UNIT II: Software Project Planning

1. Software Project Planning Objectives

1.1 Introduction

Software project planning is the process of defining the scope, objectives, and activities for a software development project. Effective planning ensures that the project is completed on time, within budget, and meets the quality expectations of stakeholders.

The main objectives of software project planning include:

- **Scope Definition**: Clearly defining what will and will not be included in the project.
- **Time Estimation**: Estimating the time required to complete various phases of the project.
- **Resource Allocation**: Determining the necessary resources (human, technical, and financial) required for the project.
- **Cost Estimation**: Estimating the total cost of the project, including development, testing, deployment, and maintenance.
- **Risk Management**: Identifying potential risks and defining strategies to mitigate them.
- **Quality Assurance**: Defining criteria for ensuring that the software meets the required quality standards.

1.2 Importance of Software Project Planning

A well-planned project increases the chances of success and reduces the likelihood of project failure. It helps:

- **Set clear goals and milestones.**
- **Ensure efficient resource use.**
- **Identify potential issues early.**
- **Manage risks proactively.**
- **Track progress against timelines.**

---

2. Decomposition Techniques in Software Planning

Decomposition techniques are essential tools in software planning, allowing project managers and developers to break down complex projects into smaller, more manageable tasks. This "divide and conquer" approach enhances project organization, estimation accuracy, and overall success.

2.1 S/W Sizing

2.1.1 Definition

Software sizing refers to determining the size or scale of the software being developed. The size can be measured in various ways, such as lines of code (LOC), function points, or object points. Sizing helps estimate the resources (time, effort, and cost) needed to complete the project.

2.1.2 Techniques for Software Sizing

1. **Lines of Code (LOC)**: Measures the number of lines of source code in the software. It is one of the simplest but most commonly used metrics.
    - **Advantages**: Easy to measure.
    - **Disadvantages**: It does not account for complexity or quality of the code.
2. **Function Points**: Measures the functionality delivered to the user, based on factors such as inputs, outputs, user interactions, and data files.
    - **Advantages**: Provides a more reliable estimate than LOC, especially for large and complex software.
    - **Disadvantages**: Requires detailed analysis and can be time-consuming.
3. **Object Points**: Measures the number of objects, such as classes or components, in object-oriented software.
    - **Advantages**: Useful for object-oriented development.
    - **Disadvantages**: Less commonly used compared to other methods.

2.2 Problem-based Estimation

2.2.1 Definition

Problem-based estimation focuses on understanding the nature of the problem being solved rather than focusing on the technical details of the software. It considers the problem's complexity, scope, and the overall solution approach.

2.2.2 Approach

1. **Problem Analysis**: The software project is broken down into its main components or modules, and each is analyzed in terms of complexity, functionality, and size.
2. **Identify Past Projects**: Compare the current project with similar past projects to make rough estimates.
3. **Expert Judgment**: Seek expert advice to estimate the time and resources required based on previous experience with similar projects.

2.2.3 Advantages:

- Focuses on the problem domain, not just the technical solution.
- Can be more adaptable to new projects where previous data might not be available.

2.2.4 Disadvantages:

- Relies heavily on subjective judgment, which may introduce bias.
- Difficult to use for complex or novel projects.

2.3 Process-based Estimation

2.3.1 Definition

Process-based estimation involves estimating the time and effort required to complete a project based on predefined processes and methodologies. It is rooted in standard practices and historical data from similar projects.

2.3.2 Approach

1. **Identify Process Phases**: Break the software development process into phases such as requirements gathering, design, development, testing, and deployment.
2. **Estimate Effort per Phase**: Estimate the effort and time required for each phase based on historical data or predefined standards.
3. **Use of Historical Data**: Use data from past projects to estimate the time required for similar activities.

2.3.3 Advantages:

- More structured and systematic compared to problem-based estimation.
- Easier to justify and track as the estimation process is based on documented procedures.

2.3.4 Disadvantages:

- Less flexibility when dealing with unique or novel projects.
- May lead to underestimation or overestimation if historical data is not accurate.

---

3. Cost Estimation Models

3.1 COCOMO Model

3.1.1 Introduction to COCOMO

The **COCOMO (Constructive Cost Model)** is a popular software cost estimation model developed by Barry Boehm in 1981. It provides a framework for estimating the cost, effort, and time required to develop software based on project size and other factors.

3.1.2 Types of COCOMO Models

1. **Basic COCOMO**: The simplest form of COCOMO, where the cost is estimated based on the size of the software (measured in LOC).
    - **Effort** = a * (KLOC)^b
        - Where:
            - `KLOC` = the size of the software in thousands of lines of code.
            - `a` and `b` are constants derived from historical data.
2. **Intermediate COCOMO**: Takes into account additional cost drivers, such as product, hardware, personnel, and project attributes.
    - **Effort** = a * (KLOC)^b * EAF
        - Where:
            - `EAF` = Effort Adjustment Factor, which is a product of various factors like complexity, team capability, etc.
3. **Detailed COCOMO**: Involves a more detailed breakdown of the project, considering factors such as individual project phases, required resources, and environmental factors.
    - Provides a more granular estimate of cost and effort but requires more data input.

3.1.3 Cost Estimation Formulae

COCOMO uses the following basic formula to calculate effort:

- **Effort (Person-Months)** = a * (KLOC)^b

- Where:
  - `a` = Constant specific to the type of project.
  - `b` = Exponent specific to the type of project.
  - `KLOC` = Size of the project in thousands of lines of code.

3.1.4 COCOMO Cost Drivers

COCOMO uses several **cost drivers** to adjust the basic cost estimate:

1. **Product attributes**: Reliability, complexity, etc.
2. **Hardware attributes**: Availability of hardware and tools.
3. **Personnel attributes**: Experience and skill level of the team.
4. **Project attributes**: Requirements volatility, documentation needs, etc.

3.1.5 Advantages of COCOMO:

- Provides a quantitative basis for estimating software costs.
- Can be adapted to different project types and sizes.
- Can be refined to produce more accurate estimates as the project progresses.

3.1.6 Disadvantages of COCOMO:

- Requires accurate size estimation (KLOC), which can be difficult in early stages of development.
- Assumes that all projects can be estimated using similar parameters, which may not always be true.
