SE Module 5

Managing Software Projects - Key Concepts

The Management Spectrum:

- **The 4 P's of Project Management**:
    1. **People**: The most crucial element for success. It includes all stakeholders involved, such as managers, developers, clients, and end-users.
    2. **Product**: The software being developed, with a clear scope, functions, and performance criteria.
    3. **Process**: The methodologies, activities, and framework followed to develop the software.
    4. **Project**: The task involving all operations and resources to achieve the goal, which is the successful delivery of the software.

1. People:

- **Stakeholders**: People involved include senior managers, project managers, developers, customers, and end-users.
    - **Team Leader**: Responsible for team execution, quality, pace, and culture. The leader encourages team growth and ensures the environment is conducive to creativity and productivity.
    - **Project Manager**: Oversees the planning, scheduling, budgeting, and execution of the project. Focuses on problem-solving, decision-making, and team-building to ensure smooth operation.
    - **Software Team**: Composed of team members with diverse skills. Key factors to consider when planning the team structure include the problem's complexity, the size of the project, the duration, the required quality, and the team's communication needs.
- **Team Coordination & Communication**:
    - Communication is key to successful project management. This includes formal methods (documents, technical memos, project schedules) and

informal methods (meetings, emails, casual conversations). These communication channels keep the team aligned, share updates, and solve problems quickly.

2. The Product:

- **Software Scope**: This defines the boundaries of the project. Key questions to answer include how the software fits into a larger system, what data it processes, and what performance characteristics are expected.
- **Problem Decomposition**: Breaking down complex tasks into smaller, manageable pieces makes it easier to analyze, design, and solve individual parts of the problem. This step is critical for efficient and effective software development.

3. The Process:

- Framework activities such as communication, planning, modeling, construction, and deployment guide the software development process.
- The process model selected should fit the specific product, customer needs, and project environment. It determines how tasks will be organized and managed, and helps align the work with the project's goals.

4. The Project:

- A project is a collection of tasks with the goal of delivering a software product. Common issues that cause project delays or failure include:
  - Poor understanding of customer requirements.
  - Undefined or poorly defined product scope.
  - Mismanagement of changes or requirements.
  - Unrealistic deadlines or shifting technology.
  - Lack of skilled team members or leadership.
- **Common-sense Approach** to Managing Projects:
  - **Start with a clear understanding of the problem**: Thoroughly analyze and define project requirements.

- **Maintain momentum**: Ensure the team stays motivated, focused, and minimize personnel turnover.
- **Track progress**: Use work products, technical reviews, and quality assurance to track development milestones and ensure high standards.
- **Make smart decisions**: Avoid unnecessary complexity and keep the focus on delivering quality work.
- **Postmortem analysis**: After project completion, conduct an analysis to identify lessons learned and areas of improvement.

Critical Practices in Project Management:

- **Risk Management**: Identify potential risks early and develop strategies to mitigate them.
- **Cost and Schedule Estimation**: Accurately estimate the resources and time required to complete the project.
- **Metrics-Based Project Management**: Measure project progress and quality using key metrics to stay on track.
- **Earned Value Tracking**: Monitor the project's progress by comparing the planned value with the actual progress.
- **Defect Tracking**: Identify and track software defects to ensure the quality of the final product.
- **People-Aware Management**: Manage and motivate the team effectively, ensuring they have the right tools, resources, and support to succeed.

By focusing on these elements and practices, the software project can achieve its objectives, delivering a product that meets user requirements on time and within budget. Effective communication, careful planning, and proactive risk management are key to successful software project delivery.

Estimation for Software Projects - Key Concepts

Introduction:

- **Estimation** is the process of calculating the resources, time, and costs required for a software project.
- **Project planning** starts with estimation, where the software team must estimate the work, resources, and time required for the project's completion.
- The goal of planning is to create a strategy to control, track, and monitor the project.

Software Project Planning:

- **Project Planning** is the longest and most crucial phase of the project lifecycle. Proper planning ensures the project's success.
- **Two task sets** to accomplish during planning:
  - **Task Set-I**: Establish project scope, determine feasibility, and analyze risks.
  - **Task Set-II**: Estimate cost and effort, develop a project schedule, and define required resources.

Task Set-I:

- **Software Scope**: Defines what functions, features, data input/output, and end-user requirements the project will cover.
- **Estimation of Resources**:
  - **Human Resources**: The required number of developers and other team members.
  - **Reusable Software Resources**: Components like off-the-shelf or partially developed software can be used.
  - **Environmental Resources**: Hardware, software tools, and other resources needed to complete the project.

Software Project Estimation:

- Software estimation is not an exact science because various factors like human, technical, and environmental conditions can impact the cost and effort.
- **Key requirements for accurate estimation**:

    - **Experience**: Knowledge of similar projects.
    - **Access to historical data**: Metrics from past projects.
    - **Courage to make predictions**: Sometimes only qualitative data is available.
- **Important Considerations**:
    - Understanding project scope is critical.
    - Decomposition of the project into manageable parts is necessary.
    - Use at least two estimation techniques to improve accuracy.
    - There will always be some degree of uncertainty in estimation.

Project Estimation Process:

1. **Estimate Software Size**: Begin by estimating the size of the software to be developed.
2. **Estimate Effort**: Based on the size, calculate the effort in man-months.
3. **Estimate Schedule**: From the effort estimate, derive the project schedule.
4. **Estimate Costs**: Apart from effort and size, consider costs like hardware, travel, telecommunication, and training.

Project Estimation Responsibility:

- Software managers, engineers, and estimators are responsible for providing accurate project estimates.

Estimation Techniques:

1. **Past Project Experience**: Use data and experience from similar past projects.
2. **Conventional Estimation Techniques**: Techniques like expert judgment or analogical estimation.
3. **Empirical Models**: Use statistical models like COCOMO (Constructive Cost Model) for predictions.
4. **Automated Tools**: Software tools can help automate the estimation process.

Estimation Accuracy:

- **Accuracy** measures how close the estimate is to the actual outcome. Several factors affect accuracy:
    1. **Accuracy of Input Data**: The quality of data used for estimation.
    2. **Calculation Method**: How the estimation is calculated and whether the method is reliable.
    3. **Historical Data**: The relevance of previous data used to match the current project.
    4. **Process Predictability**: The stability and predictability of the software development process.
    5. **Stability of Requirements**: Changes in product requirements can affect the accuracy of estimates.
    6. **Planning and Monitoring**: Accurate planning, monitoring, and control can help avoid surprises that lead to delays or incorrect estimates.

By understanding these steps and techniques, software teams can better estimate the cost, time, and resources required for successful project delivery. However, it's important to recognize the inherent risks and uncertainties involved in estimation.

Software Project Estimation Techniques - Overview

Software Project Estimation Techniques are essential for predicting the effort, time, and cost of software development. Several methods help project managers make accurate estimations, based on historical data, decomposition strategies, or empirical models.

Key Estimation Techniques:

1. **Experience (Historical Data)**
2. **The Decomposition Technique**
3. **The Empirical Estimation Model**

---

1. Experience-Based Estimation (Historical Data):

- **Experience-based estimation** uses data from previous projects to make predictions for new projects. It considers past project details such as effort, cost, and duration, and applies them to the current project.
- In this method, estimates are formed as functions of certain variables (e.g., Lines of Code (LOC) or Function Points (FP)) that predict the required resources.
- Formula:
  - Estimated Value (d) = Function of variables (v_1, v_2,...v_n) from previous similar projects.

---

2. The Decomposition Approach:

- The **Decomposition Technique** involves breaking down a software project into smaller, manageable components, each with its own estimation for size, cost, and effort.
- **Steps** in the decomposition approach:
  - **Software Sizing**: Estimate the size of the project (direct or indirect approach).
    - **Direct Approach**: Using **Lines of Code (LOC)**.
    - **Indirect Approach**: Using **Function Points (FP)**.

2.1 Software Sizing:

- The project size affects the effort required. Sizing is a quantifiable measure used to estimate the work involved in a project.
- **Sizing Approaches**:
  1. **Fuzzy Logic Sizing**: Uses approximate reasoning techniques for vague or incomplete data.
  2. **Function Point Sizing**: Focuses on inputs, outputs, files, and user interactions to determine effort.
  3. **Standard Component Sizing**: Breaks the software into components like modules, screens, or reports, measuring them by predefined standards.

4. **Change Sizing**: Estimation based on modifications to existing software.

2.2 Problem-Based Estimation:

- **LOC and FP** are used to estimate the size of software elements. Both techniques involve:
  1. Estimating the size of software components (functions, features).
  2. Comparing historical data to develop cost and effort predictions.

  Key Points:

  - **Three-Point Estimation**: An estimate that considers three values:
    - Optimistic (S_{opt})
    - Most likely (S_m)
    - Pessimistic (S_{pess})
  - The **Expected Value (EV)** is calculated using a weighted average of these three values.

  Formula:

  - $EV = (S_{opt} + 4*S_m + S_{pess}) / 6$.

2.3 Lines of Code (LOC) Estimation:

- **LOC** is one of the simplest and oldest methods for estimating software size. It calculates the total lines of code needed for the software and provides an estimation of effort.
- **Example**:
  - For the **Library Management System** case, the expected LOC for different functions like user interface, database management, and report generation is estimated, and the total LOC is calculated.

  Estimated Cost:

  - Cost per LOC = Burdened labor rate / Productivity.

- **Cost Calculation**: Using the estimated LOC and historical productivity data, project costs and efforts are derived.

Example:

- **Total LOC** = 10,900.
- **Cost per LOC** = $12.
- **Total Cost** = 10,900 * $12 = $130,800.
- **Total Effort** = 21.8 Person-months.

Problems with LOC:

- Different programming languages lead to varying LOC counts.
- It only measures the volume, not the complexity of the code.

---

3. Function Point (FP)-Based Estimation:

- **Function Points** measure the software's functionality, focusing on data and interactions rather than lines of code.
- **FP Calculation**:

Example:

- Example: For a system with 320 FP, applying an adjustment factor (e.g., 1.17) results in 375 FP.
- **Cost per FP**: Based on the labor rate and historical productivity.
- **Productivity** = 6.5 FP/person-month.
- **Cost per FP** = $1230.
- **Total Cost**: $461,000.
- **Estimated Effort**: 58 person-months.

---

4. Process-Based Estimation:

- This approach estimates effort based on the software development process to be followed.

- The project's process is broken down into smaller tasks, and the effort for each task is estimated.
- This is similar to **Problem-Based Estimation** but focuses on the tasks in the process rather than the specific software components.

---

5. Estimating with Use Case Points (UCP):

- **Use-Case Points** (UCP) are used to measure software size based on **use cases**.
  - A use case describes a sequence of interactions between the user and the system to achieve a specific goal.

Steps for UCP Calculation:

1. **Unadjusted Use Case Weight (UUCW)**: Based on the complexity of the use cases.
2. **Unadjusted Actor Weight (UAW)**: Based on the complexity of the actors involved.
3. **Technical Complexity Factor (TCF)**: Adjusts for technical aspects of the system.
4. **Environmental Complexity Factor (ECF)**: Adjusts for environmental factors, like the team's experience.

Formula:

- **UCP** = (UUCW + UAW) × TCF × ECF.

Example:

- For a project with 1 simple use case, 1 average use case, and 3 complex use cases, the UUCW is 60.
- If the technical and environmental factors adjust to 1.09 and 1.1, the final **UCP** is 80.33.

Use Case-Based LOC Calculation:

- Use the average **LOC per use case** and adjust it based on project specifics.

Challenges with Use Cases:

- Use cases are often described in varying formats, and their complexity can be difficult to quantify.
- They focus on the external view (user's perspective), which may not address all internal complexities.

Conclusion:

Each estimation technique has its advantages and challenges. By selecting the appropriate method based on the project's specifics—whether through historical data, decomposition, or use cases—software managers can make informed predictions about the project's cost, effort, and schedule.

Sample Problem of Function Point (FP)-based estimation:

You are tasked with estimating a software project using **Function Points (FP)**. Here is the information you have:

1. **Function Points** Calculation:
    - **External Inputs (EI)**: 8
    - **External Outputs (EO)**: 6
    - **User Inquiries (EQ)**: 4
    - **Internal Logical Files (ILF)**: 5
    - **External Interface Files (EIF)**: 2
2. The **Value Adjustment Factor (VAF)** is given as 1.17.
3. The historical productivity for similar projects is **6.5 Function Points per person-month (FP/pm)**.
4. The **burdened labor rate** is $8,000 per person-month.

Steps for FP Estimation:

1. **Calculate the Unadjusted Function Point Count (UFP):**
   Function Points are calculated based on the complexity of various components, and each component is assigned a weight. Here's the weighting scheme:

Component	Low Complexity	Average Complexity	High Complexity
External Inputs (EI)	4	5	7
External Outputs (EO)	5	7	10
User Inquiries (EQ)	4	5	7
Internal Logical Files (ILF)	7	10	15
External Interface Files (EIF)	5	7	10

   Based on the sample question:

   - **External Inputs (EI)**: 8 (Assumed to be average complexity, so weight = 5)
   - **External Outputs (EO)**: 6 (Assumed to be average complexity, so weight = 7)
   - **User Inquiries (EQ)**: 4 (Assumed to be average complexity, so weight = 5)
   - **Internal Logical Files (ILF)**: 5 (Assumed to be average complexity, so weight = 10)
   - **External Interface Files (EIF)**: 2 (Assumed to be average complexity, so weight = 7)

   Now, calculate the **Unadjusted Function Points (UFP)**:

   $$\text{UFP} = (\text{EI} \times \text{Weight for EI}) + (\text{EO} \times \text{Weight for EO}) + (\text{EQ} \times \text{Weight for EQ}) + (\text{ILF} \times \text{Weight for ILF}) + (\text{EIF} \times \text{Weight for EIF})$$

   Substituting values:

   $$\text{UFP} = (8 \times 5) + (6 \times 7) + (4 \times 5) + (5 \times 10) + (2 \times 7)$$

   $$\text{UFP} = 40 + 42 + 20 + 50 + 14 = 166$$

So, the **Unadjusted Function Point (UFP)** is **166**.

2. **Apply the Value Adjustment Factor (VAF)**:
   The VAF adjusts the UFP based on various technical and environmental factors. The formula for
   Function Points (FP) is:

   $$FP = UFP \times VAF$$

   Given the **VAF** is **1.17**, we can calculate:

   $$FP = 166 \times 1.17 = 194.22$$

   So, the **adjusted Function Points (FP)** are **194.22**.

3. **Calculate the Effort in Person-Months (PM)**:
   Now, we need to calculate the effort required for the project based on historical productivity, which is
   6.5 FP/pm. The formula for effort in person-months is:

   $$\text{Effort (PM)} = \frac{\text{FP}}{\text{Productivity (FP/pm)}}$$

   Substituting values:

   $$\text{Effort (PM)} = \frac{194.22}{6.5} = 29.87 \text{ person-months}$$

   So, the estimated effort is approximately **29.87 person-months**.

4. **Calculate the Cost**:
   To calculate the total project cost, we need to use the
   burdened labor rate of **$8,000 per person-month**. The formula for cost is:

   $$\text{Cost} = \text{Effort (PM)} \times \text{Labor Rate (per PM)}$$

   Substituting values:

   $$\text{Cost} = 29.87 \times 8,000 = 239,000$$

   So, the estimated **project cost** is **$239,000**.

Summary of Results:

- **Unadjusted Function Points (UFP)** = 166

- **Adjusted Function Points (FP)** = 194.22
- **Effort** = 29.87 person-months
- **Cost** = $239,000

This estimation gives you an idea of the resources and cost required to complete the project using **Function Point-based estimation**.

Empirical Estimation Models in Software Engineering
Empirical estimation models help software engineers predict the effort, cost, and time required for software development based on historical data and project metrics. These models typically rely on formulas derived from regression analysis of data from past software projects. Some well-known empirical models include **COCOMO** (Constructive Cost Model), **Putnam's Software Equation**, and **COCOMO II**, among others.
1. COCOMO Model

The COCOMO model, proposed by Barry Boehm in 1970, predicts effort and time based on the number of Lines of Code (LOC) required for the project. It consists of different levels:

- **Basic COCOMO:** Uses a simple formula to calculate effort and time based on the estimated size of the project in terms of KLOC (thousands of lines of code).
- **Intermediate COCOMO:** Includes additional cost drivers that reflect the attributes of the product, hardware, personnel, and project.
- **Detailed COCOMO:** Further refines estimates by considering the impact of cost drivers on each phase of the software development lifecycle.

COCOMO Model Formulas:

- **Effort (E):**
- **Development Time (T):**
-

Personnel Required:
Where:
• (A, B, C, D) are empirically derived constants for different project types (organic, semi-detached, or embedded).
•
Organic projects: Simple projects with small teams and well-understood problems.
•
Semi-detached projects: Projects with moderate complexity.
•
Embedded projects: Complex projects requiring highly experienced teams.
Example Calculation with Basic COCOMO

Given a project with 400 KLOC, let's calculate the effort and development time for **Organic, Semi-detached, and Embedded modes**.
•
Organic Mode:
  ◦ Effort (E) = (2.4 \times (400)^{1.05} = 1295.31) person-months
  ◦ Time (T) = (2.5 \times (1295.31)^{0.38} = 38.07) months
  ◦ Average Staff Size = (\frac{1295.31}{38.07} = 34) people
•
Semi-detached Mode:
  ◦ Effort (E) = (3.0 \times (400)^{1.12} = 1295.31) person-months
  ◦ Time (T) = (2.5 \times (12462.79)^{0.35} = 38.45) months
•
Embedded Mode:
  ◦ Effort (E) = (3.6 \times (400)^{1.2} = 4772.81) person-months
  ◦ Time (T) = (2.5 \times (4772.81)^{0.32} = 38) months

Example Calculation with Intermediate COCOMO

For the intermediate COCOMO model, cost drivers such as software reliability, personnel capability, and hardware constraints are also considered. Here's a simplified example based on the intermediate model:
•
Formula for Effort:

Where EAF is the Effort Adjustment Factor (calculated based on 15 cost drivers).

2. The Software Equation (Putnam's Model)

The
Software Equation is used for predicting project effort and duration over time, assuming a specific distribution of effort across the software development life cycle. The general formula is:
Where:

- **E** = Total effort (person-months)
- **t** = Project duration (in months or years)
- **P** = Productivity parameter (specific to the project type)
- **B** = Special skills factor (depends on the complexity of the project)

3. COCOMO II Model

COCOMO II is an updated version of the original COCOMO model, which addresses modern software development techniques, including the use of object-oriented programming and component-based development. It uses the same basic formulas but includes factors like the use of tools, software reuse, and team experience.

- **COCOMO II Stages:**
  - **Application Composition Model:** Used in the early stages of development.
  - **Early Design Stage Model:** Used once requirements are stabilized.
  - **Post-Architecture Model:** Used during software construction.

For example, in the
Application Composition Model, object points (similar to function points) are used to estimate effort based on the number of screens, reports, and components in the system.

Application: COCOMO II Object Point Example

Let’s say you have a project with:
• 4 screens (each with 4 views),
• 2 reports (each with 6 sections),
• 10% reuse of object points.

Step-by-Step Calculation:

1.
Object Point Calculation:

2.
New Object Points (NOP) considering 10% reuse:

3.
Effort Calculation (with a productivity rate of 7):

Key Takeaways:

•
COCOMO is the most widely used empirical model for effort estimation.
• The
Basic COCOMO model offers a quick estimate based on project size (KLOC).
• The
Intermediate COCOMO model incorporates additional factors (cost drivers).
• The
Detailed COCOMO model refines estimates by considering the impact of cost drivers on each step of development.
•
COCOMO II updates COCOMO with object points and accounts for modern software development practices.
These models allow software developers and project managers to estimate project size, effort, cost, and time more accurately, but they should be calibrated based on local conditions and the specific characteristics of each project.

Software Project Scheduling Overview

Software project scheduling involves planning and assigning tasks in a structured way to ensure that a software project is completed on time and within budget. It includes defining deliverables, milestones, work breakdown structure, resource allocation, and the sequence of activities.

Key Concepts in Software Project Scheduling

1. **Project-Task Scheduling**:
    - **Project-task scheduling** is a significant project-planning activity, where decisions are made about when various functions or tasks should be performed during the project.
2. **Scheduling Principles**:
    - **Compartmentalization**: The project is broken down into manageable activities and tasks.
    - **Interdependency**: Tasks are organized based on their dependencies (parallel vs. serial).
    - **Time Allocation**: Each task is given specific start and end times considering dependencies.
    - **Effort Validation**: Ensure sufficient resources are allocated for tasks.
    - **Defined Responsibilities**: Each task is assigned to a specific team member.
    - **Defined Outcomes**: Every task must result in a specific outcome (work product or deliverable).
    - **Defined Milestones**: Milestones mark the completion of significant work products or quality reviews.

Steps in Software Project Scheduling

1. **List the Deliverables**:

- Examples: Documents, function demonstrations, subsystem demonstrations, accuracy, reliability, security, or speed.

2. **Define the Milestones**:
   - Milestones mark the completion of deliverables and are measurable.
3. **Work Breakdown Structure (WBS)**:
   - Break down the project into phases, steps, and activities for more effective planning.

Project Effort Distribution

Generally accepted guidelines for effort allocation:

- 2-3% for planning.
- 10-25% for requirements analysis.
- 20-25% for design.
- 15-20% for coding.
- 30-40% for testing and debugging.

The Relationship Between People and Effort

- **Putnam Norden Rayleigh (PNR) Curve**:
  - Illustrates the relationship between effort and project delivery time.
  - Compression of delivery time (moving left on the curve) can raise costs and resources sharply.
  - The curve indicates that delivering a project too quickly may not be feasible without significant cost increases.

Task Networks

- A **task network** (or **activity network**) visually represents the sequence of tasks and their interdependencies.

- It helps visualize how tasks relate to each other and which tasks can run concurrently or must be completed sequentially.

Scheduling Methods

1. **PERT (Program Evaluation and Review Technique)**:
    - Used when task durations are uncertain.
    - Focuses on the best-case, worst-case, and most likely times for tasks.
    - PERT uses three time estimates:
        - **Optimistic**: The best-case scenario.
        - **Pessimistic**: The worst-case scenario.
        - **Most likely**: The realistic expected duration.
    - PERT helps plan when there is uncertainty in task durations.
2. **CPM (Critical Path Method)**:
    - Used when task durations are more predictable.
    - Focuses on identifying the **critical path**, the series of tasks that dictate the project's duration.
    - Delays in any critical path task will delay the entire project.
    - Helps prioritize tasks to ensure the project stays on track.

Key Similarities Between PERT and CPM

- **Estimate Effort**: Both methods help estimate how much work is involved in each task.
- **Task Breakdown**: Both break down the project into smaller tasks.
- **Critical Path**: Both help identify the critical path, ensuring that project deadlines are met.
- **Time Estimates**: Both methods allow for estimating the "most likely" times for each task.

Summary:

- **PERT** is useful when task durations are uncertain and provides estimates based on multiple possibilities (optimistic, pessimistic, most likely).
- **CPM** is more appropriate when task durations are predictable and focuses on identifying the critical path to avoid delays.

Time-Line Charts

- **Gantt Charts** are used to visually represent the project timeline. They help in tracking what tasks are being worked on at any point in time.

Earned Value Analysis (EVA)

EVA is a technique to monitor and control project performance. It compares actual progress with the planned schedule and budget.

Steps to calculate EVA:

1. **Percent Completion**: Determine the percent of completion for each task.
2. **Planned Value (PV)**: The budgeted cost for work scheduled to be done.
3. **Earned Value (EV)**: The value of work actually completed.
4. **Actual Cost (AC)**: The actual cost incurred to complete the work.
5. **Schedule Variance (SV)**: SV = EV - PV. A negative SV means the project is behind schedule.
6. **Cost Variance (CV)**: CV = EV - AC. A negative CV means the project is over budget.
7. **Other Status Indicators**:
    - **Schedule Performance Index (SPI)**: SPI = EV / PV. A value less than 1 means the project is behind schedule.
    - **Cost Performance Index (CPI)**: CPI = EV / AC. A value less than 1 means the project is over budget.
    - **Critical Ratio (CR)**: CR = SPI * CPI.

Conclusion

Software project scheduling is an essential part of project management. By following proper principles, using methods like PERT and CPM, and applying tools such as Gantt charts and Earned Value Analysis, project managers can plan, monitor, and control software development projects more effectively. This ensures timely delivery, cost management, and quality control throughout the project lifecycle.