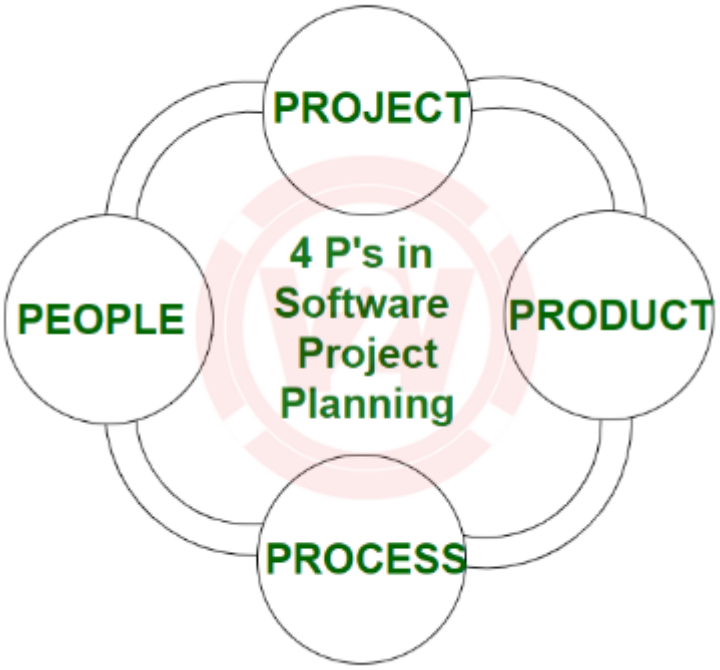
## Unit - III Software Project Management

### 📘 The Management Spectrum – The 4 P's

In software engineering, the **Management Spectrum** refers to the four essential elements that influence the **success or failure** of a software project.It focuses on the 4P's. People, Product, Process and Project.

These are known as the **4 P's**




#### 🔹 1. People

People are the **most important resource** in any software project. This includes project managers, developers, testers, customers, users, and other stakeholders.
Effective **communication, collaboration, motivation, and leadership** play a vital role in managing people.

Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.

People connected directly or indirectly are the dynamic and live factor in project management. Project management has to deal and interact with different levels of people within and outside the organization.

People from within the organization are developers, analysts, designers, technology specialists, HR, managers and senior management.

- **2. Product**

The product is the **software system to be developed**. It includes all functional and non-functional requirements specified by the customer.
The product in the context of software is the scope of the software that is proposed to solve the requirement of the user.

To develop successfully, product objectives and scope should be established, alternative solutions should be considered, and technical and management constraints should be identified.
Without this information, it is impossible to define reasonable (and accurate) estimates of the cost.
Understanding the product clearly is essential for correct estimation of **resources, time, and cost**.

The software and its scope are a product that is defined and described by following four factors:

1.Objective : Objective is beneficial to users to solve business problems.

2.Performance: This stipulates non-functional requirements of speed, quality and features such as ease of use, adaptability, interoperability.

3.Context: The scenario or situation in business which has problems and which need solutions within the given domain.

4.Functions: Software system processes and functions it provides.

- **3. Process**

A process is the **set of activities and tasks** used to develop the software.Process refers to methodologies to be followed for developing the software. It is the framework for establishment of a comprehensive plan and strategies for software development.Choosing an appropriate process model (e.g., Waterfall, Agile, Spiral) ensures that the project is **well-managed, efficient, and structured**.
The process specifies the policies, procedures, tools and techniques to be used for software development.

- **4. Project**

The project includes the **overall management and execution** of work. It involves **planning, monitoring, scheduling, resource allocation**, and **risk management** to ensure timely and successful delivery.

A project is defined as an activity that has a definite start and a definite end require multiple resources and is full uncertainties on a number of counts.

Project management is "the application of knowledge, skills, tools and techniques to manage resources to deliver customer needs within agreed terms of cost time, and scope of delivery".

✅ **Applications of Management Spectrum:**

- Used in **software project planning and execution**
- Helps in **team coordination and communication**
- Guides **selection of suitable development model**
- Aids in **risk management and resource allocation**

---

✅ **Advantages:**

- Ensures **balanced project management**
- Improves **project visibility and control**
- Promotes **team efficiency and productivity**
- Supports **timely and successful delivery**

---

❌ **Disadvantages:**

- Relies heavily on **people skills and experience**
- Mismanagement of any one "P" can affect the whole project
- Can be **complex** for large and dynamic projects
- May need continuous **monitoring and updates**

---

## 📊 3.2 Metrics for Size Estimation

**Size estimation** is an important part of project management. It helps estimate **effort, cost, and duration** of the software project.
Estimating the size of the software to be developed is the very first step to make an effective estimation of the project.
A customer's requirements and system specification forms a baseline for Estimation of the size of software. It helps to determine or estimate the size of a software application of a component.

Two widely used size metrics are:

| Project | LOC | Effort | $(000) | Pp doc | Errors | Defects | People |
|---|---|---|---|---|---|---|---|
| alpha | 12,100 | 24 | 168 | 365 | 134 | 29 | 3 |
| beta | 27,200 | 62 | 440 | 1224 | 321 | 86 | 5 |
| gamma | 20,200 | 43 | 314 | 1050 | 256 | 64 | 6 |

Fig. Size-Oriented Metrics

---

### 🔹 A. Line of Code (LoC)

**Definition:** Line of Code (LoC) is a metric that measures the **number of lines of source code** in a software program. It does not include blank lines and comments.

Counting the number of lines of code in computer programs is the most traditional and widely used software metric.It is also the most controversial. Lines Of Code (LOC) is one of the widely used methods for size estimation.

Lines of code written for assembly language are more than lines of code written in C++ From LOC.

**Advantages:**

- Simple and easy to measure after coding is done
- Useful for measuring programmer productivity
- Helps estimate effort using historical data

**Disadvantages:**

- Language-dependent (LoC varies by programming language)
- Doesn't measure functionality or logic
- May encourage writing more lines instead of efficient code

**Example:** A module with 500 LoC can be used to estimate time and cost using past productivity data.

---

🔹 **B. Function Points (FP)**

**Definition:** Function Point is a **functionality-based size metric** that estimates the size of software by evaluating **user-visible features**, not lines of code.

**Key Components Considered:**

- External Inputs (EI)
- External Outputs (EO)
- External Inquiries (EQ)
- Internal Logical Files (ILF)
- External Interface Files (EIF)

Each component is assigned a weight based on complexity (Low, Medium, High), and then total function points are calculated.

**Advantages:**

- Language-independent
- Can be estimated early in development
- Focuses on business functionality, not implementation

**Disadvantages:**

- Estimation may vary based on evaluator's judgment
- Requires proper training to calculate accurately

**Example:** If a system has 5 inputs, 3 outputs, and 2 files, function points can be calculated using a weighted formula.

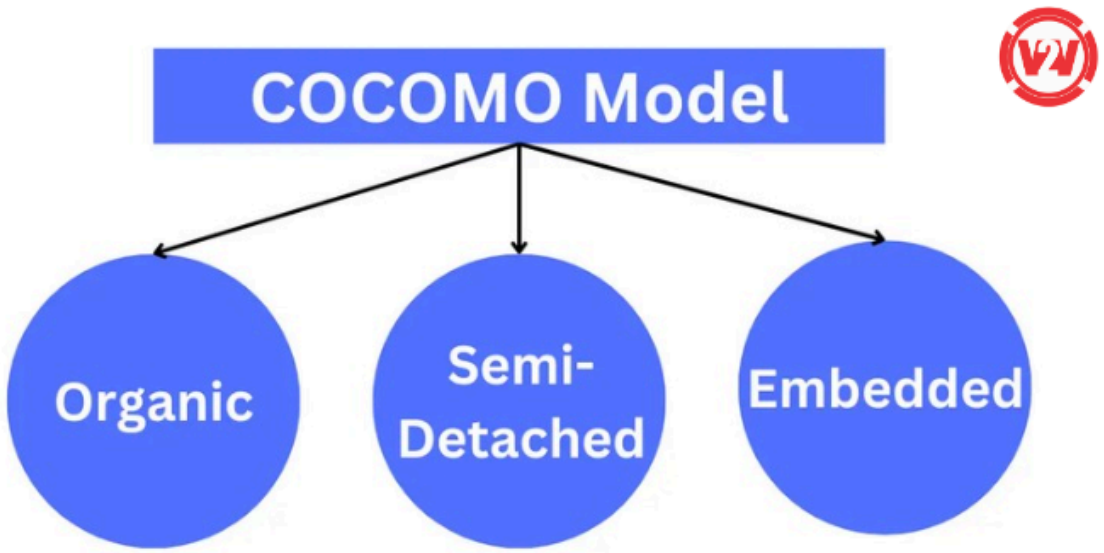
## Difference Between LoC and FP

| Aspect | Lines of Code (LoC) | Function Points (FP) |
|---|---|---|
| 1. Basis | Counts physical lines of code | Measures functionality delivered to the user |
| 2. Language Dependence | Yes, depends on the programming language | No, independent of programming language |
| 3. Timing of Use | Useful **after** coding starts | Useful in the **early stages** of project (before coding) |
| 4. Accuracy | Can be misleading (same logic may use more or fewer lines in different languages) | More consistent and standardized |
| 5. Measurement | Easy to count using tools | Requires functional analysis and understanding of scope |
| 6. Usability | Good for **maintenance projects** | Better for **new project estimation and planning** |

## 📘 Project Cost Estimation using COCOMO & COCOMO II

**COCOMO** (Constructive Cost Model) is a **cost estimation model** used in software engineering to estimate the **effort, cost, and time** required to develop a software project.

Developed by **Barry Boehm** in **1981**, COCOMO is based on historical project data and uses mathematical formulas.




### 🔹 Types of Projects (Modes):

| Mode | Description |
|---|---|
| **Organic** | Small projects, experienced teams, well-defined requirements |
| **Semi-Detached** | Medium-size projects, mixed experience, moderately flexible |
| **Embedded** | Large, complex projects, strict constraints and tight coupling |

### 🔹 Purpose:

To estimate:

- **Effort** (in person-months)
- **Development Time** (in months)

- **Cost** (in labor units or ₹/$)

## 📊 COCOMO Basic Model Constants Table

| Software Project Type | a | b | c | d |
|---|---|---|---|---|
| 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 |

### 🔹 COCOMO I Modes and Formulas

| Mode | Effort Formula | Time Formula |
|---|---|---|
| **Organic** | Effort = 2.4 × (KLoC)^1.05 | Time = 2.5 × (Effort)^0.38 |
| **Semi-Detached** | Effort = 3.0 × (KLoC)^1.12 | Time = 2.5 × (Effort)^0.35 |
| **Embedded** | Effort = 3.6 × (KLoC)^1.20 | Time = 2.5 × (Effort)^0.32 |

📌 Where:

- **KLoC** = Thousands of Lines of Code
- **Effort** = Person-Months
- **Time** = Development Time in Months

### ✅ Solved Example

**Problem Statement:**
Estimate the **Effort** and **Development Time** for a **software project of 50 KLoC** using:

- (a) Organic Mode
- (b) Semi-Detached Mode
- (c) Embedded Mode

---

### 🔸 (a) Organic Mode

**Formulas:**
Effort = 2.4 × (KLoC)^1.05
Time = 2.5 × (Effort)^0.38

**Step 1: Effort**

$$\text{Effort} = 2.4 \times (50)^{1.05}$$

$$50^{1.05} \approx 56.3$$

$$\text{Effort} \approx 2.4 \times 56.3 \approx \boxed{135.12 \text{ person-months}}$$

**Step 2: Time**

$$\text{Time} = 2.5 \times (135.12)^{0.38}$$

$$135.12^{0.38} \approx 6.87$$

$$\text{Time} \approx 2.5 \times 6.87 \approx \boxed{17.18 \text{ months}}$$

### 🔸 (b) Semi-Detached Mode

**Formulas:**
Effort = 3.0 × (KLoC)^1.12
Time = 2.5 × (Effort)^0.35

**Step 1: Effort**

$$\text{Effort} = 3.0 \times (50)^{1.12}$$

$$50^{1.12} \approx 66.7$$

$$\text{Effort} \approx 3.0 \times 66.7 \approx \boxed{200.1 \text{ person-months}}$$

**Step 2: Time**

$$\text{Time} = 2.5 \times (200.1)^{0.35}$$

$$200.1^{0.35} \approx 7.93$$

$$\text{Time} \approx 2.5 \times 7.93 \approx \boxed{19.82 \text{ months}}$$

### 🔸 (c) Embedded Mode

**Formulas:**
Effort = 3.6 × (KLoC)^1.20
Time = 2.5 × (Effort)^0.32

**Step 1: Effort**

$$\text{Effort} = 3.6 \times (50)^{1.20}$$

$$50^{1.20} \approx 84.3$$

$$\text{Effort} \approx 3.6 \times 84.3 \approx \boxed{303.5 \text{ person-months}}$$

**Step 2: Time**

$$\text{Time} = 2.5 \times (303.5)^{0.32}$$

$$303.5^{0.32} \approx 8.97$$

$$\text{Time} \approx 2.5 \times 8.97 \approx \boxed{22.43 \text{ months}}$$

---

## 📘 COCOMO II – Post-Architecture Model

---

🔷 **Definition:** The **Post-Architecture Model** is the **most detailed** and widely used model in the **COCOMO II suite**. It is used after the project's **architecture and major requirements have been finalized**.

It helps estimate:

- **Effort** (in person-months)
- **Schedule (time)**
- **Cost**

It supports modern software practices like:

- **Component-based development**
- **Agile and iterative models**
- **Reuse and Object-Oriented Design**

---

🔷 **When is it used?**

After:

- Requirements are stable
- High-level architecture is defined
- Key project decisions are made (language, platform, team, etc.)

**COCOMO II – Formulas**

### 🔹 1. Effort Estimation (in Person-Months):

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

- **A** = Constant (usually 2.94)
- **Size** = Estimated size of software in **KSLOC** (Thousands of Lines of Code)
- **B** = Exponent calculated from scale factors
- **EAF** = Effort Adjustment Factor (product of 17 cost driver multipliers)

### 🔹 2. Exponent Calculation (B):

$$B = 0.91 + 0.01 \times \sum(\text{Scale Factors})$$

- Scale Factors: Precedentedness, Flexibility, Risk Resolution, Team Cohesion, Process Maturity

### 🔹 3. Development Time Estimation (Optional):

$$\text{Time} = 3.67 \times (\text{Effort})^S$$

- S ≈ 0.28 to 0.35 (based on schedule constraints)

**✅ Solved Example: COCOMO II**

**Estimate the effort for a project with:**

- Size = 50 KSLOC
- ∑(Scale Factors) = 18
- EAF = 1.15
- A = 2.94

### 🔸 Step 1: Calculate Exponent B

$$B = 0.91 + 0.01 \times 18 = 0.91 + 0.18 = \boxed{1.09}$$

---

### 🔸 Step 2: Calculate Effort

$$\text{Effort} = 2.94 \times (50)^{1.09} \times 1.15$$

$$(50)^{1.09} \approx 66.35$$

$$\text{Effort} = 2.94 \times 66.35 \times 1.15 \approx \boxed{224.3 \text{ person-months}}$$

---

### 🔸 Step 3: (Optional) Development Time

Assume S = 0.33:

$$\text{Time} = 3.67 \times (224.3)^{0.33} \approx 3.67 \times 6.47 \approx \boxed{23.75 \text{ months}}$$

---

## 📘 Risk in Software Engineering

🔹 **Definition:**

A **risk** is a potential problem or uncertain event that may negatively affect the success of a software project.
It could cause **delays, increased costs, reduced quality**, or **project failure**.

---

## ✅ Types of Risk in Software Engineering

Risks are generally categorized into the following:

- **1. Project Risks**
  - Risks that affect **project schedule, resources, or budget**
  - Examples:
    - Inadequate staff or skills
    - Unrealistic deadlines
    - Cost overruns

- **2. Technical Risks**
  - Related to the **technology, tools, or platforms** used
  - Examples:
    - New/unfamiliar technology
    - Integration failures
    - System performance issues

- **3. Business Risks**
  - Risks that affect the **business or organization**
  - Examples:
    - Product may not meet customer needs
    - Loss of market value
    - Change in business strategy

- **4. Operational Risks**
  - Risks during **day-to-day development activities**
  - Examples:
    - Poor communication
    - Mismanagement
    - Poor documentation

🔸 **5. External Risks**

- Caused by **factors outside the organization**
- Examples:
  - o Vendor delays
  - o Regulatory changes
  - o Natural disasters


📘 **RMMM Risk Monitoring, Management Strategy – Mitigation, and**

---


🔹 **Definition:**

RMMM stands for:

**Risk Mitigation, Risk Monitoring, and Risk Management**

It is a structured **risk-handling approach** used in software engineering to **identify, reduce, track, and respond** to potential risks in a project.

---

🎯 **Goal of RMMM Strategy**

The **main goal** of the RMMM strategy is:

"To **identify risks early** in the software process, **minimize their impact** through planning, **monitor them continuously**, and **handle them effectively** when they occur."

It ensures:

- Project **continuity**
- Reduced **cost and schedule impact**
- Higher **quality and reliability**

---

✅ **Components of RMMM Strategy**

---

🔸 **1. Risk Mitigation**

- **Goal**: Prevent the risk or reduce its **probability or impact**
- Involves creating **preventive plans**
- **Examples**:
  - Use experienced developers for complex modules
  - Backup plans for third-party tools
  - Allocate extra time in schedule for high-risk tasks

---

🔸 **2. Risk Monitoring**

- **Goal**: Track the risk and **observe warning signs**
- Ongoing activity throughout the project
- Includes regular **status reports**, **risk checklists**, and **review meetings**

---

🔸 **3. Risk Management**

- **Goal**: Take proper **action when the risk actually occurs**
- Execute **contingency plans**

- **Examples**:
  - Reassign tasks if a developer leaves
  - Use alternative tool if preferred tool fails
  - Adjust schedule/resources when delays happen

---

📘 **Project Scheduling in Software Engineering**

**Definition: Project Scheduling** is the process of **planning tasks**, **allocating resources**, and **setting timelines** to ensure a software project is **completed on time and within budget**.

It defines:

- **What tasks need to be done**
- **When they need to be done**
- **Who will do them**
- **How long they will take**

---

### 🎯 Basic Principles of Project Scheduling

1. **Compartmentalization** – Divide the project into smaller manageable tasks or modules.
2. **Interdependency** – Determine relationships between tasks (what depends on what).
3. **Time Allocation** – Estimate the time required for each task.
4. **Effort Validation** – Ensure enough resources are available for each task.
5. **Defined Responsibilities** – Assign responsibilities to team members.
6. **Milestones and Deadlines** – Mark key dates and review points.
7. **Monitoring** – Track progress and update the schedule regularly.

---

### 🔧 Project Scheduling Techniques

---

### 🔷 1. CPM (Critical Path Method)

🔹 **Definition:** CPM is a technique used to find the **longest sequence of dependent tasks** (called the **critical path**) that determines the **minimum project duration**.

🔹 **Key Concepts:**

- Tasks are represented as **nodes** or **activities**
- Each task has a **fixed duration**
- **No uncertainty** in task time estimates

🔹 **CPM Steps:**

1. List all tasks and their durations
2. Identify dependencies between tasks
3. Draw a network diagram
4. Calculate the **critical path**
5. Highlight tasks with **zero slack** (delays in these will delay the project)

✅ **Advantages:**

- Helps identify most important (critical) tasks
- Easy to use for **well-defined projects**
- Optimizes **time and resource allocation**

❌ **Disadvantages:**

- Assumes **fixed time estimates**
- Less effective for **uncertain or risky** projects

---

🔷 **2. PERT (Program Evaluation and Review Technique)**

🔹 **Definition:** PERT is used to handle **uncertainty in project scheduling** by using **probabilistic time estimates**.

🔹 **Key Concepts:**

- Each task has **three time estimates**:
  - **Optimistic time (O)** – best-case
  - **Most likely time (M)** – expected case

- **Pessimistic time (P)** – worst-case

🔹 **PERT Steps:**

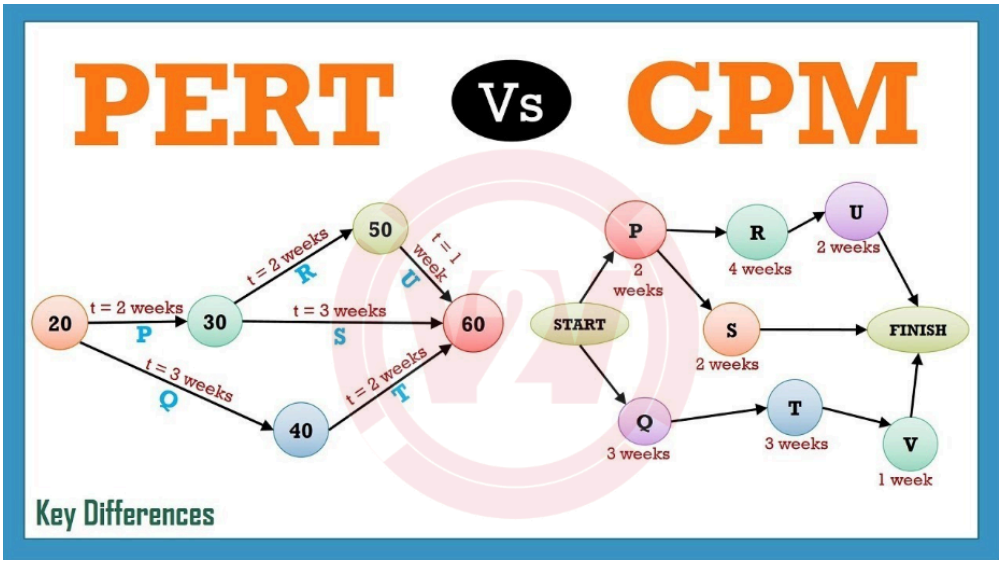
1. List all activities and estimate O, M, P times
2. Calculate Expected Time (TE) for each activity
3. Build the network diagram
4. Find the **critical path** based on TE values

✅ **Advantages:**

- Useful for **complex or uncertain projects**
- Better time estimation using **probabilities**
- Helps in **risk analysis**

❌ **Disadvantages:**

- Requires accurate estimates for O, M, P
- More complex than CPM



## Difference Between PERT and CPM

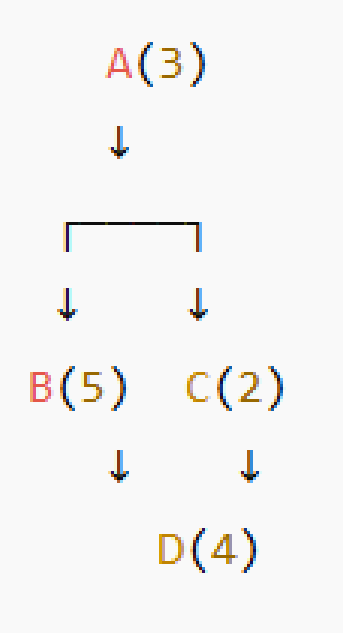
| Aspect | PERT (Program Evaluation and Review Technique) | CPM (Critical Path Method) |
|---|---|---|
| Focus | Time (scheduling under uncertainty) | Time and cost (for predictable projects) |
| Nature | **Probabilistic** – used when task durations are uncertain | **Deterministic** – used when task durations are known |
| Time Estimates | Uses **three estimates:** Optimistic, Most Likely, Pessimistic | Uses **one fixed time** estimate per activity |
| Application | Used in **R&D, new product development** | Used in **construction, engineering, software** |
| Critical Path | Helps in identifying uncertain critical paths | Identifies a fixed critical path |
| Cost Consideration | Does **not focus on cost** directly | Includes **cost analysis** and optimization |

### 📘 1. CPM Example – Critical Path Method

- **Given Tasks:**

| Activity | Duration (days) | Predecessor |
|---|---|---|
| A | 3 | - |
| B | 5 | A |
| C | 2 | A |
| D | 4 | B, C |

🔹 **Step 1: Draw the Network Diagram**




🔹 **Step 2: Identify All Paths & Durations**

1. **Path 1**: A → B → D = 3 + 5 + 4 = **12 days**
2. **Path 2**: A → C → D = 3 + 2 + 4 = **9 days**

---

🔹 **Step 3: Find the Critical Path**

- **Critical Path = A → B → D**
- **Minimum project duration = 12 days**

✅ **Why?** It's the **longest path** with **no slack**.

---

✅ **Final CPM Summary:**

- **Critical Path**: A → B → D
- **Project Duration**: 12 days

📘 **2. PERT Example – Program Evaluation and Review Technique**

🔹 **Given Activity with Time Estimates:**

| Activity | Optimistic (O) | Most Likely (M) | Pessimistic (P) |
|---|---|---|---|
| A | 2 | 4 | 6 |

### 🔹 Step 1: Use PERT Formula to Calculate Expected Time

$$TE = \frac{O + 4M + P}{6}$$

$$TE = \frac{2 + 4(4) + 6}{6} = \frac{2 + 16 + 6}{6} = \frac{24}{6} = \boxed{4 \text{ days}}$$

---

### 🔹 Step 2: Calculate Variance (optional for risk analysis)

$$\text{Variance} = \left(\frac{P - O}{6}\right)^2 = \left(\frac{6 - 2}{6}\right)^2 = \left(\frac{4}{6}\right)^2 = \boxed{0.44}$$

✅ **Final PERT Summary:**

- **Expected Time (TE)**: 4 days
- **Variance**: 0.44 (used for risk and confidence analysis)

### 📘 Project Tracking in Software Engineering

🔷 **Definition: Project tracking** is the process of **monitoring and controlling** the software project's progress to ensure it stays on **schedule, within budget, and meets quality** expectations.


**It involves:**

- Tracking task completion
- Identifying delays
- Updating progress regularly
- Making adjustments if needed

---

🎯 **Purpose of Project Tracking:**

- Compare **planned vs actual** progress
- Identify and fix **bottlenecks**
- Ensure **resource efficiency**
- Keep the team and stakeholders **informed**

---

## 📊 1. Timeline Chart

🔹 **Definition:** A **Timeline Chart** shows the **sequence of events or tasks** in a software project over time. It is a **simple horizontal bar** representation of the schedule.

🔹 **Characteristics:**

- Time is shown on the **horizontal axis**
- Tasks/milestones are placed in sequence
- Good for showing **major events and deadlines**

✅ **Advantages:**

- Easy to understand
- Great for **high-level planning**
- Visualizes the **order of tasks**

❌ **Disadvantages:**

- Does not show task dependencies
- Not suitable for **complex projects**

## ✅ 1. Timeline Chart – Example

📘 **Scenario:**

You are developing a **student management system**.

🔹 **Major Project Events (with Dates):**

| Milestone | Planned Date |
|---|---|
| Requirements Gathering | Day 1 |
| Design Completion | Day 4 |
| Coding Start | Day 6 |
| Testing Start | Day 10 |
| Deployment | Day 14 |

## 📊 Timeline Chart (Text Representation)

```
Day →       1     4     6     10    14
            ↓     ↓     ↓     ↓     ↓
Event →  [Req] [Design] [Code] [Test] [Deploy]
```

—

📊

**2.**

**Gantt Chart**

- 🔹 **Definition:** A **Gantt Chart** is a detailed bar chart that shows:
  - **Start and end dates** of tasks
  - **Duration** of each task
  - **Dependencies** between tasks
  - **Progress tracking** (completion percentage)

  It is the **most widely used tool for project tracking**.

---

- 🔹 **Components of Gantt Chart:**
  - Vertical axis: **Tasks/Activities**
  - Horizontal axis: **Time scale (days/weeks/months)**
  - Bars: Represent the **duration of each task**
  - Arrows/Lines: Show **dependencies** between tasks
  - Progress shading: Shows **% of task completed**

---

✅ **Advantages:**

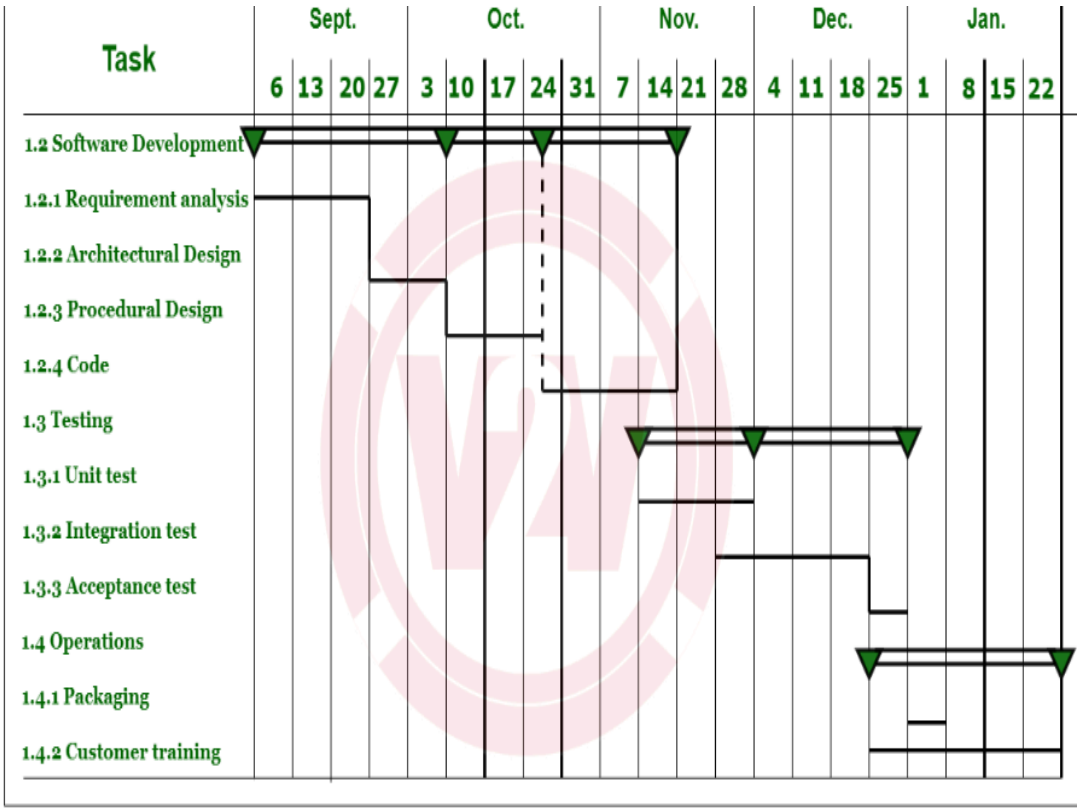
- Clearly shows **project schedule and status**
- Tracks **progress visually**

- Helps with **resource and time management**

❌ **Disadvantages:**

- Can become **complex for large projects**
- Needs to be **updated regularly**

📊 **Gantt Chart (Example)**




Gantt Chart