SPM

Software Project Management

(LECTURE NOTES)

Prepared by:

Er. AKSHAY KUMAR PATRA

(Assistant Professor)

DEPT. OF COMPUTER SCIENCE. & ENGINEERING

Modern Engineering and Management Studies

Banaparia, Kuruda, Balasore, Odisha.

Module-1

Software Project Management

Software Project Management (SPM) is the application of knowledge, skills, tools, and techniques to plan, execute, and control software projects. The goal is to deliver high-quality software on time, within budget, and according to user requirements.

Software Projects

Software projects involve activities such as requirement gathering, designing, coding, testing, deployment, and maintenance. These are distinct from other types of engineering projects because of:

- **Intangibility**: Software is not physical.
- **Complexity**: Software systems often have many interconnected parts.
- **Flexibility**: Software can be changed more easily than hardware.
- **Uncertainty**: Requirements often evolve during development.

Ways of Categorizing Software Projects

Software projects can be categorized by several dimensions:

- **Size**: Small (e.g., a website), Medium (e.g., inventory system), Large (e.g., ERP system).
- **Application domain**: Web, mobile, embedded, enterprise, etc.
- **Complexity**: Simple UI to complex distributed systems.
- **Development model**: Agile, Waterfall, V-model, Spiral, etc.
- **Risk**: Safety-critical, business-critical, low-risk.

Problems with Software Projects

Common problems include:

- **Unclear requirements**
- **Scope creep** (uncontrolled changes)
- **Poor planning**
- **Underestimation of effort/time**
- **Lack of skilled personnel**
- **Poor communication among stakeholders**
- **Inadequate testing**
- **Failure to meet deadlines or budget**

Project Life Cycle

Typical phases in a software project life cycle:

1. **Initiation**: Define the project's purpose, feasibility.
2. **Planning**: Schedule, resources, risks, costs.
3. **Execution**: Development and testing.
4. **Monitoring & Control**: Track progress, manage changes.
5. **Closure**: Final delivery and project review.

Project Management

Involves:

- **Planning**: Define scope, objectives, schedule.
- **Organizing**: Build a team and allocate resources.
- **Leading**: Direct and motivate team members.
- **Controlling**: Monitor performance, quality, and costs.

Setting Objectives

Effective objectives should be SMART:

- **Specific**
- **Measurable**
- **Achievable**
- **Relevant**
- **Time-bound**

Example: "Deliver a working prototype of the booking system in 3 months."

Stakeholders

Stakeholders are individuals or groups affected by the project or its outcome:

- **Internal**: Project team, managers, developers.
- **External**: Clients, customers, suppliers, regulatory bodies.

9. Project Team

A typical software project team includes:

- **Project Manager**
- **Developers**
- **Testers**
- **Business Analysts**
- **UI/UX Designers**
- **DevOps Engineers**

Team roles should be well-defined to avoid overlap and confusion.

Step Wise: An Overview of Project Planning

The **Step Wise** method is a structured approach to planning:

1. Identify project scope and objectives.
2. Identify project infrastructure.
3. Analyze project characteristics.
4. Identify products and activities.
5. Estimate effort for each activity.
6. Identify and allocate resources.
7. Create a schedule.
8. Risk analysis.
9. Monitor and revise the plan.

Project Evaluation

Evaluation ensures the project is justified and feasible:

- **Technical feasibility**: Can we build it?
- **Economic feasibility**: Is it worth it?
- **Operational feasibility**: Will it work in practice?
- **Legal feasibility**: Any legal constraints?

Selection of Appropriate Project Approach

Common approaches:

- **Waterfall**: Linear, sequential.
- **Agile**: Iterative, incremental.
- **Spiral**: Risk-driven, iterative.
- **V-Model**: Emphasizes verification and validation.

Selection depends on project size, complexity, client needs, and team experience.

Software Size Estimation

Estimating size is crucial for planning and pricing:

- **Lines of Code (LOC)**
- **Function Points (FP)** – measures functionality from the user's perspective.
- **Use Case Points (UCP)** – based on use case complexity.

Estimation of Effort & Duration

Common techniques:

- **Expert judgment**
- **Delphi method** (consensus from experts)
- **Analogy-based estimation**
- **Parametric models** (like COCOMO)

COCOMO Models (Constructive Cost Model)

Developed by Barry Boehm, COCOMO estimates effort and time based on project size (in KLOC):

- **Basic COCOMO**: Uses a simple formula.
- **Intermediate COCOMO**: Adds cost drivers (e.g., complexity, reliability).
- **Detailed COCOMO**: Includes phase-wise effort estimation.

Effort (in person-months) = $a \times (KLOC)^{b}$

Where a and b are constants based on project type (organic, semi-detached, embedded).

Putnam's Work (SLIM Model)

Putnam's model uses **Rayleigh curve** to model effort over time. The SLIM (Software Life-cycle Management) model is used to estimate staffing levels and schedule.

Key concept: Effort = (Size / Productivity)^3

Jensen's Model

A refinement of Putnam's model, Jensen's model incorporates multiple parameters such as team size and productivity. It's based on empirical data and considers the impact of staff buildup and decay.

Halstead's Software Science

Introduced by Maurice Halstead, this model measures software complexity using:

- $\mathbf{n_1}$ = number of distinct operators
- $\mathbf{n_2}$ = number of distinct operands
- $\mathbf{N_1}$ = total number of operators
- $\mathbf{N_2}$ = total number of operands

From these, Halstead defines:

- **Program Length** = $N = N_1 + N_2$
- **Vocabulary** = $n = n_1 + n_2$
- **Volume** = $N \times \log_2(n)$
- **Effort** = Volume / (2 × Level)

Used for estimating program complexity, maintainability, and effort.

MODULE-2

1. Activity Planning in Software Project Management

a. Project Schedules

Refers to the timeline of tasks and milestones in a software project. Scheduling ensures that work is completed on time, within scope, and on budget.

b. Sequencing and Scheduling Projects

- **Sequencing**: Determining the order of project activities.
- **Scheduling**: Assigning start and end dates based on dependencies, resources, and durations.

---

2. Network Planning Models

a. AON (Activity on Node)

- Nodes represent activities.
- Arrows show dependencies.
- Easier to read and more commonly used in software project management.

b. AOA (Activity on Arrow)

- Arrows represent activities.
- Nodes represent events (start/end).
- Less intuitive for complex software projects.

---

3. Identifying Critical Activities

- Critical activities directly affect the project deadline.
- Found using **Critical Path Method (CPM)**.
- **Critical Path**: The longest path through the project with no slack time.
- Any delay in these activities will delay the entire project.

---

4. Crashing and Fast Tracking

a. Crashing

- Adding extra resources to reduce activity duration.
- Increases cost but shortens schedule.

b. Fast Tracking

- Performing activities in parallel that were originally scheduled sequentially.
- Increases risk but may reduce duration.

5. Risk Management in Software Projects

a. Risk Categories

- **Technical risks** (e.g., new tech, performance issues)
- **Project management risks** (e.g., estimation errors)
- **Organizational risks** (e.g., team turnover)
- **External risks** (e.g., regulations, market shifts)

b. Risk Planning, Management, and Control

- **Risk Planning**: Identifying risks and planning responses.
- **Risk Management**: Monitoring and managing risks during the project.
- **Risk Control**: Implementing mitigation or contingency plans when risks materialize.

c. Evaluating Risks to the Schedule

- Analyze risk impact on schedule.
- Use techniques like **Monte Carlo Simulation** or **PERT analysis**.

6. PERT (Program Evaluation and Review Technique)

- Used for time estimation in uncertain situations.
- Estimates based on:
  - **Optimistic (O)**
  - **Pessimistic (P)**
  - **Most Likely (M)**
  - **Expected Time (TE)**:

    TE=O+4M+P6TE = \frac{O + 4M + P}{6}TE=6O+4M+P

- Useful for managing uncertainty in software projects.

7. Resource Allocation

a. Identifying Resource Requirements

- Determine the people, tools, and technologies needed.
- Includes developers, testers, project managers, etc.

b. Scheduling Resources

- Assign resources to tasks.
- Avoid over-allocation and resource conflicts.

c. Creating Critical Paths

- Align critical path with resource constraints.
- Consider resource availability while calculating the path.

d. Publishing Schedule

- Share finalized schedule with stakeholders.
- Should include milestones, dependencies, and resource plans.

e. Cost Schedules

- Combine schedule with cost estimates.
- Useful for budgeting and cost control.

f. Sequence Schedule

- Ensure the logical flow of activities.
- Maintain dependencies and optimal task order.

---

8. CPM (Critical Path Method)

- Determines the sequence of critical tasks.
- Used for identifying the shortest time to complete the project.
- Highlights which tasks can be delayed without affecting the project.

---

9. Gantt Chart

- Visual timeline of project activities.
- Shows start/end dates, duration, dependencies.
- Ideal for tracking progress and milestones.

---

10. Staffing in Software Projects

- Assigning people to project roles.
- Based on skills, experience, and availability.
- May use **RACI matrix** (Responsible, Accountable, Consulted, Informed).

---

11. Organizing a Software Engineering Project

- Define project structure (Agile, Waterfall, Hybrid).
- Set up teams, roles, responsibilities.
- Establish communication plans, documentation standards, version control, and development methodologies.

Summary Diagram (Text-based View):

```
pgsql
CopyEdit
Activity Planning
  ├── Project Scheduling
  ├── Sequencing & Scheduling
  ├── Network Models: AON & AOA
  ├── Critical Path (CPM)
  ├── Crashing / Fast Tracking

Risk Management
  ├── Categories: Technical, Mgmt, Org, External
  ├── Planning / Control
  ├── PERT for Schedule Risk

Resource Allocation
  ├── Identify Requirements
  ├── Schedule Resources
  ├── Create & Sequence Schedule
  ├── Publish & Track Cost

Project Organization
  ├── Gantt Charts
  ├── Staffing
  ├── Team Structure & Processes
```

MODULE-3

1. Monitoring and Control – Visualizing Progress

Monitoring and control are essential processes in project management that ensure a project stays on track concerning **scope, time, cost, and quality**.

Key Concepts:

- **Tracking Progress:** Comparing actual performance against the planned schedule.
- **Visual Tools:**
  - **Gantt Charts:** Show task durations and progress.
  - **Burn-down Charts:** Common in Agile, show work remaining over time.
  - **Dashboards:** Provide at-a-glance updates on KPIs (Key Performance Indicators).

Purpose:

- Identify deviations.
- Take corrective actions early.
- Facilitate stakeholder communication.

---

2. Earned Value Analysis (EVA)

EVA is a technique that integrates **scope, schedule, and cost** to assess project performance and progress objectively.

Key Metrics:

- **Planned Value (PV):** Budgeted cost of work planned.
- **Earned Value (EV):** Budgeted cost of work actually performed.
- **Actual Cost (AC):** Actual cost incurred for performed work.

Performance Indicators:

- **Cost Performance Index (CPI) = EV / AC**
- **Schedule Performance Index (SPI) = EV / PV**
- **Estimate at Completion (EAC):** Forecast of final project cost.

Benefits:

- Provides early warning of performance issues.
- Allows forecast of future performance.

3. Managing People and Organizing Teams

People are the most critical asset in software projects. Effective management ensures motivation, productivity, and collaboration.

Key Areas:

- **Motivation Theories:** Maslow's hierarchy, Herzberg's two-factor theory.
- **Conflict Resolution:** Negotiation, mediation, and consensus building.
- **Communication:** Regular meetings, open channels, feedback loops.
- **Leadership Styles:** Autocratic, democratic, laissez-faire depending on team maturity.

Team Building:

- Forming → Storming → Norming → Performing → Adjourning (Tuckman's model).

---

4. Organizational Structures

Different organizational setups affect communication, authority, and project execution.

Types:

1. **Functional:** Departments like IT, HR. Projects handled within departments.
2. **Projectized:** Full authority given to project managers.
3. **Matrix:** Mix of both. Can be:
    - **Weak Matrix:** PM has limited power.
    - **Strong Matrix:** PM has more control.
    - **Balanced Matrix:** Shared authority.

Implications:

- Communication lines.
- Resource availability.
- Decision-making efficiency.

---

5. Planning for Small Projects

Small projects require a lightweight and flexible planning approach.

Key Practices:

- **Simplified Documentation:** Focus on key deliverables.
- **Timeboxing:** Limit tasks to fixed time periods.
- **Flexible Scope:** Often evolving, less rigid.
- **Minimal Resources:** Often involves a small team, requiring multi-skilled members.

Tools:

- Kanban Boards, Trello, Google Sheets.

---

6. Case Studies

Real-world examples help illustrate how theory is applied.

Common Lessons:

- Miscommunication leads to delays and cost overruns.
- Overly rigid planning can fail in dynamic environments.
- Strong leadership and stakeholder engagement are vital.

Example:

- **Denver Airport Baggage System:** A classic failure due to poor requirement analysis and over-automation.

---

7. Agile Development

Agile is an **iterative and incremental** approach focusing on flexibility, collaboration, and customer feedback.

Core Principles (from Agile Manifesto):

- Individuals and interactions > processes and tools
- Working software > comprehensive documentation
- Customer collaboration > contract negotiation
- Responding to change > following a plan

Agile Practices:

- **Scrum:** Roles (Scrum Master, Product Owner), Sprints, Stand-ups.
- **Kanban:** Continuous flow, WIP limits.
- **XP (Extreme Programming):** Pair programming, TDD, frequent releases.

Benefits:

- Faster delivery.
- Better quality through continuous feedback.
- Improved team morale and stakeholder engagement.

Summary Table:

Topic	Purpose	Tools/Methods
Monitoring & Control	Ensure project stays on track	Gantt, Dashboards, Burn-down Charts
Earned Value Analysis	Measure cost and schedule performance	EV, PV, AC, CPI, SPI
Managing People	Build cohesive, motivated teams	Leadership styles, Communication
Organizational Structures	Define authority and workflow	Functional, Matrix, Projectized
Planning for Small Projects	Efficient execution with limited scope/resources	Timeboxing, Kanban
Case Studies	Learn from real-world success/failure	Denver Airport, FBI Sentinel
Agile Development	Deliver value iteratively and adaptively	Scrum, Kanban, XP

MODULE-4

1. Software Quality & Quality Engineering

Software Quality

- Refers to how well software conforms to functional and non-functional requirements.
- Attributes include reliability, usability, maintainability, efficiency, portability, etc.
- Quality is both **conformance to requirements** and **fitness for use**.

Quality Engineering

- A discipline combining software engineering and quality management to ensure the delivery of high-quality products.
- Involves planning, assurance, control, and improvement.
- Emphasizes proactive processes, automated testing, and continuous improvement.

---

2. Defining Quality Requirements

- Done during the requirements phase.
- Includes both **functional** (e.g., login process) and **non-functional requirements** (e.g., performance, usability).
- Techniques: quality attribute scenarios, goal-question-metric (GQM) approach.
- Helps in setting measurable targets for software quality.

---

3. Quality Standards

Common Software Quality Standards:

- **ISO 9000**: Generic quality management standards.
- **ISO 9001**: Focused on quality management systems (QMS), applicable across industries.
- **IEEE** standards: Specifically for software processes and documentation.
- **CMMI (Capability Maturity Model Integration)**.

Importance:

- Promote consistency and customer confidence.
- Enable certification and benchmarking.

---

4. Quality Practices & Conventions

- **Coding standards** (e.g., naming conventions, indentation).
- **Code reviews**, **pair programming**.
- **Version control**, **unit testing**, **continuous integration**.
- Use of tools like SonarQube for static code analysis.

---

5. ISO 9000 & ISO 9001

ISO 9000

- A family of standards for quality management systems.
- Lays down the fundamentals and vocabulary.

ISO 9001

- Specifies requirements for a QMS.
- Key principles: customer focus, leadership, process approach, improvement.
- Certification indicates a company maintains quality management processes.

---

6. Software Quality Metrics (Matrices)

- **Product Metrics**: Measure software product (e.g., defect density, cyclomatic complexity).
- **Process Metrics**: Measure process effectiveness (e.g., defect removal efficiency).
- **Project Metrics**: Schedule variance, effort variance.
- Examples:
    - **Defect Density** = Total defects / Size of software (KLOC or FP)
    - **Mean Time to Failure (MTTF)**
    - **Customer Problem Metric**

---

7. Managerial and Organizational Issues

- **Commitment to Quality**: From top-level management.
- **Training and Skill Development**: For developers and testers.
- **Quality Budgeting**: Allocate funds for testing, tools, audits.
- **Organizational Culture**: Encourage quality ownership and continuous improvement.

---

8. Defect Prevention

- **Goal**: Reduce defects rather than just detect.
- Methods:
  - **Root Cause Analysis (RCA)**
  - **Process improvements**
  - **Design reviews**
  - **Static code analysis**
  - **Automated testing**

---

9. Reviews & Audits

- **Reviews**:
  - **Peer Reviews**
  - **Walkthroughs**
  - **Technical Reviews**
  - **Inspections** (most formal)
- **Audits**:
  - Conducted by external bodies or QA departments.
  - Ensure compliance with standards and processes.

---

10. SEI Capability Maturity Model (CMM)

CMM Levels:

1. **Initial** – Ad hoc processes.
2. **Repeatable** – Basic project management.
3. **Defined** – Standardized process.
4. **Managed** – Quantitative quality goals.
5. **Optimizing** – Continuous process improvement.

Benefits:

- Structured path for process improvement.
- Enhances predictability and product quality.

---

11. Personal Software Process (PSP)

- Developed by Watts Humphrey.
- Helps engineers improve personal work process.
- Involves:
  - Time tracking
  - Defect tracking
  - Personal estimation techniques
- Encourages individual responsibility for quality.

---

12. Six Sigma

- A data-driven approach for process improvement.
- Goal: 3.4 defects per million opportunities (DPMO).
- Methodologies:
  - **DMAIC**: Define, Measure, Analyze, Improve, Control (for existing processes).
  - **DMADV**: Define, Measure, Analyze, Design, Verify (for new processes).
- Tools: Pareto charts, fishbone diagrams, control charts.

Summary Table

Topic	Key Focus
Software Quality	Conformance and usability
Quality Engineering	Integration of engineering & quality practices
Quality Requirements	Functional + non-functional specifications
Quality Standards	ISO, IEEE, CMMI
Practices & Conventions	Code standards, reviews, testing
ISO 9000/9001	Quality management frameworks
Software Quality Matrices	Metrics to measure process and product quality
Managerial Issues	Budget, leadership, training
Defect Prevention	Root cause analysis, early detection

Topic	Key Focus
Reviews & Audits	Formal assessments for improvement
SEI CMM	Maturity levels for software processes
PSP	Individual productivity and quality management
Six Sigma	Statistical process improvement

Thank You…