SOFTWARE PROJECT MANAGEMENT

Project: “It is a temporary endeavor undertaken to create a unique, service, or result”

- A specific plan or design

Characteristics:

- Non routine
- Planned
- Finite Duration
- Requires resources
- Non-trivial

Task: A small piece of work

Software:

- A collection of programs
- Combined in a package
- To perform different applications

Management: It is an individual or group of individuals that accepts the responsibilities to run an organization.

Features:

- Result oriented
- Dynamic in Nature
- It follows established principles and rules
- It is a continuous and never ending process

Difference between Software projects and Normal projects:

- Tangible vs nontangible
- Complexity
- Flexibility
- Invisibility
- More problematic

Types of project:

- **In-house:** clients and developers are employed by the same Organization
- **Out-sourced:** clients and developers employed by different organizations

Software Project Management:
"It is the process of planning, organizing, leading, and controlling software projects to ensure they are completed on time, within budget, and meet quality standards"

Component of SPM

1. **Project Planning**
    - ➢ Scope Definition: Identifying project goals, deliverables, and boundaries.
    - ➢ Requirements Gathering: Documenting functional and non-functional requirements.
    - ➢ Work Breakdown Structure (WBS): Dividing tasks into manageable sections.
    - ➢ Scheduling: Creating timelines with milestones and deadlines.
    - ➢ Resource Planning: Allocating human, technological, and financial resources.
2. **Project Estimation**
    - ➢ Time Estimation: Calculating the time required for each task or phase.
    - ➢ Cost Estimation: Budgeting for development, tools, and contingencies.
    - ➢ Effort Estimation: Determining the number of hours or days required for tasks.
3. **Risk Management**
    - ➢ Risk Identification: Spotting potential risks early in the project.
    - ➢ Risk Analysis: Assessing the likelihood and impact of risks.
    - ➢ Risk Mitigation Planning: Developing strategies to minimize or handle risks.
4. **Team Management**
    - ➢ Role Definition: Assigning clear roles and responsibilities to team members.

- Communication Management: Establishing channels and tools for effective collaboration.
- Motivation and Leadership: Ensuring team morale and resolving conflicts.

5. **Quality Management**
   - Quality Assurance (QA): Establishing processes to ensure high-quality deliverables.
   - Quality Control (QC): Testing and validating the software against requirements.
   - Standards Compliance: Adhering to industry and organizational standards.
6. **Project Monitoring and Control**
   - Progress Tracking: Monitoring timelines and deliverables.
   - Performance Metrics: Measuring productivity and quality.
   - Issue Management: Identifying and resolving problems quickly.
7. **Stakeholder Management**
   - Engagement: Keeping stakeholders informed and involved.
   - Expectation Management: Aligning project deliverables with stakeholder needs.
   - Feedback Integration: Incorporating stakeholder input into project adjustments.
8. **Configuration Management**
   - Version Control: Managing changes to software artifacts.
   - Baseline Management: Maintaining approved versions of deliverables.
   - Change Control: Handling requests for modifications systematically.
9. **Delivery and Deployment**
   - Implementation Planning: Defining deployment processes.
   - User Training: Providing necessary documentation and training to users.
   - Post-Deployment Support: Ensuring smooth transition and addressing issues after launch.
10. **Project Closure**
    - Final Review: Ensuring all objectives are met and deliverables are accepted.

- Documentation: Archiving lessons learned, project reports, and artifacts.
- Handover: Transitioning the system to maintenance teams or end-users.

Challenge in SPM:

- Unclear Requirements: Incomplete, evolving, or ambiguous requirements can derail project plans, leading to **scope creep, rework, and potential project failure**.
- Time Constraints – Unrealistic deadlines often force teams to **compromise quality**, resulting in **stress, rushed deliverables, and burnout**.
- Budget Limitations – Financial restrictions can limit essential **resources, tools, and personnel**, negatively impacting **quality, delivery timelines, and team morale**.
- Team Collaboration – Poor communication and misalignment within the team lead to **inefficiencies, delays, and internal conflicts**, reducing overall productivity.
- Technical Challenges – Working with complex or unproven technologies can **introduce risks**, causing **delays, increased debugging efforts, and potential redevelopment**.
- Risk Management – Inadequate risk identification and mitigation can result in **unplanned disruptions, cost overruns, and project derailment**.
- Quality Assurance – Balancing speed with high-quality standards is a challenge, increasing the **risk of defects, subpar software, and customer dissatisfaction**.
- Adapting to Change – Rapid technological advancements and shifting market demands require **continuous adaptation**, often disrupting initial project plans.

Opportunities in SPM

1. Innovation in Tools and Processes:
   - ✓ Opportunity: Leverage advanced project management tools (e.g., Agile, DevOps).
   - ✓ Benefit: Improves efficiency, collaboration, and tracking.
2. Global Talent Pool:
   - ✓ Opportunity: Access skilled professionals worldwide.

- ✓ Benefit: Enhances expertise and innovation.

3. Agile Methodologies:
   - ✓ Opportunity: Implement flexible frameworks for iterative development.
   - ✓ Benefit: Allows faster adaptation to changing requirements.
4. Improved Communication Technology:
   - ✓ Opportunity: Use tools like Slack, Zoom, or MS Teams for collaboration.
   - ✓ Benefit: Bridges communication gaps in distributed teams.
5. Data-Driven Decision-Making:
   - ✓ Opportunity: Use analytics and performance metrics to guide decisions.
   - ✓ Benefit: Identifies bottlenecks and optimizes resource allocation.
6. Automation in Testing and Deployment:
   - ✓ Opportunity: Automate repetitive tasks such as code testing.
   - ✓ Benefit: Reduces manual errors and accelerates delivery.
7. Focus on Sustainability:
   - ✓ Opportunity: Adopt eco-friendly and cost-efficient practices.
   - ✓ Benefit: Aligns with organizational goals and public sentiment.
8. Scalability and Cloud Technology:
   - ✓ Opportunity: Use cloud platforms for flexible infrastructure.
   - ✓ Benefit: Facilitates seamless scaling and efficient resource usage.
9. Emphasis on Soft Skills:
   - ✓ Opportunity: Invest in leadership and interpersonal skills.
   - ✓ Benefit: Builds cohesive teams and effective communication.
10. Enhanced Stakeholder Engagement:
    - ✓ Opportunity: Involve stakeholders through collaborative planning.
    - ✓ Benefit: Ensures alignment and increases project acceptance.

Tools in SPM

Category	Purpose	Examples
Project Planning Tools	Create project plans, timelines, and schedules	Microsoft Project, Smartsheet, Monday.com, Asana
Collaboration & Communication Tools	Facilitate team communication and file sharing	Slack, Microsoft Teams, Zoom, Google Workspace
Version Control Systems	Manage code versions and track changes collaboratively	Git, GitHub, Bitbucket, GitLab
Task & Workflow Management Tools	Track tasks, assign responsibilities, and visualize workflows	Jira, Trello, ClickUp, Wrike
Risk Management Tools	Identify, analyze, and mitigate risks	RiskWatch, Active Risk Manager (ARM)
Resource Management Tools	Manage resource allocation and availability	Resource Guru, Hub Planner, TeamGantt
Quality Assurance (QA) Tools	Automate and manage testing processes	Selenium, TestRail, JMeter, Postman
Documentation Tools	Create and manage project documentation	Confluence, Notion, Microsoft OneNote
Time Tracking & Reporting Tools	Track time spent on tasks and generate reports	Toggl, Harvest, Clockify
Agile Tools	Support Agile methodologies like Scrum and Kanban	Rally, Azure DevOps, Scrumwise
Budgeting & Financial Tools	Manage project budgets and financial planning	QuickBooks, Planview, Scoro
Deployment & Integration Tools	Automate CI/CD pipelines and deployment processes	Jenkins, Kubernetes, Docker, Ansible

Techniques in SPM

Technique	Purpose
Work Breakdown Structure (WBS)	Breaking down the project into smaller, manageable tasks.
Critical Path Method (CPM)	Identifying the sequence of tasks that determine the project duration.
PERT (Program Evaluation Review Technique)	Estimating project duration using optimistic, pessimistic, and most likely time estimates.
Agile Methodologies	Frameworks like Scrum, Kanban, and SAFe for iterative development.

Gantt Charts	Visualizing project schedules and timelines.
Earned Value Management (EVM)	Measuring project performance against scope, schedule, and budget.
Kanban Boards	Visualizing task progress in columns (e.g., To Do, In Progress, Done).
Risk Assessment Techniques	Tools like SWOT Analysis, Risk Matrices, and Monte Carlo simulations to evaluate risks.
MoSCoW Prioritization	Categorizing tasks as Must-have, Should-have, Could-have, and Won't-have.
Rapid Prototyping	Building quick prototypes for early feedback.
Continuous Integration and Continuous Deployment (CI/CD)	Automating code integration and deployment to enhance reliability.
Stand-Up Meetings	Short, daily team meetings to review progress and address blockers.
Change Control Processes	Systematically evaluating and approving project changes.
Root Cause Analysis (RCA)	Investigating the cause of problems and preventing recurrence.

Stakeholders: These are people who have a stake or interest in the project. They could be users/clients or developers/implementers.
They could be:

- Within the project team
- Outside the project team, but within the same organization
- Outside both the project team and the organization.

Human Resource Management in SPM:

1. Resource Planning

Resource planning ensures efficient utilization of human and technical assets in a project. It involves:

- **Role Identification:** Clearly defining roles and responsibilities for developers, testers, designers, and other team members.
- **Skill Mapping:** Matching the skills of team members with the project's technical and business requirements.
- **Resource Allocation:** Distributing personnel efficiently across different phases of the project to optimize productivity.

2. Team Building

A strong team is essential for successful project execution. Key aspects of team building include:

- **Hiring and Onboarding:** Recruiting individuals with the necessary technical and soft skills for the project.
- **Training:** Providing training on relevant tools, technologies, and methodologies such as Agile and DevOps.
- **Team Cohesion:** Encouraging collaboration, trust, and a positive work environment among team members.

3. Defining Roles and Responsibilities

Each team member must have a clearly defined role to ensure smooth project execution:

- **Project Manager:** Oversees the project, manages risks, and ensures timely delivery.
- **Business Analyst:** Gathers requirements, analyzes business needs, and communicates them to the team.
- **Developer:** Writes and maintains code based on project specifications.
- **Tester:** Ensures the quality of the software by identifying and fixing defects.
- **UI/UX Designer:** Designs user-friendly and visually appealing interfaces for the software.

Setting clear expectations for each role, including deliverables and performance metrics, ensures accountability.

4. Communication Management

Effective communication is crucial for project success. This involves:

- **Communication Channels:** Using tools like Slack, Microsoft Teams, or email for seamless interaction.
- **Meetings:** Conducting regular team meetings, daily stand-ups, and one-on-one sessions for status updates and issue resolution.

- **Feedback Mechanisms:** Implementing a structured system for continuous performance reviews and improvement suggestions.

5. Conflict Resolution

Conflicts can arise in any project, and addressing them promptly is key to maintaining a productive team environment.

- **Addressing Conflicts Promptly:** Identifying and resolving issues before they escalate.
- **Encouraging Open Communication:** Fostering transparency to resolve misunderstandings and misalignments.

6. Motivation and Leadership

A motivated team performs better and stays engaged throughout the project. Leadership plays a crucial role in maintaining motivation through:

- **Incentives:** Offering bonuses, recognition, flexible work options, or career growth opportunities.
- **Support and Guidance:** Providing emotional and professional support to help team members handle stress and challenges.

7. Time Management

Efficient time management ensures timely delivery of the project. It involves:

- **Scheduling:** Assigning realistic deadlines and workloads to avoid overburdening team members.
- **Prioritization:** Helping the team focus on high-impact tasks that contribute directly to project goals.

8. Retention and Succession Planning

Employee retention is essential to maintain project continuity, while succession planning ensures business continuity.

- **Retention Strategies:** Offering career development opportunities, competitive salaries, and a positive work culture to retain key employees.

- **Succession Planning:** Preparing backups for critical roles to prevent disruptions in case of unexpected attrition.

Challenges in Managing Human Resources

Managing human resources in software projects comes with several challenges, including:

- **Skill Gaps:** Addressing shortages in required technical expertise through training and upskilling.
- **Team Dynamics:** Managing diverse work styles, personalities, and potential conflicts within the team.
- **Remote Collaboration:** Ensuring productivity and seamless communication in geographically distributed teams.
- **Burnout:** Preventing overwork by balancing workloads effectively.
- **Attrition:** Managing the impact of key team members leaving during a critical project phase.

Managing Technical Resources in SPM

Managing technical resources is a critical aspect of **Software Project Management (SPM)** that involves the efficient planning, allocation, and utilization of tools, infrastructure, and technologies to ensure smooth project execution. Proper management of technical assets ensures availability, optimization, and alignment with project goals.

Key Aspects of Technical Resource Management

1. Resource Planning

- Identify existing technical resources and assess their suitability for the project.
- Determine additional resources required based on project requirements.
- Plan and allocate budgets for procuring and maintaining technical resources.

2. Technology Selection

- Choose appropriate **development, testing, deployment, and project management tools** that fit the project's needs.
- Select the best **technologies and frameworks** based on scalability, performance, and compatibility.

3. Infrastructure Management

Managing infrastructure is essential for a smooth software development lifecycle. Key components include:

- **Development Environment:** Setting up workstations, servers, and network configurations for efficient coding.
- **Testing Environment:** Provisioning staging and testing setups that closely mimic production environments.
- **Continuous Integration/Continuous Deployment (CI/CD):** Implementing automated pipelines for streamlined software delivery.

4. Version Control

Using **Git-based repositories** like GitHub or Bitbucket for managing source code ensures:

- Proper tracking of changes.
- Collaboration among developers.
- Prevention of versioning conflicts.

5. Resource Allocation

- Ensure timely procurement and renewal of **software licenses** to avoid disruptions.
- Allocate **sufficient computing resources** (RAM, CPU, storage) to prevent bottlenecks in performance.

6. Performance Monitoring

- Track **technical resource utilization** to prevent overuse or underuse.

- Use **monitoring tools** like **New Relic, AWS CloudWatch**, or **Datadog** for real-time tracking of infrastructure and software health.

7. Security Management

- **Access Control:** Restrict system access based on roles and permissions.
- **Data Protection:** Implement encryption and backups to secure sensitive information.
- **Threat Detection:** Use security tools to identify and mitigate cyber threats.

8. Scalability and Flexibility

- Utilize **cloud services** like AWS, Azure, or Google Cloud to accommodate growing workloads.
- Design infrastructure that can **scale dynamically** with user demand.

9. Training and Documentation

- Train team members on **how to use specific tools and technologies** effectively.
- Maintain **comprehensive documentation** for configurations, troubleshooting, and best practices.

10. Lifecycle Management

- **Procurement to Retirement:** Plan the acquisition, maintenance, and decommissioning of technical assets.
- **Upgrades:** Regularly update tools and technologies to remain compatible with industry standards.

Challenges in Managing Technical Resources

1. **Budget Constraints:** High costs of tools, infrastructure, and software licenses.
2. **Rapid Technological Change:** Keeping up with new trends and evolving industry standards.

3. **Overuse or Underutilization:** Inefficient resource allocation leading to either excessive costs or performance bottlenecks.
4. **Security Risks:** Protecting technical resources from cyber threats, unauthorized access, and data breaches.

Costing and Pricing in Project

- **Costing:**
  It involves calculating all the expenses incurred in delivering the project. These include direct, indirect, fixed, and variable costs.
- **Pricing:**
  It is the process of determining how much to charge the client. It includes not only the costs but also profit margins and market considerations.

Components of Costing

1. Direct Costs: Salaries and wages for developers, testers, designers, and project managers, Software licenses, tools, and hardware, Cloud services or hosting fees.
2. Indirect Costs: Overhead expenses such as office rent, utilities, and administrative support, Training and team-building activities.
3. Variable Costs: Costs that change based on project requirements, like third-party integrations or additional infrastructure.
4. Fixed Costs: Costs that remain constant irrespective of project scope, such as long-term subscriptions.
5. Contingency Costs: Buffer amount to handle unforeseen circumstances like scope changes or resource unavailability.

Pricing Models

- Fixed Price Model: A predetermined price for the entire project.
  - Advantages: Predictability for clients, clear scope.
  - Challenges: Risk of underestimating costs or scope creep.
- Time and Material (T&M) Model: Charges based on the time spent and materials used.
  - Advantages: Flexibility for changes in scope.
  - Challenges: Less predictability for clients.
- Cost-Plus Pricing: Adding a fixed profit margin to the total cost.
  - Advantages: Guarantees profit.

    - Challenges: May not be competitive.
- Value-Based Pricing: Pricing based on the value the software delivers to the client.
    - Advantages: High profit potential if value is significant.
    - Challenges: Requires deep understanding of client needs and outcomes.
- Subscription Model: Regular, recurring payments for ongoing access to software or services.
    - Advantages: Steady revenue stream.
    - Challenges: Initial costs may not be covered immediately.
- Freemium Model: Offering a basic version for free with premium features at a cost.
    - Advantages: Attracts a large user base initially.
    - Challenges: Conversion to paid users may be low.

Factors Influencing Pricing

- ✓ Market Competition: Price competitively based on what competitors are offering.
- ✓ Client Budget: Align pricing with the client's budgetary constraints.
- ✓ Complexity of the Project: Higher complexity warrants higher prices due to increased effort.
- ✓ Technology Stack: Costs vary based on the technology and tools required.
- ✓ Risk Factors: High-risk projects may require premium pricing to account for contingencies.

A business Case

- Introduction/background describes a problem to be solved or an opportunity to be exploited.
- The proposed project:
    - A brief outline of the project scope.
    - The market: The likely demand for the product would need to be assessed.
- Organizational and operational infrastructure:
    - How the organization would need to change.
    - This would be important where a new information system application was being introduced.
- Benefits:

  - These should be express in financial terms where possible. In the end it is up to the client to assess these – as they are going to pay for the project.
- Outline implementation plan:
  - how the project is going to be implemented.
  - This should consider the disruption to an organization that a project might cause.
- Costs: the implementation plan will supply information to establish these.
- Financial analysis: combines costs and benefit data to establish value of project

Training and Development in SPM

Training and development play a vital role in **Software Project Management (SPM)** by ensuring that team members possess the necessary **skills, knowledge, and tools** to execute projects efficiently and adapt to new technologies.

Importance of Training and Development

Effective training and development lead to:

- **Skill Enhancement:** Improves technical and project management capabilities.
- **Project Efficiency:** Enables smoother execution and faster delivery.
- **Adaptability:** Helps teams adjust to evolving technologies and methodologies.
- **Innovation:** Encourages creative problem-solving and the adoption of new techniques.
- **Employee Satisfaction:** Increases motivation, engagement, and retention.

Types of Training in SPM

1. Technical Training

Enhances expertise in programming, database management, cloud computing, and software testing.

- **Examples:**

- Training on tools like **Git, Docker, Kubernetes**
- Learning programming languages such as **Python, Java, JavaScript**

2. Methodology Training

Familiarizes teams with software development methodologies.

- **Examples:**
  - **Agile, Scrum, Kanban**
  - **Waterfall or Hybrid models**

3. Project Management Training

Equips managers with leadership and organizational skills.

- **Examples:**
  - Courses on **PMI, PRINCE2, PMBOK**
  - Training on project management tools like **Jira, Microsoft Project**

4. Soft Skills Training

Develops interpersonal and communication skills for effective collaboration.

- **Examples:**
  - Leadership development
  - Conflict resolution and time management workshops

5. Domain-Specific Training

Helps teams understand industry-specific knowledge relevant to the project.

- **Examples:**
  - **Healthcare software compliance**
  - **Financial technology regulations**

6. Quality Assurance (QA) Training

Improves testing, debugging, and quality control skills.

- **Examples:**
  - Tools like **Selenium, Postman** for automated testing
  - **Test-Driven Development (TDD) practices**

7. Security Training

Enhances cybersecurity awareness and secure coding practices.

- **Examples:**
  - **Secure coding principles**
  - Compliance with **GDPR, ISO standards**

8. Continuous Learning

Encourages ongoing skill development and certification.

- **Examples:**
  - Online courses (**Coursera, Udemy, Pluralsight**)
  - Certifications like **AWS Certified Developer, Microsoft Azure Fundamentals**

Methods of Training and Development

1. **Workshops and Seminars:** Short, focused training sessions on specific topics.
2. **On-the-Job Training:** Learning through real project tasks.
3. **Mentorship Programs:** Pairing junior employees with experienced mentors.
4. **Online Learning Platforms:** Flexible, self-paced e-learning options.
5. **Bootcamps:** Intensive hands-on training in specific technologies.
6. **Conferences and Hackathons:** Exposure to industry trends and networking opportunities.
7. **Self-Paced Learning:** Using documentation, tutorials, and hands-on exercises.

8. **Simulations and Case Studies:** Real-world scenario-based training.

Challenges in Training and Development

- **Time Constraints:** Balancing training with project deadlines.
- **Budget Limitations:** High costs of training programs and certifications.
- **Employee Resistance:** Hesitancy to adopt new skills or methodologies.
- **Rapid Technological Changes:** Keeping training materials updated.
- **Effectiveness Assessment:** Measuring training impact on project success.

Outcomes of Effective Training and Development

✓ Increased productivity and efficiency in project execution.
✓ Reduced risks from skill gaps and errors.
✓ Higher employee engagement and retention.
✓ Competitive advantage through advanced expertise.
✓ Improved project quality and client satisfaction.

Numerical Type topics:

- Cost benefit analysis (CBA)
- Decision Tree

UNIT 2

Monitoring and Measurement in Software Development

Monitoring and measurement are **critical** in software development to **track progress, ensure quality, and meet project goals**.

These practices involve:

- **Tracking key metrics**
- **Analysing data for insights**
- **Improving team performance and product quality**

Why Monitor and Measure?

❖ **Ensure alignment** with project objectives.
❖ **Track progress** and identify deviations early.
❖ **Improve team productivity** and optimize workflows.
❖ **Enhance product quality** through continuous evaluation.
❖ **Define meaningful metrics** based on project needs.

Key Metrics in Software Development

1. Process Metrics (Measures development efficiency)

- **Velocity:** Work completed per sprint/iteration.
- **Cycle Time:** Time taken to complete a task from start to finish.
- **Lead Time:** Total time from task request to delivery.
- **Work in Progress (WIP):** Number of tasks actively being worked on.

2. Product Metrics (Evaluates software performance)

- **Defect Density:** Number of defects per unit of code/functionality.
- **Code Coverage:** Percentage of code covered by automated tests.
- **Technical Debt:** Effort required to fix and refactor problematic code.
- **Mean Time to Failure (MTTF):** Average time before software failure.

3. Quality Metrics (Measures overall software quality)

- **Customer Satisfaction (CSAT):** Feedback and ratings from users.
- **Net Promoter Score (NPS):** Likelihood of users recommending the product.
- **Defect Resolution Time:** Average time taken to fix identified defects.

4. Team Metrics (Assesses team efficiency and well-being)

- **Team Morale:** Surveys and feedback loops to gauge employee satisfaction.
- **Collaboration Metrics:** Measures teamwork effectiveness (e.g., communication frequency, meeting efficiency).

Introduction to Software Estimation

Software estimation is a critical aspect of project planning, involving predicting the time, cost, and effort required for successful completion. However, estimating software projects is challenging due to the complexity and invisibility of software processes.

What Makes a Successful Project?

A project is considered successful when it meets the following criteria:

- **Delivering the agreed functionality**
- **Completing on time**
- **Staying within the agreed cost**
- **Maintaining the required quality standards**

Stages of Estimation

1. **Setting Targets** – Establishing clear goals based on project requirements.
2. **Attempting to Achieve Targets** – Managing resources and timelines to meet set objectives.

Challenges in Software Estimation

1. Subjective Nature

- Small tasks are often underestimated, while large projects tend to be overestimated.
- Lack of a standardized approach can lead to inconsistent estimations.

2. Political Pressures

- Managers may manipulate estimates to make a project appear more feasible.
- Some projects are overestimated to create a buffer or reduce risks.

3. Changing Technologies

- New tools and frameworks introduce uncertainties.
- The learning curve in emerging technologies makes past project experience less reliable.

4. Project Differences

- No two projects are exactly alike, making it difficult to apply past experiences.
- Variations in team skills, project requirements, and technologies impact estimation accuracy.

Basis for successful estimation

- Use historical data: Refer to past projects of similar scope for more accurate estimates.
- Adopt estimation techniques like Top-down or bottom-up approaches; PERT (Program Evaluation and Review Technique);
- Function point analysis: Estimate effort based on software functionalities.
- Involve cross-functional teams: Collaborative estimation brings diverse perspectives and mitigates biases.
- Iterative adjustments: Review and refine estimates as the project progresses and more information becomes available.
- Buffer wisely: Add realistic contingency buffers for known risks but avoid excessive padding.
- Use tools: Leverage project management software to model effort and resource allocation.

- Stakeholder communication: Align expectations early and communicate potential risks transparently.

A Taxonomy of Estimating Methods

Software estimation methods can be classified into various approaches based on how they derive effort, cost, and duration.

1. Bottom-Up Estimation

- **Approach:** Breaks the project into smaller, manageable tasks (Work Breakdown Structure – WBS).
- **Techniques:**
  - Insert, amend, update, display, delete, print (activity-based, analytical).
- **Advantages:**
  - High accuracy due to task-level granularity.
  - Improved accountability by involving teams in estimates.
  - Better risk identification through detailed breakdowns.
- **Disadvantages:**
  - Time-consuming and complex for large projects.
  - Requires well-defined tasks for accuracy.
- **Best for:** Large, well-defined projects with significant historical data.

2. Top-Down Estimation

- **Approach:** Starts with a high-level estimate, which is then distributed among project components.
- **Techniques:**
  - Parametric or algorithmic models.
- **Advantages:**
  - Faster and requires less effort in early project phases.
  - Works well when detailed requirements are unavailable.

  - Useful for feasibility studies and budget approvals.
- **Disadvantages:**
  - Less accurate due to high-level assumptions.
  - Prone to bias and inflexibility if assumptions change.
- **Best for:** Early-stage planning, smaller projects, or when historical data is available.

3. Expert Opinion

- **Approach:** Relies on domain experts to estimate project scope based on experience.
- **Challenges:**
  - Estimates may be subjective or biased.
  - Accuracy depends on expert judgment.

4. Analogy-Based Estimation

- **Approach:** Compares the current project with similar past projects to estimate effort and cost.
- **Advantages:**
  - Leverages historical data for reasonable accuracy.
  - Faster than bottom-up estimation.
- **Disadvantages:**
  - Requires relevant past project data.
  - Differences in project scope can affect accuracy.

5. Function Point Analysis (Albrecht Method)

- **Approach:** Measures project size based on functional components rather than lines of code.
- **Preferred Over SLOC Because:**
  - SLOC lacks a precise definition.
  - Difficult to estimate early in a project.

- Programmer-dependent and ignores code complexity.

Parameters to Be Estimated

1. Project Size

- A fundamental measure of the work required.
- **Metrics:**
  - **Source Lines of Code (SLOC):** Measures the total lines of code written.
  - **Function Points (FP):** Estimates based on software functionality.

FP is favored over SLOC due to:

- Lack of a clear definition for SLOC.
- Difficulty in estimating at the project's start.
- SLOC being only a code measure without considering complexity.

2. Effort

- The total workload required, measured in **person-months**.

3. Duration

- The time required for project completion, measured in **months**.

Empirical Size Estimation Techniques

Empirical estimation techniques rely on prior experience and expert analysis to predict project size, effort, and cost. These techniques involve educated guessing and comparisons with similar past projects.

1. Expert Judgment Technique

- **Approach:**
  - An experienced professional analyzes the project scope and provides an estimate based on their knowledge and experience.
  - Estimates different components separately and combines them to derive the final cost.

- **Advantages:**
  - Quick and easy to implement.
  - Works well when experts have experience with similar projects.
- **Disadvantages:**
  - Prone to **human error and bias**.
  - Highly dependent on expert availability and experience.
  - Lack of historical data can lead to incorrect estimations.

2. Delphi Cost Estimation

- **Approach:**
  - A **team of experts** estimates the project independently.
  - A **coordinator** collects and circulates information without revealing expert identities.
  - Experts refine their estimates over multiple rounds without direct discussion.
- **Process:**
  1. Experts independently estimate the project size, effort, or cost.
  2. Each expert documents **unusual characteristics** affecting their estimate.
  3. The **coordinator compiles responses** and highlights extraordinary justifications.
  4. Experts re-evaluate their estimates considering new insights.
  5. The process is repeated until a **final consensus is reached**.
- **Advantages:**
  - Reduces individual bias as experts do not directly influence each other.
  - More accurate than a single expert's judgment due to multiple perspectives.

- Encourages logical reasoning through multiple iterations.

- **Disadvantages:**
  - Time-consuming due to multiple rounds of estimation.
  - Still relies on expert experience and availability.

Comparison: Expert Judgment vs. Delphi Estimation

Feature	**Expert Judgment**	**Delphi Cost Estimation**
Number of Experts	Single expert	Multiple experts
Bias Reduction	Prone to bias	Minimizes bias through independent reviews
Accuracy	Can be subjective	More reliable due to iterative refinement
Time Required	Quick	Time-consuming
Use Case	Small, less complex projects	Large, complex projects requiring accuracy

Function Point Analysis (FPA),

Function Point Analysis (FPA), developed by **Allan J. Albrecht** at IBM, is a widely used technique for **measuring software size** based on its functionality rather than lines of code. It helps estimate **effort, cost, and time** required for software development.

Function Point Analysis (FPA) – Key Concepts

FPA measures the functional size of a system in **Function Points (FPs)**, which represent the **amount of business functionality** provided to the user.

1. Functional Components

Function points are calculated based on five components:

Component	**Description**
External Inputs (EI)	User inputs that modify system data (e.g., forms, API requests)
External Outputs (EO)	Processed information sent to users (e.g., reports, UI outputs)
External Inquiries (EQ)	User requests for system information (e.g., search queries)
Internal Logical Files (ILF)	Data stored and maintained by the system (e.g., database tables)
External Interface Files (EIF)	Data referenced from external systems but not modified (e.g., APIs, linked databases)

2. Steps in Function Point Calculation

1. **Identify and Classify Functional Components**
   - List all **EIs, EOs, EQs, ILFs, and EIFs** in the system.
2. **Assign Complexity Weights**
   - Each component is categorized as **Low (L), Average (A), or High (H)** based on complexity.
   - Each level has a predefined function point value.

Component	**Low (L)**	**Average (A)**	**High (H)**
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

3. Calculate Unadjusted Function Points (UFP)

$$\text{UFP} = \sum(\text{Component Count} \times \text{Complexity Weight})$$

4. Apply Value Adjustment Factor (VAF)
   - Adjust function points based on 14 **general system characteristics (GSCs)** like performance, security, reusability, etc.
   - VAF is calculated as:

$$VAF = 0.65 + (0.01 \times \sum GSCs)$$

   - GSCs are rated from **0 (no impact) to 5 (strong impact)**.

3. Applications of Function Points in SPM

- **Effort Estimation**: Helps predict developer hours required.
- **Cost Estimation**: Converts effort into project cost.
- **Project Productivity Measurement**: Helps track team performance over time.
- **Comparison of Different Technologies**: Measures software productivity across languages (e.g., Java vs. Python).

Example Calculation

Assume a project has:

- 4 **EI** (Avg), 3 **EO** (Low), 2 **EQ** (High), 2 **ILF** (Avg), and 1 **EIF** (Low).

Step 1: Assign Weights

Component	Count	Complexity	FP Value	Total FP
EI	4	Avg (4)	4 × 4	**16**
EO	3	Low (4)	3 × 4	**12**
EQ	2	High (6)	2 × 6	**12**
ILF	2	Avg (10)	2 × 10	**20**
EIF	1	Low (5)	1 × 5	**5**
Total UFP				**65**

Step 2: Apply Value Adjustment Factor (VAF)

- Assume **GSC total score = 30** VAF=0.65+(0.01×30)=0.95VAF = 0.65 + (0.01 \times 30) = 0.95

Step 3: Final Function Points

FP=65×0.95=61.75≈62FP = 65 \times 0.95 = 61.75 = 62

Function Point Analysis - Mark II (MK II FPA)

Function Points Mark II (MK II FPA) is an improved version of the original **Function Point Analysis (FPA)** by **Allan J. Albrecht**. It was developed in the late 1980s by **Charles Symons** to address some limitations of the original method, particularly in **real-time systems and object-oriented development**.

Key Differences Between MK II and Albrecht's Function Points

Feature	**Albrecht's FPA**	**MK II Function Points**
Complexity Levels	3 (Low, Medium, High)	1 (No complexity weighting)
Function Types	5 (EI, EO, EQ, ILF, EIF)	3 (Input, Output, Entity)
Adjustment Factors	14 general system characteristics (GSCs)	None (avoids subjectivity)
Units of Measurement	**Unadjusted FP** (then adjusted using VAF)	**Mk II FP (directly measured)**
Applicability	Best for business applications	Better for real-time & object-oriented systems
Calculation Complexity	More detailed but subjective	Simpler and more consistent

Key Components of MK II Function Point Analysis

Instead of five functional components like in **Albrecht's FPA**, **MK II FPA** focuses on just **three function types**:

1. **Inputs (I)** – Data or control information entering the system, requiring validation or processing (e.g., form submissions, API requests).
2. **Outputs (O)** – Information leaving the system, including reports, screens, or control signals.
3. **Entity Types (E)** – Persistent data maintained by the system (e.g., database tables, files).

Each function is counted **without complexity weighting**, making the method simpler and more objective.

MK II Function Point Calculation Steps

1. **Identify Functional Transactions**
   - Categorize system functions into **Inputs (I), Outputs (O), and Entities (E)**.
2. **Count the Function Points**
   - Assign predefined weightings:
     - **Inputs (I) = 0.58 FP per input**
     - **Outputs (O) = 1.66 FP per output**
     - **Entities (E) = 1.00 FP per entity**
3. **Sum the Function Points**

$$FP=(0.58\times Inputs)+(1.66\times Outputs)+(1.00\times Entities)$$

4. **Obtain Final MK II Function Points**
   - No need for a value adjustment factor (VAF), as the raw count provides the function size.

Example Calculation (MK II FPA)

Assume a system has:

- **8 Inputs (I)**
- **5 Outputs (O)**
- **3 Entity Types (E)**

Using MK II weighting:

FP=(0.58×8)+(1.66×5)+(1.00×3)

FP=4.64+8.3+3FP = 4.64 + 8.3 + 3

FP=4.64+8.3+3 FP=15.94≈16 (rounded)

FP = 15.94 = approx 16

Thus, the system size is **16 MK II Function Points**.

Advantages of MK II Function Points

- **Simpler & More Objective** – No complexity weights or VAF, reducing subjectivity.
- **Better for Real-Time & OO Systems** – Works well for modern software architectures.
- **Consistent Measurement** – Fixed function weights make it easier to compare systems.
- **Better Cost & Effort Estimation** – Provides a more stable metric for project estimation.

Other Numerical topics:

- COSMIC full function points
- COCOMO