

Answer:

Expected duration = $(O + 4M + P) \div 6$

= $(4 + 4 \times 7 + 16) \div 6$

= $(4 + 28 + 16) \div 6$

= $48 \div 6$

= 8 days

Perform PERT Analysis on following

Activity	Predecessor	Duration
A	-	4,4,7
B	A	4,6,10
C	A	4,6,8
D	B	6,7,8
E	C	3,4,5
F	D,E	4,6,7

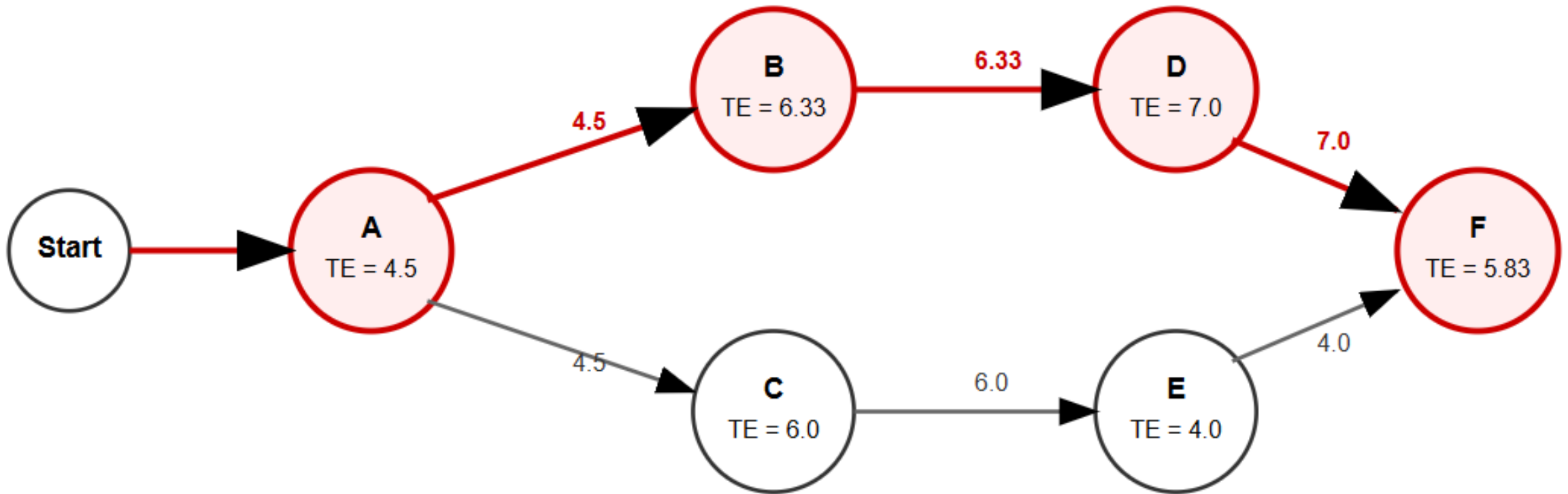
1. How long will it take to complete each path (how many?)
2. Which path is the critical path?

Step 1: Calculate Expected Duration for Each Activity

For each activity, I'll use the PERT formula: Expected time (TE) = $(O + 4M + P) \div 6$

Activity	Optimistic (O), Most Likely (M), Pessimistic (P)	Expected Duration (TE)
A	4, 4, 7	$(4 + 16 + 7)/6 = 4.5$
B	4, 6, 10	$(4 + 24 + 10)/6 = 6.33$
C	4, 6, 8	$(4 + 24 + 8)/6 = 6.00$
D	6, 7, 8	$(6 + 28 + 8)/6 = 7.00$
E	3, 4, 5	$(3 + 16 + 5)/6 = 4.00$
F	4, 6, 7	$(4 + 24 + 7)/6 = 5.83$

PERT Activity Network Diagram



Step 2: Identify All Possible Paths

Based on the predecessor relationships:

- Path 1: $A \rightarrow B \rightarrow D \rightarrow F$
- Path 2: $A \rightarrow C \rightarrow E \rightarrow F$

Step 3: Calculate Duration of Each Path

- Path 1: $A \rightarrow B \rightarrow D \rightarrow F = 4.5 + 6.33 + 7.0 + 5.83 = 23.66$ time units
- Path 2: $A \rightarrow C \rightarrow E \rightarrow F = 4.5 + 6.0 + 4.0 + 5.83 = 20.33$ time units

Step 4: Identify the Critical Path

- The critical path is the path with the longest duration:
Path 1 ($A \rightarrow B \rightarrow D \rightarrow F$) = 23.66 time units
- Therefore, Path 1 ($A \rightarrow B \rightarrow D \rightarrow F$) is the critical path, and the expected project completion time is 23.66 time units.
- This network diagram clearly shows why $A \rightarrow B \rightarrow D \rightarrow F$ is the critical path - it's the longest path through the network and determines the minimum project completion time. Any delay in activities on this path will directly delay the entire project.

Class Activity

Perform PERT Analysis on following

Activity	Predecessor	Duration
A	-	3,4,7
B	A	3,6,10
C	A	3,6,8
D	B	4,7,8
E	C	4,4,5
F	D,E	4,6,7

1. How long will it take to complete each path (how many?)
2. Which path is the critical path?

Function Point Analysis (FPA)

- Function Point Analysis (FPA) is a standardized method in software engineering **used to measure the size and complexity of a software application** from a user's perspective. Unlike lines of code, which are language-dependent, function points provide a **language-independent, technology-independent, and vendor-independent** way of estimating software size.

What is a Function Point?

- A Function Point (FP) is a unit of measurement that quantifies the functional requirements of a software system. It reflects what the software does (functionality) rather than how it is developed (code).

What are Unadjusted Function Points (UFP)?

- Unadjusted Function Points are the raw function points calculated before applying any complexity or environmental adjustment (i.e., before the Value Adjustment Factor or VAF).
- They are based solely on the count and complexity (Low, Average, High) of five types of components:
 1. External Inputs (EI)
 2. External Outputs (EO)
 3. External Inquiries (EQ)
 4. Internal Logical Files (ILF)
 5. External Interface Files (EIF)

Value Adjustment Factor (VAF)

The Value Adjustment Factor is like a fine-tuning tool. After you count the basic features of a software system (Unadjusted Function Points), VAF adjusts the size based on how complex or demanding the system really is. It considers 14 general system characteristics, such as:

- ☐ How fast the system needs to be
- ☐ How secure it must be
- ☐ How often it changes
- ☐ How easy it is to use

Each one is rated from 0 (not important) to 5 (very important). The total score is used to calculate the VAF.

$$\text{VAF} = 0.65 + (0.01 \times \text{Total Influence Score})$$

Components of Function Point Analysis

FPA breaks down software into five major functional components:

- **External Inputs (EI)**

User-originated inputs that update internal data (e.g., forms, data entry screens).

- **External Outputs (EO)**

Outputs sent to the user that require processing logic (e.g., reports, messages).

- **External Inquiries (EQ)**

User requests that retrieve data without updating it (e.g., search queries).

- **Internal Logical Files (ILF)**

User-recognizable groups of logically related data maintained by the system.

- **External Interface Files (EIF)**

Files used by the system but maintained by external systems.

1. External Inputs (EI)

These are inputs coming from the user or external systems to the application, typically used to create or update internal data.

- Example: A "User Registration" form where users enter their name, email, and password.
- Why it's EI: It captures data and stores it in the system's internal user database.

2. External Outputs (EO)

These are outputs that the system sends to the user or another system, often involving calculations or derived data.

- **Example:** A "Monthly Sales Report" that summarizes total revenue, number of sales, and top-performing products.
- **Why it's EO:** It gathers data, performs calculations, and formats it for display.

3. External Inquiries (EQ)

These involve data retrieval with minimal or no processing, and do not update the internal data.

- Example: A "Product Search" feature where users can look up items by name or category.
- Why it's EQ: It retrieves information but does not alter it or involve complex logic.

4. Internal Logical Files (ILF)

These are logically related groups of data maintained by the application (its own database).

- Example: A "Customer" database storing customer names, contact info, and purchase history.
- Why it's ILF: The system creates, reads, updates, and deletes this data.

5. External Interface Files (EIF)

These are logical groups of data that the system uses but does not maintain—they are maintained by another system.

- Example: An inventory system reads "Product Info" from a centralized ERP system's database.
- Why it's EIF: The data is used for processing but is owned and updated by a different application.

Steps in Function Point Analysis

1. Identify and classify all components (EI, EO, EQ, ILF, EIF).
2. Assign complexity (Low, Average, High) to each component based on defined criteria.
3. Apply weights to each component using a standard table:

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

4. Calculate Unadjusted Function Points (UFP) by summing all weighted counts.
5. Adjust for complexity/environment using the Value Adjustment Factor (VAF), which accounts for general system characteristics (GSCs), such as performance, reusability, and complexity.

Case study: Library Management System

Features:

1. Add new members
2. Add new books
3. Search for books
4. Borrow a book
5. Return a book
6. Generate overdue reports

Step 1: Identify Functional Components

Feature	Component Type	Description
Add new members	External Input (EI)	Input data to register a new library member
Add new books	External Input (EI)	Input data to register a new book
Search for books	External Inquiry (EQ)	Users search for books by title/author
Borrow a book	External Input (EI)	Update book and user status
Return a book	External Input (EI)	Update book and user status
Overdue report	External Output (EO)	Generate a list of overdue books and fines
Member records	Internal Logical File (ILF)	Data stored about library members
Book records	Internal Logical File (ILF)	Data stored about books
Fines data from finance system	External Interface File (EIF)	Fines are calculated using external finance module

Step 2: Assign Complexity and Weights

Let's assume most are **Average** complexity.

Component Type	Count	Complexity	Weight per item	Subtotal
EI	4	Average	4	$4 \times 4 = 16$
EQ	1	Average	4	$1 \times 4 = 4$
EO	1	Average	5	$1 \times 5 = 5$
ILF	2	Average	10	$2 \times 10 = 20$
EIF	1	Average	7	$1 \times 7 = 7$

Unadjusted Function Point (UFP) Total = $16 + 4 + 5 + 20 + 7 = 52$

Step 3: Calculate Value Adjustment Factor (VAF)

Assess 14 **General System Characteristics (GSCs)** such as performance, complexity, reusability, etc., each rated from 0 (no influence) to 5 (strong influence). Let's assume the **total score is 30**.

Formula:

$$\text{VAF} = 0.65 + (0.01 \times \text{Total GSC score}) = 0.65 + (0.01 \times 30) = 0.95$$

Step 4: Final Function Point Count

$$\text{Final FP} = \text{UFP} \times \text{VAF} = 52 \times 0.95 = 49.4 \approx 49 \text{ function points}$$

Assumptions

- Effort per Function Point = 8 hours
- Cost per Function Point = ₹6,000
- Team capacity = 160 hours/month per developer

1. 🕒 **Total Effort:**

$$\text{Effort} = 49 \times 8 = \boxed{392 \text{ hours}}$$

2. 💰 **Total Cost:**

$$\text{Cost} = 49 \times ₹6,000 = \boxed{₹294,000}$$

3. 📅 **Estimated Timeline** (with 1 developer):

$$\text{Timeline} = \frac{392 \text{ hours}}{160 \text{ hours/month}} = \boxed{2.45 \text{ months}} \approx 2.5 \text{ months}$$

The COCOMO (Constructive Cost Model)

- COCOMO (Constructive Cost Model) is a regression model developed by Barry Boehm in 1981 to estimate software development effort, cost, and schedule. It's one of the most widely used algorithmic software cost estimation models.
- A regression model is a type of statistical or mathematical model used to predict the value of a dependent variable (output) based on one or more independent variables (inputs).
- In the context of COCOMO, the regression model:
 - ❑ Estimates effort, time, or cost (**dependent variables**)
 - ❑ Based on input factors like KLOC (thousand lines of code) and cost drivers (**independent variables**)

Project Categories in COCOMO

COCOMO defines three categories (or *modes*) based on team experience, tools, and project complexity:

Project Mode	Description	
Organic	<ul style="list-style-type: none">- Small teams- Familiar domain- Simple, well-understood requirements	$a = 2.4, b = 1.05$
Semi-detached	<ul style="list-style-type: none">- Mixed teams- Moderate experience- Intermediate complexity	$a = 3.0, b = 1.12$
Embedded	<ul style="list-style-type: none">- Complex systems- Hardware + software interaction- Tight constraints	$a = 3.6, b = 1.20$

Types of COCOMO Models

1. Basic COCOMO

- Basic COCOMO (Constructive Cost Model) is the simplest form of Boehm's cost estimation model. It estimates the effort (in person-months) required to develop software based solely on the size of the project in KLOC (Thousands of Lines of Code).
- Formula: $\text{Effort} = a \times (\text{KLOC})^b$
 - KLOC = Thousand Lines of Code
 - a, b = constants based on project complexity

Effort

- Definition: Total amount of work required to complete a project.
- Unit: Person-months
- Meaning: One person working full-time for one month.

Development Time

- Definition: The calendar time it takes to complete the project.
- Unit: Months
- Depends on: Effort and team size (how many people are working on it).

Formula

$$\text{Effort (E)} = a \times (\text{KLOC})^b$$

Where:

- **E** = Effort in person-months (One person working for X month)
- **KLOC** = Estimated size of the software (in 1000 lines of code)
- **a** and **b** = Constants that depend on the **type of project** (Organic, Semi-detached, or Embedded)

- In the COCOMO model, a and b are constants that determine how much effort is required to build software based on its size.
- These constants are derived from regression analysis of historical project data and depend on the type of software project.

Example

Let's say you're developing a simple inventory management system, estimated to be 32 KLOC, and it falls under the Organic mode.

- $a = 2.4$
- $b = 1.05$

Step 1: Apply the formula:

$$E = 2.4 \times (32)^{1.05}$$

$$E \approx 2.4 \times 35.53 \approx 85.27 \text{ person-months}$$

It means that the total effort required to complete the project is equivalent to one person working full-time for 85.27 months.

Estimating Development Time

$$\text{Development Time (D)} = c \times (E)^d$$

In Organic mode, typical values are:

- $c = 2.5$, $d = 0.38$

$$D = 2.5 \times (85.27)^{0.38} \approx 2.5 \times 6.15 \approx 15.38 \text{ months}$$

If you have a team of **6 people**:

$$\text{Development Time} = \frac{85.27}{6} \approx 14.21 \text{ months}$$

So, 6 people working full-time would complete the project in about **14.2 months**.

People Required

$$P = \frac{E}{D} = \frac{85.27}{15.38} \approx 5.5 \text{ persons}$$

Constant values

Software project	a_b	b_b	c_b	d_b
● 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

<i>Mode</i>	<i>Effort</i>	<i>Schedule</i>
Organic	$E = 2.4 * (KDSI)^{1.05}$	$TDEV = 2.5 * (E)^{0.38}$
Semidetached	$E = 3.0 * (KDSI)^{1.12}$	$TDEV = 2.5 * (E)^{0.35}$
Embedded	$E = 3.6 * (KDSI)^{1.20}$	$TDEV = 2.5 * (E)^{0.32}$

Basic COCOMO Model: Example



E.g. 2: We have determined our project fits the characteristics of **Semi-Detached** mode

- We estimate our project will have **32,000** Delivered Source Instructions. Using the formulas, we can estimate:
- **Effort** = $3.0 \times (32)^{1.12}$ = 146 man-months
- **Schedule** = $2.5 \times (146)^{0.35}$ = 14 months
- **Productivity** = 32,000 DSI / 146 MM
= 219 DSI/MM
- **Average Staffing** = 146 MM / 14 months
= 10 FSP

Basic COCOMO Model: Example



E.g.1: Suppose that a project was estimated to be 400 KLOC.
Calculate the effort and development time for each of the three modes i.e., organic, semidetached and embedded

The basic COCOMO equation take the form:

$$E = a_b(KLOC)^{b_b}$$

$$D = c_b(E)^{d_b}$$

Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi detected	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

consider a software project using semi-detached mode with 30,000 lines of code . We will obtain estimation for this project as follows:

(1) Effort estimation

$$E = a_b(KLOC) \text{Exp}(b_b) \text{ person-months}$$

$$E = 3.0(30)^{1.12} \text{ where lines of code} = 30000 = 30 \text{ KLOC}$$

$$E = 135 \text{ person-month}$$

(2) Duration estimation

$$D = c_b(E) \text{Exp}(d_b) \text{ months}$$

$$= 2.5(135)^{0.35}$$

$$D = 14 \text{ months}$$

(3) Person estimation

$$P = E/D$$

$$= 135/14$$

$$P = 10 \text{ persons approx.}$$

Limitations of Basic COCOMO Model :



1. The accuracy of this model is limited because it does not consider certain factors for cost estimation of software.
2. These factors are hardware constraints, personal quality and experiences, modern techniques and tools.
3. The estimates of COCOMO model are within a factor of 1.3 only 29% of the time.

2. Intermediate COCOMO

Intermediate COCOMO builds on the Basic COCOMO model by adding 15 cost drivers (also called effort multipliers) to account for various real-world project factors like product complexity, team capability, tools, and system constraints.

Cost Drivers (15 Factors)

Grouped into 4 categories:

- **Product attributes**

- Required software reliability
- Size of application database
- Complexity of the product

- **Hardware attributes**

- Run-time performance constraints
- Memory constraints
- Volatility of the virtual machine environment
- Required turnabout time

- **Personnel attributes**

- Analyst capability
- Software engineer capability
- Applications experience
- Virtual machine experience
- Programming language experience

- **Project attributes**

- Use of software tools
- Application of software engineering methods
- Required development schedule

Intermediate COCOMO Contd.



1. Each of the 15 attributes receives a rating on a 6-point scale that ranges from "very low" to "extra high" (in importance)
2. Based on the rating, effort multipliers is determined. The product of all effort Multipliers result in "effort adjustment factor" (EAF).
3. Typical values for EAF range from 0.9 to 1.4.

Formula

$$\text{Effort (E)} = a \times (\text{KLOC})^b \times \text{EAF}$$

Where:

- **E** = Effort (person-months)
- **KLOC** = Thousands of Lines of Code
- **a**, **b** = Constants based on project type (Organic, Semi-detached, Embedded)
- **EAF** = **Effort Adjustment Factor** (product of 15 cost driver ratings)

Example:

Consider a project having 30,000 lines of code which in an embedded software with critical area hence reliability is high. The estimation can be



$$E = a_i (KLOC)^{b_i} * (EAF)$$

As reliability is high $EAF = 1.15$ (product attribute)

$$a_i = 2.8$$

$$b_i = 1.20 \quad \text{for embedded software}$$

$$E = 2.8(30)^{1.20} * 1.15$$

$$= 191 \text{ person month}$$

$$D = c_b(E)^{d_b} = 2.5(191)^{0.32}$$

$$= 13 \text{ months approximately}$$

$$P = E/D$$

$$= 191/13$$

$$P = 15 \text{ persons approx.}$$

3. Detailed COCOMO

- Incorporates all characteristics of Intermediate model
- Adds phase-sensitive effort multipliers
- Considers impact of cost drivers at different development phases

COCOMO II

COCOMO II (released in 2000) is an updated version that addresses:

- Non-sequential and rapid-development software processes
- Reuse-driven approaches
- Object-oriented development
- Software process maturity effects

COCOMO II includes:

- Application composition model (early prototyping)
- Early design model (when requirements are established)
- Post-architecture model (detailed estimation during development)

Advantages and Limitations

- Well-documented and transparent methodology
- Adjustable to various project types
- Incorporates many factors affecting software development
- Historically validated on many projects
- Incorporates key influencing factors

Limitations

- Accuracy depends on early KLOC estimates
- Less effective for Agile and DevOps pipelines
- Needs tuning for different organizations
- May overcomplicate small projects