Software Project Management (SPM)

Course Code: CACS407
Year/ Semester: IV/VII

Compiled by Shishir Ghimire

Credit Hours: 3hrs

Unit - 03: Software Estimation Techniques

Class Load : 7 hrs

TABLE OF CONTENTS

Unit -3

Software Estimation Techniques **7 Hrs**

Software Effort Estimation: Problems with over and under estimations, Basis of software Estimating, Software effort estimation techniques, expert Judgment, Estimating by analogy. Bottoms-up estimating, Top-down approach and parametric models.

Introduction:

Objectives:

OBJECTIVES

When you have completed this chapter you will be able to:

- avoid the dangers of unrealistic estimates;
- understand the range of estimating methods that can be used;
- estimate projects using a bottom-up approach;
- estimate the effort needed to implement software using a procedural programming language;
- count the function points for a system;
- understand the COCOMO II approach to developing effort models.

Software Effort Estimation

3.1

Software Effort Estimation:

Software effort estimation is the process of **predicting the amount of effort** (typically measured in person-hours, person-days, or person-months) required to develop or maintain a software project.

- It helps in planning resources, setting timelines, and budgeting for the project.
- **Importances:**
  - **Project Planning:** Ensures timelines and resources are well-organized.
  - **Cost Management:** Provides an accurate budget for the project.
  - **Risk Reduction:** Minimizes the chances of **overestimating or underestimating** the required effort.
  - **Resource Allocation:** Helps in effectively assigning tasks to team members.

Where are estimates done?

- **Strategic Planning:** Prioritize projects, allocate resources, and plan recruitment based on cost-benefit analysis.
- **Feasibility Study:** Confirm project benefits justify costs.
- **System Specification:** Estimate implementation effort to validate feasibility and evaluate design options.
- **Supplier Evaluation:** Compare contractor bids with internal estimates to ensure accuracy and feasibility.
- **Project Planning:** Refine estimates for smaller tasks to improve resource allocation and detailed planning.

Where are estimates done?

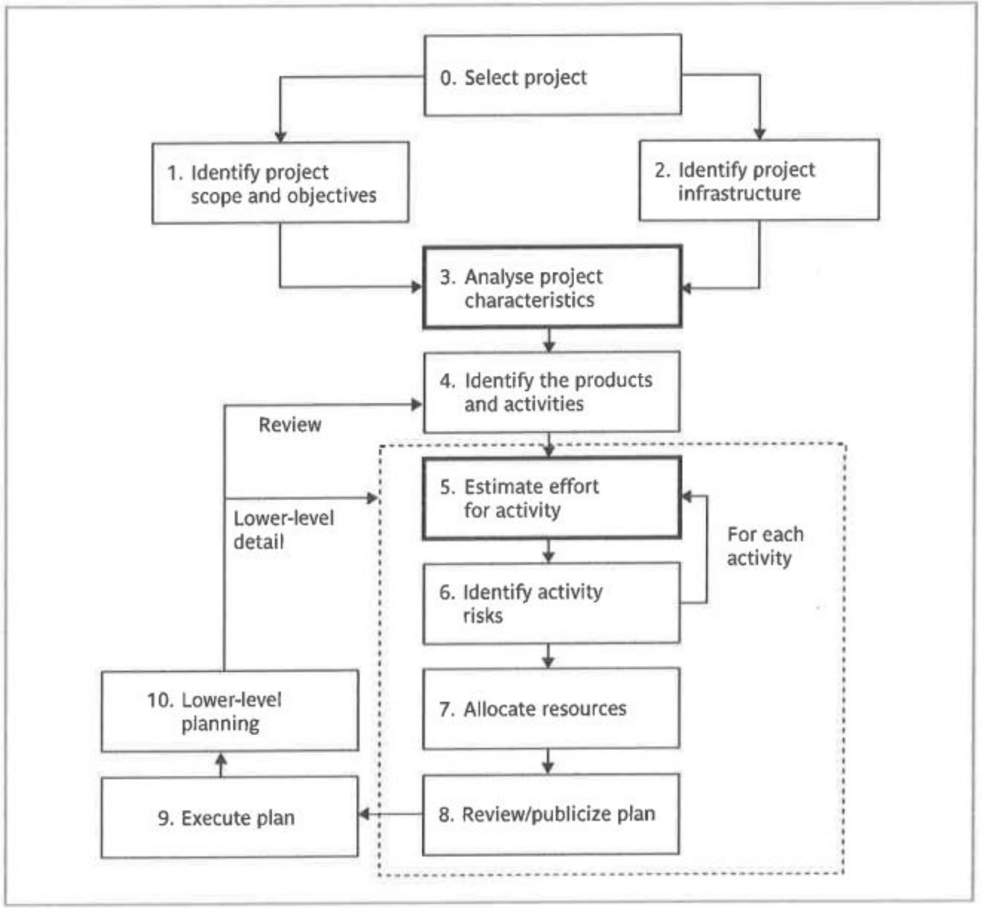
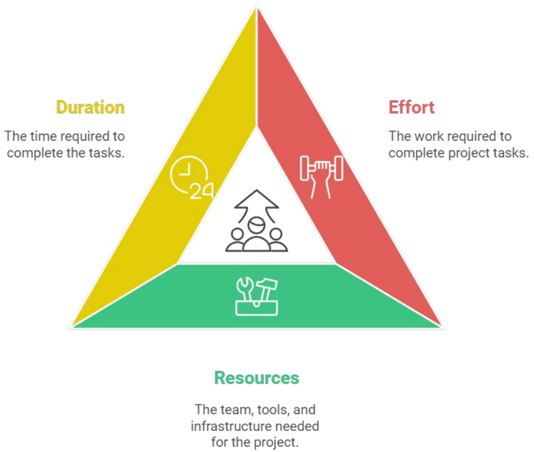


FIGURE 5.1 Software estimation takes place in Steps 3 and 5 in particular

Basic Factors of Estimating:

- **Effort (Cost):**
  - The amount of work required to complete the project tasks.
- **Resources:**
  - The team members, tools, and infrastructure needed.
- **Duration (Time):**
  - The time it will take to complete the tasks.



Other Factors to Consider for Estimating:

- **Project size and complexity:** Bigger and more intricate projects naturally take more effort.
- **Development Team's Experience and Skills:** A seasoned team might need less time than a less experienced one.
- **Technologies and tools used:** Some technologies require more specialized knowledge or have known efficiency levels.
- **Dependencies and external factors:** External factors like communication needs or regulatory compliance can impact effort.
- **Risks and Uncertainties:** Identify potential risks (technical, resource-related, or market-driven) and account for mitigation efforts.

Problems with Over and Under Estimates

3.2

Problems with Over-Estimates:

- **Wasted Time:**
  - **Parkinson's Law:** Work expands to fill the time available. Teams may slow down if the deadline is generous.
- **Increased Management Effort:**
  - **Brooks' Law:** Adding more staff than necessary increases coordination, communication, and management overheads, which can lead to inefficiencies.
- **Longer Completion Times:**
  - Generous estimates may cause projects to stretch unnecessarily, **delaying delivery**.
- **Increased Costs:**
  - Allocating unnecessary resources (e.g., more staff, tools, or facilities) can inflate project costs **without any added value**.

Problems with Over-Estimates:

- **Dilution of Focus:**
  - Excessive time allowances may lead to teams focusing on less critical tasks, diverting attention from core objectives.
- **Scope Creep:**
  - Extra time may lead to unnecessary feature additions, increasing complexity and risks.
- **Inefficient Use of Resources:**
  - Overstaffing leads to idle resources and high management overheads, as highlighted by **Brooks' Law:** Adding more people to a late job makes it later.
- **Loss of Competitive Advantage:**
  - Delayed project delivery due to over-estimation may allow competitors to launch their solutions first.

Problems with Under-Estimates:

- **Quality Risks:**
  - Tight deadlines may lead to substandard work, especially for less experienced staff. **Weinberg's Zeroth Law** warns that sacrificing reliability for speed can compromise the final product.
- **Hidden Defects:**
  - Substandard work might only become evident in later stages, such as testing, requiring rework and causing delays.
- **Missed Deadlines:**
  - The project may fail to meet its timeline or budget, creating dissatisfaction and reduced credibility.
- **Burnout and Low Morale:**
  - Unrealistic deadlines can overwhelm team members, leading to stress, decreased productivity, and potentially high attrition rates.

Problems with Under-Estimates:

- **Rework Costs:**
  - Inadequate planning increases the likelihood of defects being identified in later phases (e.g., testing), requiring costly rework and delaying project delivery.
- **Credibility Issues:**
  - Repeated under-estimation can harm the organization’s reputation, as stakeholders may lose trust in its ability to deliver.
- **Lack of Proper Risk Management:**
  - Insufficient estimates might ignore potential risks, leaving the project vulnerable to unexpected delays and issues.

Balancing Estimates:

- Both over- and under-estimates **have severe consequences**.
  - **Accurate estimates, combined with regular reviews**, can help ensure optimal resource allocation, high-quality outputs, and timely project completion.
  - **Regularly revisiting and refining estimates** during the project lifecycle can help balance resource allocation, maintain quality, and ensure timely delivery.
  - Adopting techniques like **expert judgment, analogous estimation, and progressive elaboration** can further enhance estimation accuracy.

Basis of Software Estimating

3.3

Basis of Software Estimating:

- Software Estimating in project management joins on **understanding** and **predicting** the effort required to complete the project successfully.
- The **several key aspects** to be considered while estimating are:
  - **Time:** How long it will take to develop the software based on the planned functionalities and functions?
  - **Resources:** What personal tool and infrastructure are needed to complete the project within the estimated timeframe?
  - **Cost:** What is the financial implication of the estimated time and resource utilization?

Basis of Software Estimating:

- Effective software estimation relies on key factors and data-driven approaches, including **historical data and size measurement metrics.**
- **It includes**
  a. Historical Data
  b. Challenges in Direct Cost or Time Calculations
  c. Measure of Work

Importance of Historical Data:

- **Role in Estimation:**
  - Historical data from past projects **provides a reference** for estimating costs and effort in new projects.
- **Challenges in Application:**
  - Differences in **programming languages, staff expertise, and project scope** may limit the relevance of historical data.
- **External Data Sources:**
  - In the absence of internal data, organizations can use external databases like the **International Software Benchmarking Standards Group (ISBSG)**, which includes data from over 4,800 projects.

Challenges in Direct Cost or Time Calculations:

- **Uncertainty in Early Planning:**
  - Early in the **planning phase**, **factors like** the individual capabilities of staff, their experience, and the selection of specific technologies are **often undefined.**
  - These uncertainties make **direct calculations of costs or time difficult**.

Measure of Work:

- **Source Lines of Code (SLOC):**
  - Work size is frequently measured using **Source Lines of Code (SLOC)** or **KLOC (thousands of lines of code)**.
- **Limitations of SLOC:**
  - Estimating SLOC accurately is challenging, especially in projects using **parameter-driven application builders.**
  - SLOC **doesn't capture** the complexity of the code being developed, making it an **incomplete** measure.
- **Alternative Measures – Function Points:**
  - Function points **offer a more practical** measure by focusing on **system functionality** rather than raw code quantity.
  - They consider the complexity and breadth of user requirements, providing a **more balanced estimation metric.**

Software Effort Estimation Techniques

3.4

Software Effort Estimation Techniques:

Barry Boehm, a pioneer in software effort modeling, **identified several approaches** to estimate the effort required for software development.

These techniques **range from** mathematical models to expert judgments and practical methods.

These can be explained below as:

- **Algorithmic Models**
  - Uses the **characteristics of the target system** and its **environment** to predict effort.
  - **Example:** COCOMO (Constructive Cost Model).

Software Effort Estimation Techniques:

- **Expert Judgment**
  - Relies on the advice and experience of **knowledgeable** team members or experts to estimate effort.
- **Analogous Estimation**
  - **Compares** the current project with a similar, previously completed project and uses the actual effort from the past project **as the basis** for estimation.
- **Parkinson's Law**
  - Effort available **dictates** the estimate.
  - Not a true prediction method, but a scope-setting approach (e.g., work expands to fill the time available).

Software Effort Estimation Techniques:

- **Price-to-Win**
  - Effort estimate is set at a **value low** enough to secure a contract.
  - It is **more of a business strategy** than a predictive method but can inform scope and priorities.
- **Top-Down Estimation**
  - Begins with an overall project effort estimate, which is **then broken down** into estimates for individual components.
  - Useful for quick initial estimates.
- **Bottom-Up Estimation**
  - Breaks the project into smaller tasks, estimates each task individually, and **aggregates** these estimates to get the total effort.
  - Offers detailed and accurate estimates but can be time-intensive.

Choosing the right techniques:

- **Choosing the Right Technique:**
  - There's **no one-size-fits-all** solution. The best technique depends on:
    - Project size and complexity
    - Available data and historical records
    - Team experience and expertise
    - Level of detail required
- **Tips for Accurate Estimation:**
  - **Involve multiple stakeholders:** Seek diverse perspectives from developers, testers, and project managers.
  - **Consider risks and uncertainties:** Include buffer time and resources for potential challenges.
  - **Use multiple techniques:** Combine different approaches for a more comprehensive view.
  - **Communicate regularly:** Update estimates as the project progresses and share them with stakeholders

Expert Judgement

3.4.1

Expert Judgement:

- Expert Judgment is a **simple yet valuable technique** for project estimation.
- This method is often employed in the **early stages** of a project when **detailed data is scarce.**
- It relies on the insights and experience of seasoned professionals to provide a **qualitative and subjective approach** to estimating project costs and effort.

Definition and Purpose:

- Expert Judgment involves seeking input from **individuals knowledgeable** about the application or development environment.
- The goal is to obtain an **approximate estimate** of the effort required for tasks, especially when **detailed data is not available.**

Expert Judgement:

- **When to Use:**
  - Ideal for **initial project phases** with limited data.
  - Useful for quick, high-level estimates.
  - Often employed when estimating the effort needed to **modify existing software**.
- **Process:**
  - **Identify Experts:** Gather a team of individuals with relevant experience and knowledge.
  - **Collect Inputs:** Experts examine existing code, assess the proportion of code affected, and provide estimates based on their knowledge.
  - **Consolidate Opinions:** Combine the inputs to form a cohesive estimate.

Expert Judgement:

- **Advantages:**
  - **Speed:** Quick to implement compared to data-driven methods.
  - **Simplicity:** Easy to understand and apply.
  - **Expertise Utilization:** Leverages the **practical experience** of skilled professionals familiar with the software.
- **Challenges:**
  - **Subjectivity:** Estimates can vary widely based on **individual biases** and experiences.
  - **Accuracy:** Not always precise, especially for complex projects.
  - **Over-Reliance:** Depending **solely** on expert judgment can lead to inaccuracies.

Expert Judgement:

- **Enhancing Accuracy:**
  - **Combine Techniques:** Use Expert Judgment alongside other methods such as the **Delphi Technique or analogy-based estimation** for a more rounded estimate. Experts often use an informal analogy approach, identifying similar past projects and supplementing with bottom-up estimating.
  - **Regular Updates:** Continuously update estimates as more data becomes available.
  - **Involve Multiple Experts:** Combine the opinions of multiple experts to minimize individual biases and improve accuracy.
- **Delphi Technique:**
  - A method to combine the opinions of **several experts**.
  - Helps tackle group decision-making by iteratively refining estimates through **anonymous feedback** until consensus is reached.

Estimating by Analogy

3.4.2

Estimating by Analogy:

- Estimating by analogy, also known as **case-based reasoning**, is a project estimation technique where the estimator identifies **completed projects (source cases) with similar characteristics** to the new project (the target case).
- The effort recorded for the matching source case is then used as a **base estimate** for the target project. Adjustments are made to **account for differences** between the target and source projects to produce a final estimate.
- **Process:**
  - **Identify** completed projects (source cases) with similar characteristics to the new project (target case).
  - **Use the effort** recorded for the source case as the base estimate.
  - Identify and **adjust for differences** between the target and source projects to refine the estimate.

Estimating by Analogy:

- **Advantages:**
  - **Utilizes Historical Data:** Makes use of information from previous projects to inform estimates.
  - **Efficiency:** Can be faster than building estimates from scratch, especially when **sufficient** historical data is available.
  - **Practical Insight:** Provides practical insights based on real-world project outcomes.
- **Challenges:**
  - **Identifying Similarities and Differences:** Accurately identifying the similarities and differences between projects can be difficult, especially with a large number of past projects to analyze.
  - **Data Dependency:** Relies heavily on the availability and quality of historical project data.

Estimating by Analogy:

- **Automation and Tools:**
  - The **ANCEL software tool** is an example of an attempt to automate this selection process.
  - ANCEL identifies the **source case closest to the target** by measuring the **Euclidean distance** between cases. The Euclidean distance is calculated as:

$$\text{distance} = \sqrt{\sum_{i=1}^{n} (\text{target\_parameter}_i - \text{source\_parameter}_i)^2}$$

- **Ideal Use Cases:**
  - Suitable when there is information about previous projects **but not enough data** to draw generalized conclusions about typical productivity rates or useful drivers.
  - Effective in environments where **historical project data is readily available** and can be **leveraged** for new project estimates.

3.4.3

Bottom-Up Estimating

Bottom Up Estimating:

- Bottom-Up Estimating is a project estimation technique where the **overall project is broken down into smaller**, more manageable tasks.
- This approach allows for more accurate and detailed estimation by considering each **component separately** and **summing up** the individual estimates to get the **total project effort**.
- **Process:**
  - The estimator **begins by breaking the project** into its component tasks. For large projects, this process is **iterative**, where each task is **decomposed** into subtasks.
  - The decomposition **continues until** the tasks are small enough for an individual to complete in a **week or two.**
  - The estimated effort for each activity is then **summed up** to provide the overall estimate.

Bottom Up Estimating:

- **Difference from Top-Down Approach:**
  - Although the project is initially **analyzed from the top** (creating a work breakdown schedule or WBS), the bottom-up part specifically refers to aggregating the effort estimates for each task.
  - Top-down analysis is an essential precursor but distinct from bottom-up estimating.

- **Ideal Use Cases:**
  - Best suited for **later, more detailed stages** of project planning.
  - Necessary for projects that are **novel or lack** historical data.

Bottom Up Estimating:

- **Advantages:**
  - Provides **detailed and accurate** estimates by **focusing** on smaller, manageable tasks.
  - Helps identify specific resource requirements for each task.

- **Challenges:**
  - **Time-consuming** due to the detailed breakdown required.
  - Requires **thorough knowledge** of the project components and dependencies.
  - **Assumptions about the final system** and project methods may need to be made if used in early stages.

Top-Down Approach and Parametric Models

3.4.4

Top-Down Estimating:

- The top-down approach is commonly associated with **parametric (or algorithmic) models**.
- This method is particularly useful for estimating costs and efforts in projects where the **final system's characteristics are known.**
- For example, insurance companies use parametric models to estimate rebuilding costs based on parameters such as the number of storeys and floor space.
- **Top-Down Approach:**
  - Involves estimating project effort based on **high-level parameters.**
  - Useful for rebuilding cost estimation by insurance companies.
  - **Example:** Estimating costs using parameters like floor space and number of storeys.

Top-Down Estimating:

- **Project Effort:**
  - Effort is related to **variables associated** with the final system's characteristics.
  - **Parametric model formula:**
    - Effort = (system size) × (productivity rate)
  - **Example:** Using 'thousands of lines of code' **(KLOC)** as the system size.
- **Forecasting Software Development Effort:**
  - Involves assessing the **amount of work** needed and the **rate of work**.
  - **Example:** Amanda estimates a module to be **2 KLOC.** If Kate works at **40 days per KLOC**, the task would take **2 × 40 = 80 days.** Ken, with less experience, needs **55 days per KLOC**, resulting in **2 × 55 = 110 days.**
  - KLOC serves as a size driver, while developer experience is a productivity driver.

Top-Down Estimating:

- **Productivity Rate Calculation:**
  - **Productivity = effort / size**
  - Calculated using past project data (work-days and KLOC).

- **Advanced Calculation:**
  - Uses least squares regression:
    - **Effort = $\text{constant}_1 + (\text{size} \times \text{constant}_2)$**
  - Parametric models can focus on **different factors like** function points, task size, and productivity.

THANKS!

Do you have any questions?

theciceerguy@p m.me

9841893035

ghimires.com.np