Software Project Management (SPM)

Subject Code: KCS-601 / RCS-601 (or similar)

University: Dr. A.P.J. Abdul Kalam Technical University (AKTU), Lucknow

Year: 3rd Year, B.Tech (CSE/IT)

Table of Contents

* Critical Path Method (CPM)
* Program Evaluation and Review Technique (PERT)
* Gantt Charts
* Earned Value Management (EVM)

4. **UNIT 4: Risk Management & Quality Management**
* Introduction to Risk Management
* Risk Identification
* Risk Analysis (Qualitative and Quantitative)
* Risk Response Planning
* Software Quality Assurance (SQA)
* Quality Metrics & Standards (ISO 9000, CMMI)

5. **UNIT 5: Project Closure, Team Management & SCM**
* Project Closure
* Project Closure Report
* Organizational Structures
* Team Management & Motivation Theories
* Software Configuration Management (SCM)
* Baselines and Change Control

UNIT 1: Introduction and Software Project Planning

1.1 What is a Project?

A project is a temporary endeavor undertaken to create a unique product, service, or result.

Key Characteristics of a Project:

* **Temporary:** Every project has a definite beginning and a definite end. It's not a routine, ongoing operation.

* **Unique Output:** The outcome of the project is a unique product, service, or result. For example, developing a new version of an operating system is a project.

* **Progressive Elaboration:** Projects are often defined broadly at the beginning and become more specific and detailed as the project team develops a better and more complete understanding of the objectives and deliverables.

* **Requires Resources:** Projects consume resources, which can be people, hardware, software, or money.

* **Has a Primary Sponsor/Customer:** Projects have a customer or sponsor who provides the direction and funding.

1.2 Software Projects vs. Other Types of Projects

Software projects have unique characteristics that distinguish them from other engineering projects (like building a bridge):

* **Invisibility:** The product (software) is intangible. It's hard to see progress. A manager can't "see" 80% complete software like they can see an 80% complete building.

* **High Complexity:** Software systems can have a very high degree of complexity with numerous interdependencies.

* **Flexibility and Changeability:** Software is extremely easy to change. This is both a strength and a weakness. It leads to a high frequency of change requests from clients, which can disrupt project plans.

* **Unique Development Process:** Software development does not follow the same rigid physical laws as other engineering disciplines.

1.3 Project Management Triangle (The Iron Triangle)

Project management is fundamentally about balancing competing constraints. The three primary constraints are often visualized as the "Iron Triangle."

* **Scope/Quality:** What work will be done? What are the features and functionalities of the final product? The quality of the final product is also a part of this vertex.

* **Time/Schedule:** How long will it take to complete the project? This is the timeline, including deadlines and milestones.

* **Cost/Resources:** What will the project cost? This includes all resources: labor, hardware, software, and other expenses.

The Principle: The Iron Triangle illustrates that the three constraints are interrelated.

* If you want to **decrease the time**, you might need to **increase the cost** (hire more people) or **reduce the scope** (deliver fewer features).

* If the **cost is cut**, you might need **more time** to complete the work or **reduce the scope**.

* If the **scope increases** (scope creep), it will likely require **more time and/or more cost**.

A successful project manager must balance these three goals.

Page 2

1.4 The 4 P's of Project Management

A successful software project is managed by focusing on four key elements:

1. **People:** The most important element. It involves managing stakeholders, the project team, and the organizational structure. Issues include team building, motivation, and communication.

2. **Product:** The software to be built. The manager must understand the product's objectives, scope, and requirements. Without a clear product definition, the project is bound to fail.

3. **Process:** The framework of activities and tasks required to build the software. A good process provides a roadmap for the team and a structure for managing the project. This includes choosing a suitable life cycle model.

4. **Project:** All the work required to make the product a reality. This involves planning, estimating, scheduling, monitoring, and controlling all the activities.

1.5 Project Stakeholders

Stakeholders are individuals or organizations who are actively involved in the project, or whose interests may be positively or negatively affected by the execution or completion of the project.

Key Stakeholders in a Software Project:

* **Project Manager:** The person responsible for managing the project.

* **Customer/Client:** The individual or organization that will use the project's product. They are the primary source of requirements and funding.

* **Development Team:** The group of people who perform the work of creating the product (developers, testers, designers, etc.).

* **Sponsor:** The person or group who provides the financial resources for the project. Often a senior executive in the organization.

* **End-Users:** The people who will actually use the software once it's delivered. Their feedback is crucial for usability.

1.6 Software Project Life Cycle Models

The chosen life cycle model provides the overall framework for managing the project.

1. **Waterfall Model:**

   * A linear, sequential approach. Each phase must be completed before the next phase can begin.

   * **Phases:** Requirements -> Design -> Implementation -> Testing -> Deployment -> Maintenance.

   * **Pros:** Simple to understand and manage. Disciplined.

   * **Cons:** Inflexible. Not suitable for projects with uncertain requirements. No working software is produced until late in the cycle.

2. **Prototyping Model:**

   * A working prototype of the system is built to understand the requirements.

   * The prototype is shown to the client for feedback, and refined until the requirements are clear.

   * **Pros:** Good for projects where requirements are not well understood. Reduces risk.

   * **Cons:** The client may mistake the prototype for the final system. Can be time-consuming.

3. **Spiral Model:**

* An evolutionary model that combines features of prototyping and the waterfall model.

* The project passes through four phases in an iterative spiral: Planning, Risk Analysis, Engineering (Development & Test), and Evaluation.

* **Pros:** Excellent for large, complex, and high-risk projects. Risk analysis is built-in.

* **Cons:** Complex to manage. Requires expertise in risk assessment.

4. **Agile Models (e.g., Scrum, XP):**

* An iterative and incremental approach focused on collaboration, customer feedback, and rapid releases.

* Work is done in short cycles called "sprints" or "iterations" (typically 1-4 weeks).

* **Pros:** Highly flexible and adaptive to change. Delivers value to the customer quickly. Promotes strong team collaboration.

* **Cons:** Requires a high degree of customer involvement. Less predictable in terms of final cost and time.

UNIT 2: Project Size Estimation

Project estimation is the process of predicting the most realistic amount of effort (in person-months), time, and cost required to build a software system.

2.1 The Estimation Process

1. **Establish Project Objectives:** Understand the goals and scope of the project.

2. **Plan for Estimation:** Decide which estimation techniques will be used.

3. **Decompose the Project:** Break down the project into smaller, more manageable pieces. This is typically done using a Work Breakdown Structure (WBS).

4. **Estimate Effort and Cost:** Apply estimation techniques to the smaller pieces.

5. **Review and Reconcile:** Aggregate the estimates for the smaller pieces to get an overall project estimate. Review for accuracy and make adjustments.

2.2 Decomposition Technique: Work Breakdown Structure (WBS)

A WBS is a hierarchical decomposition of the total scope of work to be carried out by the project team. It breaks down the project into smaller, manageable components called work packages.

Example WBS for a "Student Information System":

* **1.0 Student Information System**
  * **1.1 Project Management**
  * **1.2 Requirements Gathering**
    * 1.2.1 Conduct Stakeholder Interviews
    * 1.2.2 Document Requirements
  * **1.3 System Design**
    * 1.3.1 Database Design
    * 1.3.2 UI/UX Design
    * 1.3.3 Architecture Design
  * **1.4 Module Development**
    * 1.4.1 Student Registration Module
    * 1.4.2 Course Management Module
    * 1.4.3 Grade Reporting Module
  * **1.5 Integration & Testing**
    * 1.5.1 Unit Testing
    * 1.5.2 Integration Testing
    * 1.5.3 System Testing
  * **1.6 Deployment**

Why use a WBS?

* Provides a clear view of the work.
* Helps in assigning responsibilities.
* Forms the basis for estimating, scheduling, and controlling.

2.3 Estimation Metrics

* **Size-Oriented Metrics:** Based on the size of the software produced. The primary metric is **Lines of Code (LOC)** or Kilo Lines of Code (KLOC).

* **Function-Oriented Metrics:** Based on the functionality delivered by the software. The primary metric is **Function Points (FP)**.

2.4 Lines of Code (LOC) Based Estimation

This technique uses the estimated size of the software in LOC to estimate effort, cost, and duration.

Process:

1. Estimate the total KLOC for the project.
2. Use historical data to find productivity (e.g., KLOC per person-month).
3. Calculate effort: `Effort (in person-months) = Total KLOC / Productivity`
4. Calculate cost: `Cost = Effort * Cost per person-month`
5. Calculate duration: `Duration = Effort / Number of people` (This is a simplified view).

Pros:

* Simple to use.
* Widely used in the past.

Cons:

* **Language Dependent:** A project with 1000 lines of C++ is not the same as one with 1000 lines of Python.
* **Poor Estimator:** LOC is a poor measure of complexity and functionality. It penalizes well-designed, shorter code.
* **Hard to Estimate:** It's difficult to accurately estimate the LOC at the beginning of a project.
* What counts as a "line"? Comments? Blank lines? This ambiguity leads to inconsistencies.

Example:

* Estimated size = 30,000 LOC (30 KLOC).

* Historical productivity of the team = 0.5 KLOC per person-month.

* Effort = 30 / 0.5 = 60 person-months.

* If the average salary is ₹50,000 per month, Total Cost = 60 * 50,000 = ₹3,000,000.

2.5 Function Point (FP) Based Estimation

Developed by Allan Albrecht at IBM, Function Point Analysis (FPA) is a method to measure software size based on its functionality from the user's point of view. It is language-independent.

Process:

1. **Calculate Unadjusted Function Points (UFP):**

   Identify and count five types of components:

   * **External Inputs (EI):** Data going into the system (e.g., a form submission).

   * **External Outputs (EO):** Data coming out of the system (e.g., a report).

   * **External Inquiries (EQ):** A request/response pair that retrieves data but doesn't alter it (e.g., a search query).

   * **Internal Logical Files (ILF):** Data stored and maintained within the system (e.g., a database table of students).

   * **External Interface Files (EIF):** Data used by the system but stored in another system (e.g., accessing an external user authentication service).

   Each component is weighted based on its complexity (Simple, Average, Complex).

   (A table showing complexity weights for EI, EO, EQ, ILF, EIF would be here)

   UFP = Sum of (Count of each component type * Weight)

2. **Calculate the Value Adjustment Factor (VAF):**

The VAF adjusts the UFP based on 14 General System Characteristics (GSCs). Each characteristic is rated on a scale of 0 (not present) to 5 (strongly present).

The GSCs include things like data communications, distributed processing, performance, reusability, ease of installation, etc.

The sum of these ratings is the Total Degree of Influence (TDI).

TDI = Sum of all 14 ratings

VAF = 0.65 + (0.01 * TDI)

3. **Calculate the Final Adjusted Function Points (FP):**

FP = UFP * VAF

Using FP for Estimation:

Once you have the FP count, you can use historical data to estimate effort.

* Productivity = FP per person-month

* Effort = Total FP / Productivity

Pros:

* Language independent.

* Based on user requirements.

* Can be estimated early in the project life cycle.

Cons:

* Subjective (complexity and GSC ratings).

* Requires training to apply correctly.

2.6 COCOMO Model (Constructive Cost Model)

Developed by Barry Boehm, COCOMO is an algorithmic cost estimation model that uses the size of the project (in KLOC) as the primary input.

Basic COCOMO

Project Modes:

1. **Organic Mode:** Small, simple projects, experienced team.
2. **Semi-detached Mode:** Intermediate size and complexity, mixed experience team.
3. **Embedded Mode:** Operates within tight constraints, complex.

Basic COCOMO Formulas:

Project Mode	Effort (E) in Person-Months	Development Time (D) in Months
Organic	E = 2.4 * (KLOC)^1.05	D = 2.5 * (E)^0.38
Semi-detached	E = 3.0 * (KLOC)^1.12	D = 2.5 * (E)^0.35
Embedded	E = 3.6 * (KLOC)^1.20	D = 2.5 * (E)^0.32

Number of People Required (N) = E / D

Example Calculation:

A project is estimated to be 40 KLOC and is a **semi-detached** project.

1. **Calculate Effort (E):**
   E = 3.0 * (40)^1.12 ≈ 181.7 person-months.
2. **Calculate Development Time (D):**
   D = 2.5 * (181.7)^0.35 ≈ 17.25 months.
3. **Calculate People Required (N):**
   N = 181.7 / 17.25 ≈ 11 people.

Page 7

Intermediate COCOMO (COCOMO II)

Intermediate COCOMO improves upon the Basic model by introducing 15 "cost drivers" that adjust the nominal effort.

Effort Formula:

E = a_i * (KLOC)^(b_i) * EAF

Here:

* a_i and b_i are constants from the Basic model.

* **EAF (Effort Adjustment Factor)** is a multiplier calculated from the 15 cost drivers.

Cost Drivers:

Grouped into four categories:

1. **Product Attributes:** Reliability, Database size, Complexity.

2. **Hardware Attributes:** Time/Storage constraints, VM volatility.

3. **Personnel Attributes:** Analyst/Programmer capability, Experience.

4. **Project Attributes:** Use of modern practices/tools, Schedule constraints.

Each driver is rated (e.g., Very Low to Extra High), and each rating has a multiplier.

Calculating EAF:

The EAF is the product of the effort multipliers for all 15 cost drivers.

A nominal project has all multipliers equal to 1.0, so EAF = 1.0. Intermediate COCOMO provides a much more refined estimate.

Page 8

UNIT 3: Project Scheduling and Tracking

3.1 Basic Concepts of Scheduling

* **Task/Activity:** A piece of work with a start, end, and duration.

* **Milestone:** A significant event with zero duration (e.g., "Design Complete").

* **Deliverable:** A tangible output (e.g., a document, code).

* **Dependency:** A relationship between tasks (e.g., Task B can't start until A finishes).

3.2 Activity Networks

A graphical representation of project tasks and their dependencies.

1. **Activity on Arrow (AOA):** Activities are arrows, nodes are events.

2. **Activity on Node (AON):** Activities are nodes (boxes), arrows show dependencies. AON is more common.

AON Example:

Tasks: A(3 days), B(4 days, needs A), C(2 days, needs A), D(5 days, needs B & C).