1. Software Project Management: An Introduction

An Introduction to the Discipline

AG



The Nature of Software Projects Unique Characteristics of Software Projects



Intangible Product: Difficult to visualize and measure progress.



High Complexity: Millions of lines of code, intricate logic, numerous integrations.



Flexibility & Change: Requirements are rarely static; adaptation is key



Further Software Project Characteristics

More on Software Project Uniqueness

People-Intensive

- Success heavily relies on human intellect, creativity, and collaboration.

Uniqueness

- Most projects are bespoke; less 'copy-pasting'.

High Failure Rate (Historically)

- Common pitfalls: poor planning, scope creep, communication.

Rapid Technological Change

- Tools, languages, and methodologies evolve quickly.

Objectives of Software Projects

Defining Software Project Success: Beyond the Basics



The "Triple Constraint":

Deliver **High-Quality Software** (Reliable, Efficient, Maintainable)

Within Budget (Cost-effective)

On Time (Schedule adherence)



Expanded Objectives:

Meet Stakeholder Requirements: Satisfy all involved parties.

Achieve Business Value: Contribute to organizational goals.

Manage Risk: Proactive identification & mitigation.



Ensure User Satisfaction: The ultimate measure of success

Stakeholders in Software Projects

Who's Involved? Identifying Project Stakeholders



Anyone affected by, or who can affect, the project.



Key Internal Stakeholders:

- Customers/Users
- Project Manager
- Development Team
- Sponsors
- Senior Management
- Functional Managers



Key External Stakeholders:

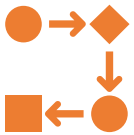
- Regulators/Government
- Suppliers/Vendors

Project Initiation and Justification



AG

Introduction to Project Initiation
Is This Project Worth It?



Purpose: The critical first step – deciding if a project should proceed.



Key Question: "Should we build this software?"



Core Activities:

Defining the project at a high level.

Conducting a Feasibility Study.

Performing Cost-Benefit Analysis



Goal: Avoid investing in unviable or unprofitable projects.

Feasibility Study – Overview

Feasibility Study: Assessing Project Viability



Definition: Systematic evaluation of a proposed project's likelihood of success.



Purpose: To determine if a project is realistic, achievable, and sensible.



Key Question: "Can we build this, and should we?"

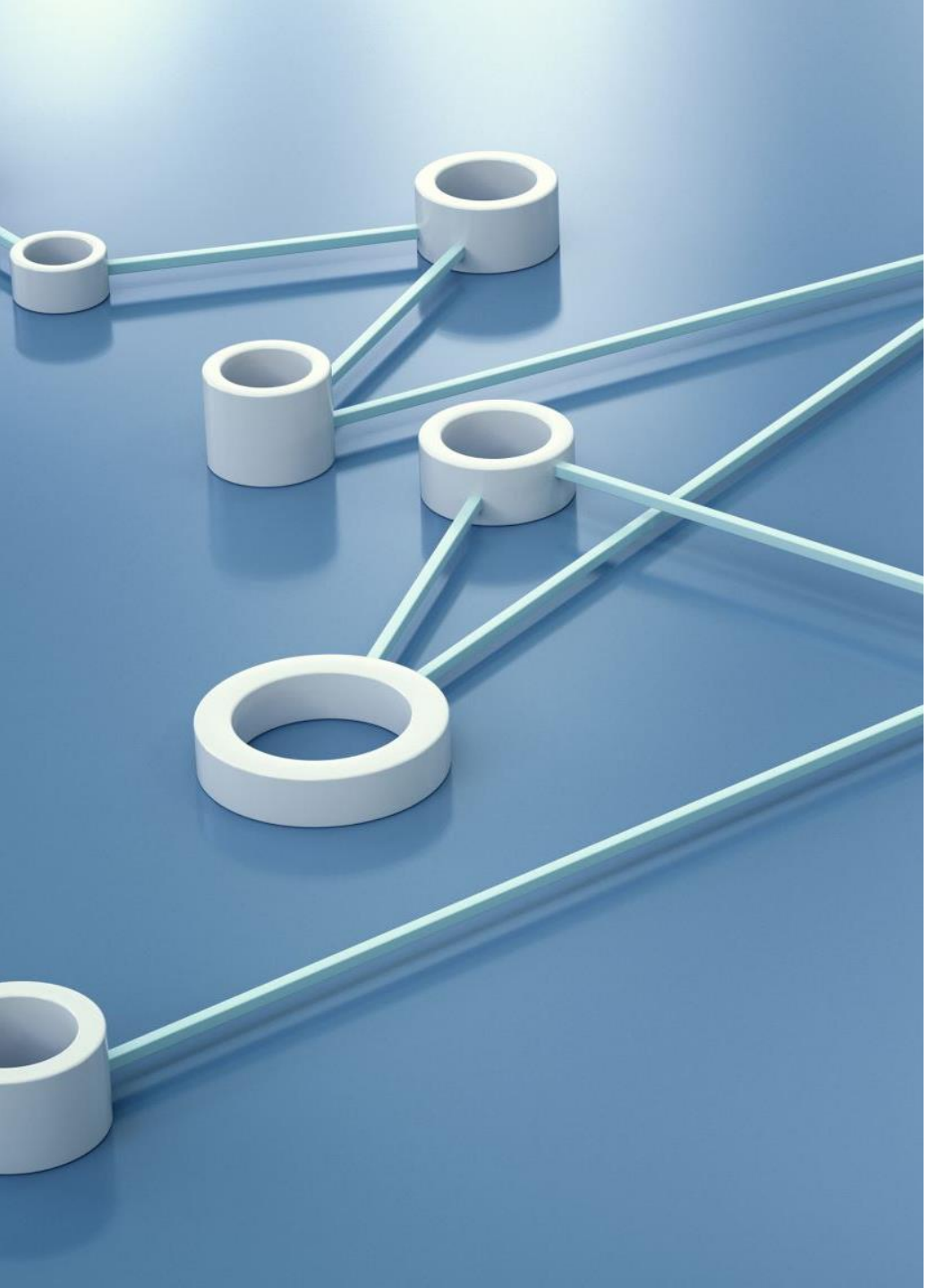


Benefits:

Reduces risk of failure.

Saves time and money.

Provides a basis for decision-making.



Feasibility Study - Key Aspects

Dimensions of Feasibility

Technical Feasibility	• Do we have the technology, infrastructure, and expertise?
Economic Feasibility	• Is it financially viable? (Cost-Benefit Analysis).
Operational Feasibility	• Will it integrate well and be used by the organization?
Schedule Feasibility	• Can it be completed within the required timeframe?
Legal & Ethical Feasibility	• Does it comply with laws, regulations, and ethics?

Feasibility Study - Output & Decision

The Feasibility Report: Go/No-Go Decision

Report Content:

- Summary of findings.
- Detailed analysis for each feasibility type.
- Recommendations.

Possible Recommendations:

- **Proceed:** Project is viable and beneficial.
- **Revise:** Project has potential but needs significant changes.
- **Abandon:** Project is not viable or worth the investment.

Cost-Benefit Analysis – Overview
CBA: The Financial Justification



PURPOSE: TO EVALUATE THE FINANCIAL ATTRACTIVENESS OF A PROJECT.



METHOD: COMPARING TOTAL ESTIMATED COSTS WITH TOTAL ESTIMATED BENEFITS.



GOAL: DETERMINE IF BENEFITS OUTWEIGH COSTS.



ROLE: CORE COMPONENT OF ECONOMIC FEASIBILITY.

Cost-Benefit Analysis - Cost Types

Understanding Project Costs



Direct Costs: Directly attributable to the project.

Examples: Developer salaries, software licenses, dedicated hardware.



Indirect Costs: Not directly tied, but project consumes a share.

Examples: Office rent, utilities, general IT support, administrative overhead.



Tangible Costs: Easy to measure and assign monetary value.

Examples: Equipment purchase, cloud service subscriptions.



Intangible Costs: Real costs, but difficult to quantify monetarily.

Examples: Loss of morale, negative reputation, productivity loss during transition.

Cost-Benefit Analysis - Benefit Types & Techniques

Understanding Project Benefits & Evaluation Techniques

Benefit Types:

Tangible Benefits: Measurable financial gains.

- Examples: Increased revenue, reduced operational costs, cost avoidance.

Intangible Benefits: Non-monetary advantages.

- Examples: Improved customer satisfaction, enhanced decision-making, competitive advantage, improved morale.

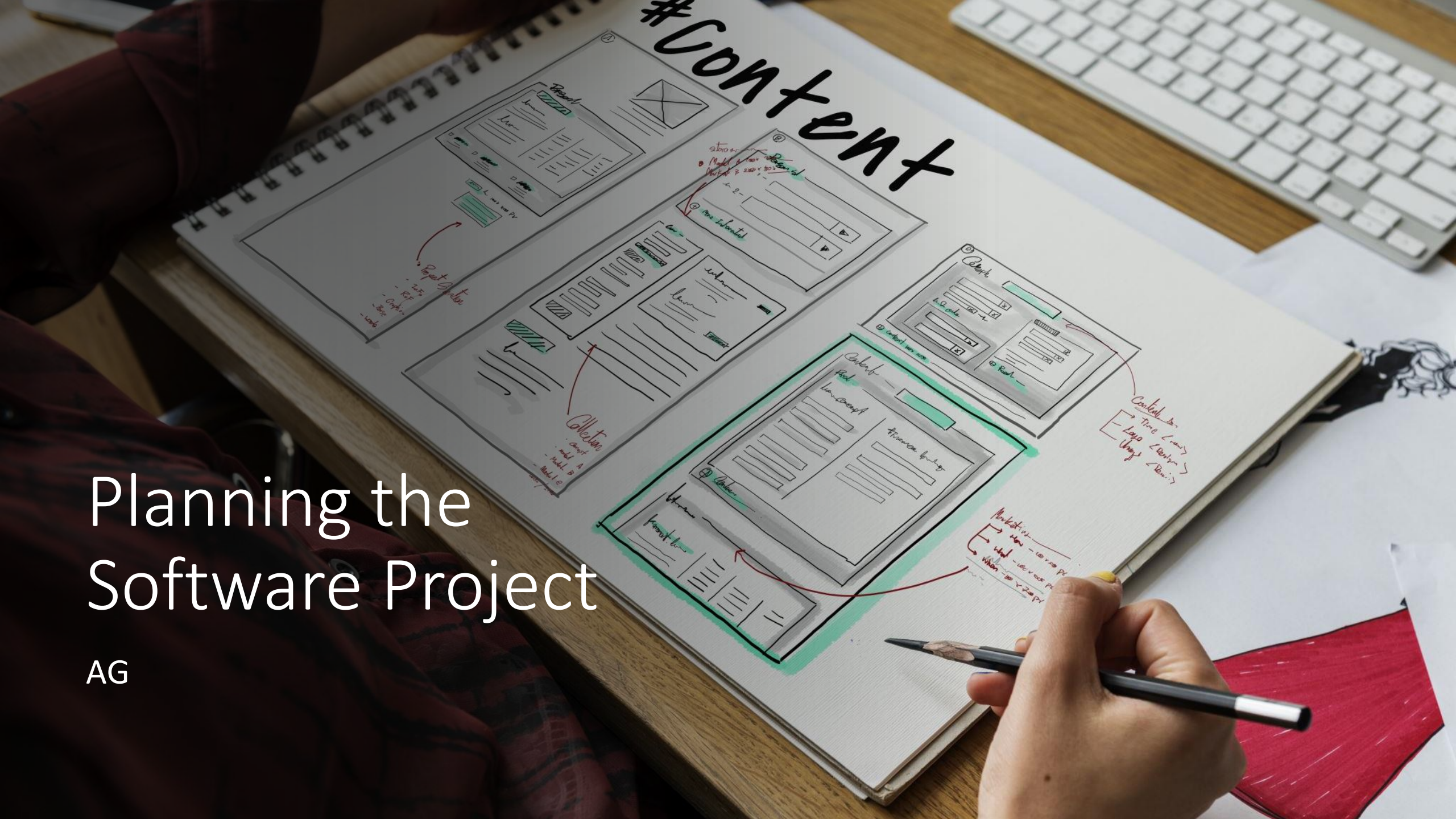
CBA Techniques:

Return on Investment (ROI): Simple percentage.

Payback Period: Time to recoup investment.

Net Present Value (NPV): Accounts for time value of money.





Planning the Software Project

AG

Introduction to Project Planning

The Blueprint for Success: Project Planning

Definition

Defining project goals, objectives, and outlining the path to achieve them.

Key Question

"How are we going to build this software?"

Importance:

Reduces uncertainty.

Provides direction and clarity.

Facilitates coordination.

Nature

An iterative and evolving process.



Defining Project Scope
Scope Definition: Setting the Boundaries



Definition

Clearly defining what the project will deliver (in-scope) and what it will not (out-of-scope).



Purpose

To establish clear boundaries and prevent "scope creep."



Key Document

Scope Statement (or Project Charter).



Challenge

"Scope Creep" – uncontrolled addition of features.

Requirement Gathering
Understanding "What" to Build

Purpose: Elicit, analyze, document, and manage stakeholder needs.

Types of Requirements:

Functional: What the system *does*.

Non-Functional: How the system *performs* (e.g., speed, security, usability).

Techniques: Interviews, workshops, surveys, prototyping, use cases/user stories.

Quality: Requirements must be Clear, Complete, Consistent, Unambiguous, Testable.



Work Breakdown Structure (WBS) & Estimation

Decomposing Work & Predicting Effort

Work Breakdown Structure (WBS):

- Hierarchical decomposition of project scope into smaller, manageable tasks.
- Deliverable-oriented.
- Benefits: Improved estimation, clear assignments, better tracking.

Estimation:

- Predicting effort, cost, and duration.
- Challenges: Uniqueness, unforeseen issues.
- Techniques: Expert Judgment, Analogous, Parametric, Three-Point (PERT), Bottom-Up.

Scheduling & Resource Planning
Timeline & Resource Allocation

Scheduling:

Arranging estimated tasks logically over time.

Gantt Charts: Visualizing tasks, durations, dependencies.

Critical Path Method (CPM): Identify longest path, determines minimum project duration.

Milestones: Key progress points.

Resource Planning:

Identify and allocate human (skills, roles) and non-human (hardware, software, tools) resources.

Ensure availability and optimal utilization.



Risk & Communication Planning
Managing Uncertainty & Information Flow

Risk Management Planning:

Proactively identify potential risks.

Analyze likelihood and impact.

Develop mitigation and contingency strategies.

Communication Planning:

Define who needs what information, when, and how.

Establish communication channels and frequency.

Ensures effective information exchange among stakeholders.



Project Executing and Life Cycles



AG

Introduction to Project Execution
Bringing the Plan to Life

Definition: The phase where the project plan is put into action.

Core Activity: The actual work of creating the software.

Nature: Dynamic, problem-solving, collaboration-intensive.

PM's Role: Guiding, monitoring, enabling the team.

Team & Task Management in Execution
Mobilizing Teams & Assigning Work

1

Team Mobilization:

- Assemble and onboard team members.
- Ensure tools, access, and understanding of goals.
- Foster team cohesion and positive environment.

2

Task Assignment:

- Distribute work packages/activities from WBS.
- Ensure clear responsibilities and deadlines.
- Balance workload and leverage team skills.

Development & Quality Assurance

Building & Ensuring Quality

Development & Coding:

Writing code, designing UIs, building databases, integrating components.

Often iterative (e.g., Agile sprints).

Quality Assurance (QA):

Integrated throughout development, not just at the end.

Purpose: Ensure software meets quality standards & requirements.

Activities: Unit, Integration, System, User Acceptance Testing (UAT), Code Reviews.



Managing During Execution
Communication, Issue & Change Management

Communication:	Issue Resolution:	Change Management:	Resource Management:
• Constant internal (team) and external (stakeholder) updates. • PM as central information hub.	• Address problems, blockers, and conflicts as they arise. • Facilitate solutions, remove obstacles, escalate when necessary.	• Formal process for handling changes to scope, schedule, budget. • Prevents uncontrolled "scope creep."	• Monitoring resource utilization, preventing burnout, ensuring support.

Software Project Life Cycle (SPLC)
The Journey of a Software Project



Phases a software project typically moves through.



Phases:

Initiation: Define project, feasibility, authorization.

Planning: Detailed scope, schedule, budget, resources.

Execution: Build the software, perform tasks.

Monitoring & Control: Track progress, manage changes (concurrent with Execution).

Closure: Formal acceptance, handover, lessons learned.



Nature: Temporary endeavor with a definite start and end.

Software Product Life Cycle (SPLC)

The Evolution of a Software Product

The journey of a software product from idea to retirement.

Phases:

Conception/Definition: Idea generation, market research, high-level requirements.

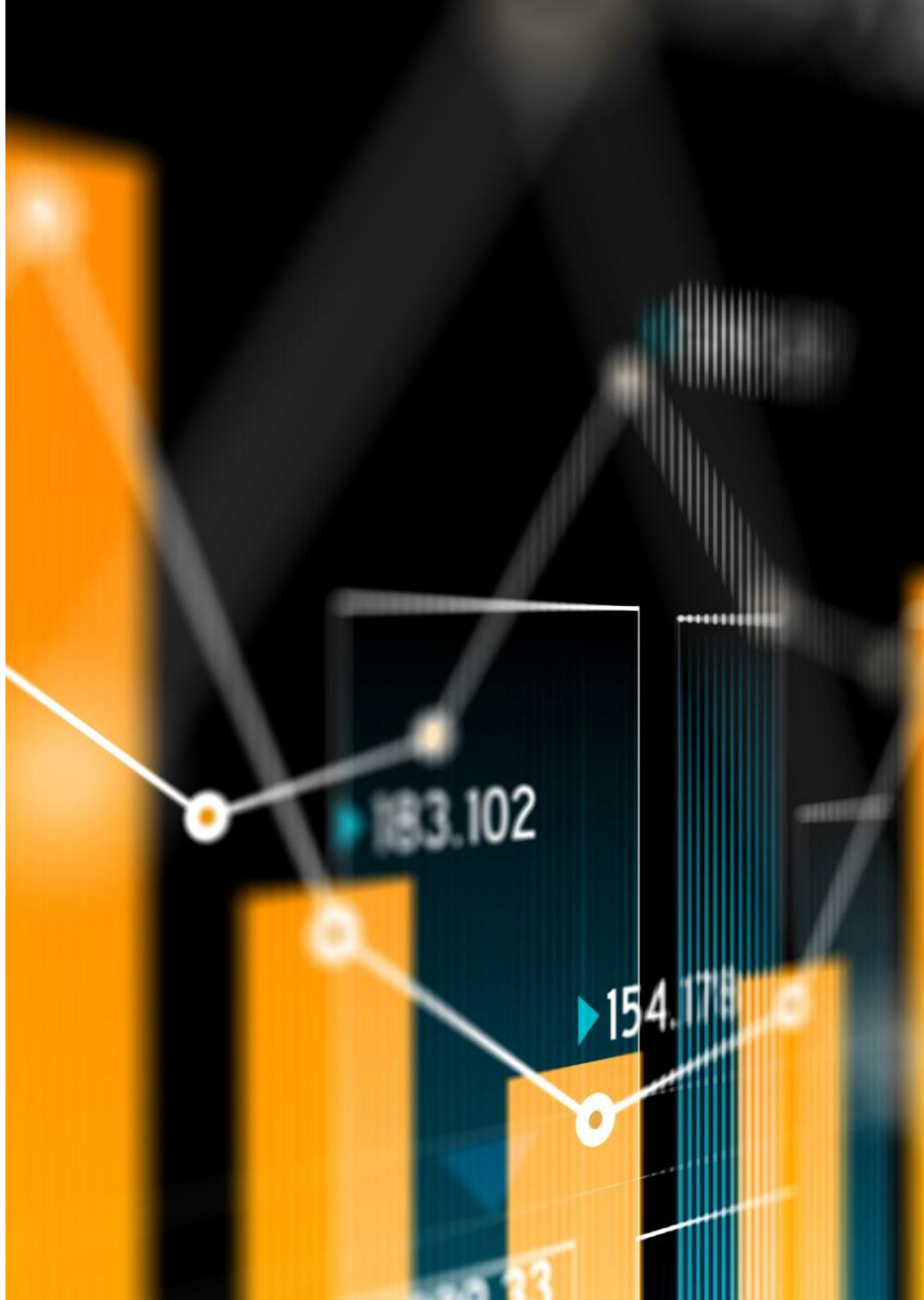
Development: Building the initial product (often a *project*).

Growth/Deployment: Launch, initial user adoption, early feedback.

Maturity: Widespread use, maintenance, minor enhancements.

Decline: Phasing out, replacement, retirement.

Relationship: A product can involve multiple projects over its lifetime.



The Project Manager's Role and Knowledge



AG

Introduction to the Project Manager's Role

The Project Manager: The Heartbeat of the Project



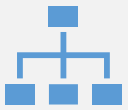
Central Figure: Leads, guides, and manages the project.



Multifaceted Role: Requires technical, business, and soft skills.



Beyond Technical: Focus on facilitation, communication, leadership.



Ultimate Accountability: Responsible for project success or failure.

Key Responsibilities of a Project Manager

Core Responsibilities of the PM



Advanced Responsibilities of a Project Manager

Beyond the Core: PM's Broader Impact





Problem Solving: Addressing issues, removing roadblocks, facilitating solutions.



Negotiation: Resolving conflicts, reaching agreements (resources, deadlines, contracts).



Stakeholder Management: Managing expectations, fostering relationships, gaining buy-in.



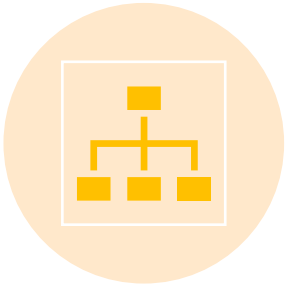
Quality Assurance Oversight: Ensuring quality processes are followed and standards met.



Team Building: Fostering a collaborative, trusting, and productive environment.

Introduction to PMBOK Knowledge Areas

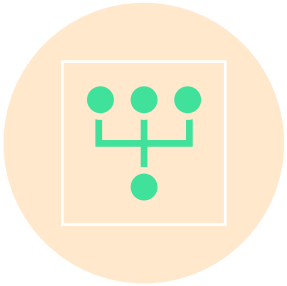
The PM's Toolkit: PMBOK Knowledge Areas



PMBOK Guide: Global standard by Project Management Institute (PMI).



10 Knowledge Areas: Domains of project management expertise.



Interconnected: Not silos, but highly integrated and overlapping.



Purpose: Provide a comprehensive framework for managing projects

Core PMBOK Knowledge Areas (Part 1)

PMBOK: Managing Integration, Scope, & Schedule

1. Project Integration Management:

Coordinating all aspects; the 'glue'.

Ensures all project elements work together.

2. Project Scope Management:

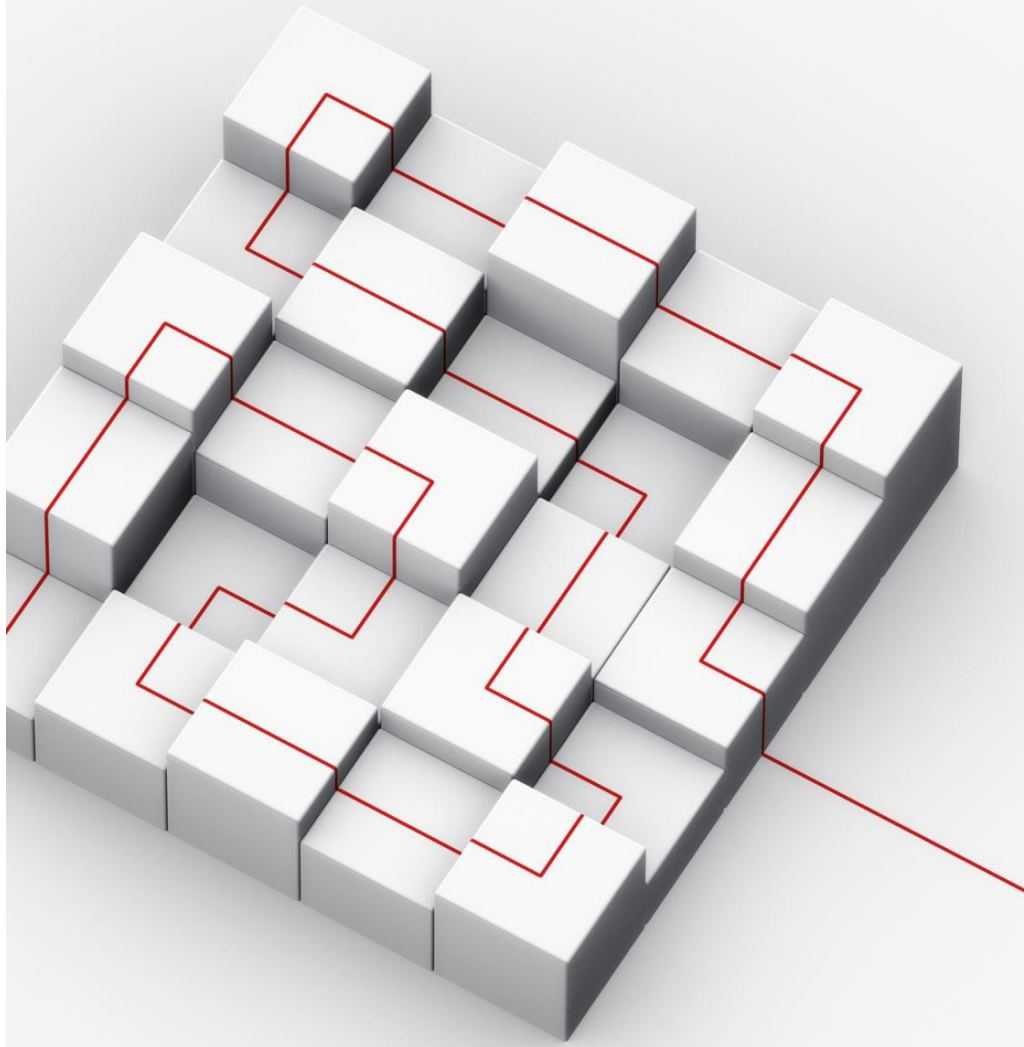
Define & control what's in/out of the project.

Manage requirements, prevent scope creep.

3. Project Schedule Management:

Manage timely completion.

Activities: sequencing, duration estimation, schedule development, control.





Core PMBOK Knowledge Areas (Part 2)

PMBOK: Managing Cost, Quality, & Resources

Project	Project	Project
Project Cost Management: • Plan, estimate, budget, and control project costs. • Ensure project stays within financial limits.	**Project Quality Management:** • Ensure project and product meet defined quality standards. • Activities: planning, assurance, control.	**Project Resource Management:** • Acquire, manage, and utilize team and physical resources. • Ensure right people/tools are available.

Core PMBOK Knowledge Areas (Part 3)

PMBOK: Managing Communications, Risk, Procurement, & Stakeholders



Project

Project Communications Management:

- Plan, execute, and monitor effective information exchange.
- Ensure timely and appropriate information flow.



Project

Project Risk Management:

- Identify, analyze, and respond to project risks.
- Minimize negative impacts, maximize opportunities.



Project

Project Procurement Management:

- Acquire goods and services from outside the organization.
- Manage contracts and vendor relationships.



Project

Project Stakeholder Management:

- Identify all stakeholders, manage expectations, foster engagement.
- Ensure their needs are addressed.



Tools, Techniques, and System View

AG

Introduction to Project Management Tools

Practical Aids: Tools & Techniques for SPM



Purpose of Tools: Automate, standardize, facilitate project management.



Empowerment: Enhance efficiency, visibility, communication.



Selection: Depends on project size, methodology, team preference.



Focus: How tools support planning, execution, quality, and analysis.

Planning & Management Software

Tools for Planning & Overall Management

Planning Tools (Conceptual & Software-Aided):

- Work Breakdown Structure (WBS) creation.
- Gantt Charts & PERT/CPM for scheduling.
- Estimation Techniques (software can aid calculations).

Project Management Software:

- Comprehensive platforms for task tracking, progress, resource allocation.
- Examples: Jira, Asana, Trello, Microsoft Project.
- Features: Dashboards, reports, task assignment, dependencies.



Collaboration & Documentation Tools for Team Collaboration & Knowledge Management

Version Control Systems:

Manage changes to source code and other files.

Enable collaborative development.

Examples: Git (GitHub, GitLab, Bitbucket).

Communication Platforms:

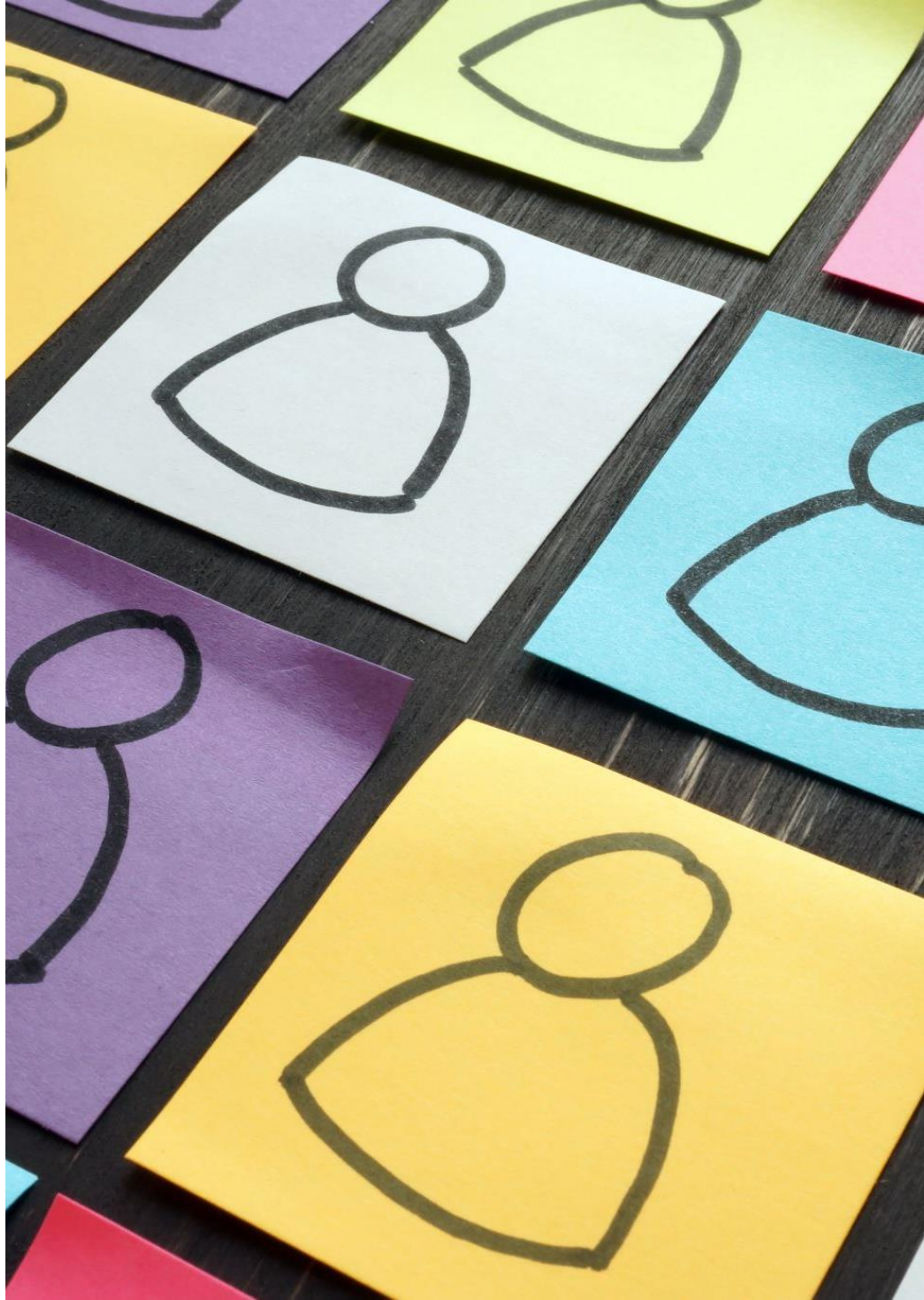
Facilitate real-time chat, file sharing, video calls.

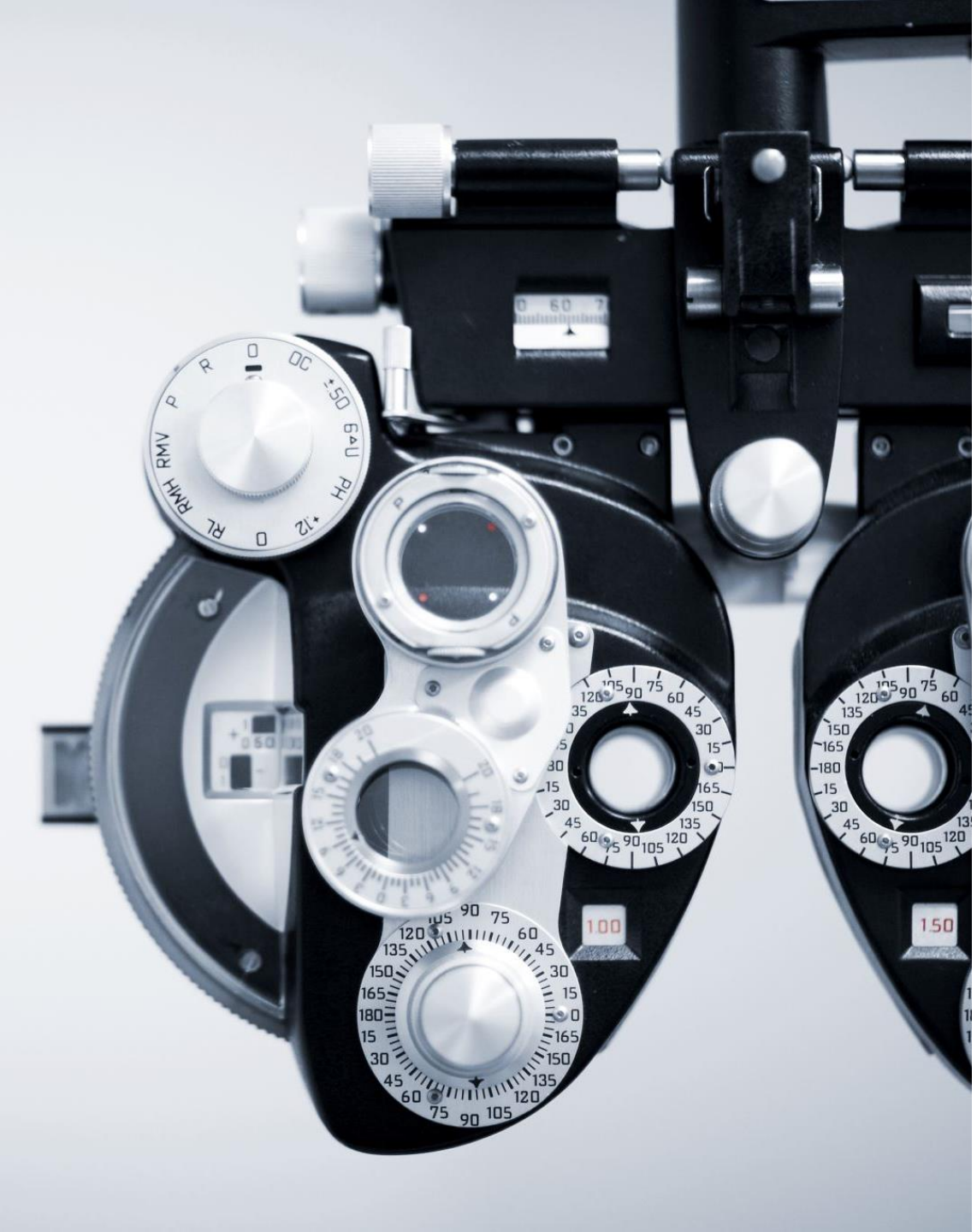
Examples: Slack, Microsoft Teams, Google Chat.

Document Management Systems:

Organize and share project documentation.

Examples: Confluence, SharePoint, Google Drive.





Quality, Testing & Analysis Tools for Quality, Testing & Design

Quality & Testing Tools:

- Unit Testing Frameworks (e.g., JUnit, Pytest).
- Integration Testing Tools.
- Automated Testing Tools (e.g., Selenium, Cypress).
- Bug Tracking Systems (e.g., Jira, Bugzilla).

Analysis & Design Tools:

- UML (Unified Modeling Language) tools.
- Data Flow Diagrams (DFDs).
- CASE Tools (Computer-Aided Software Engineering).

The System View of Project Management – Overview

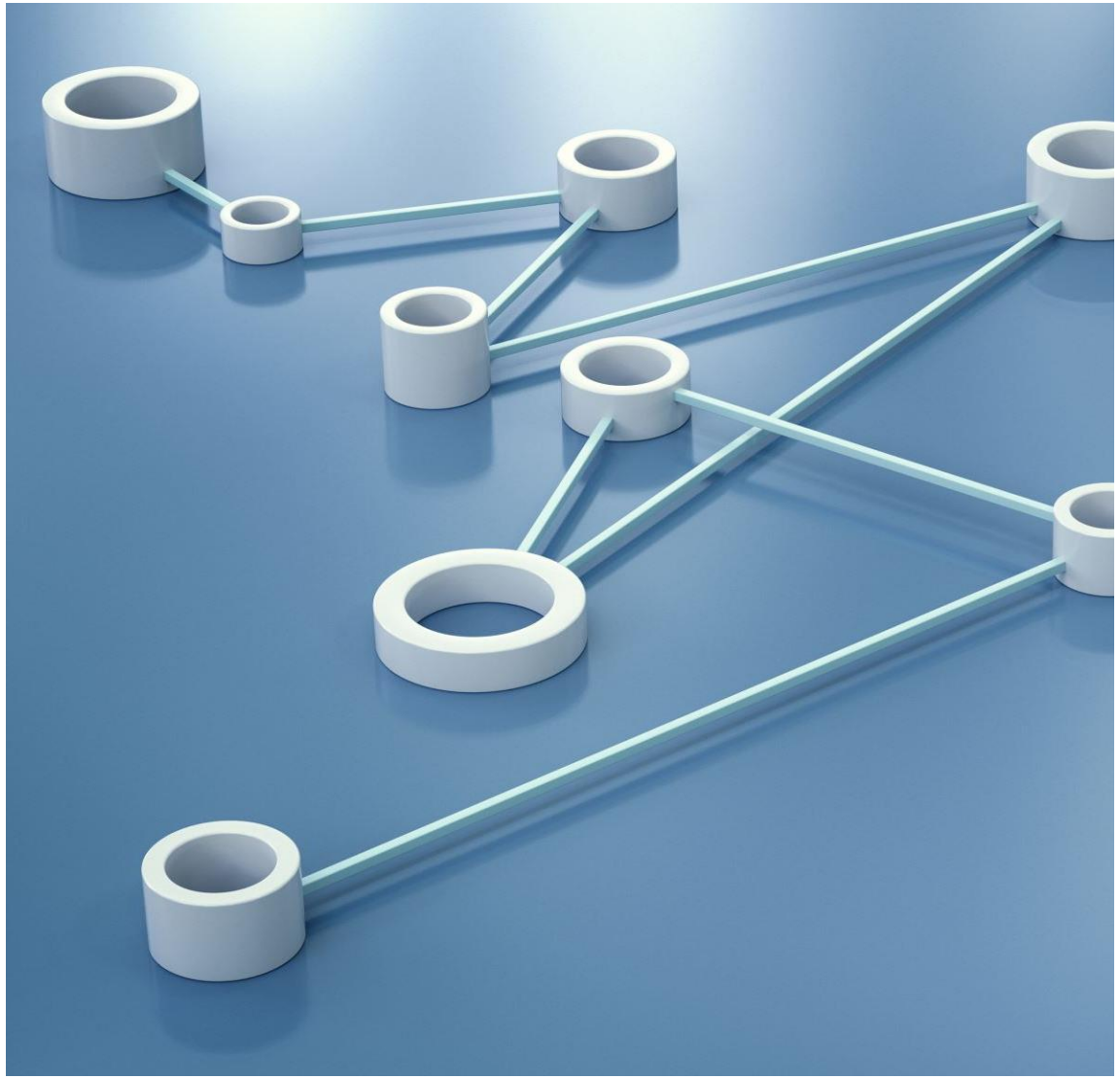
The Big Picture: System View of Project Management

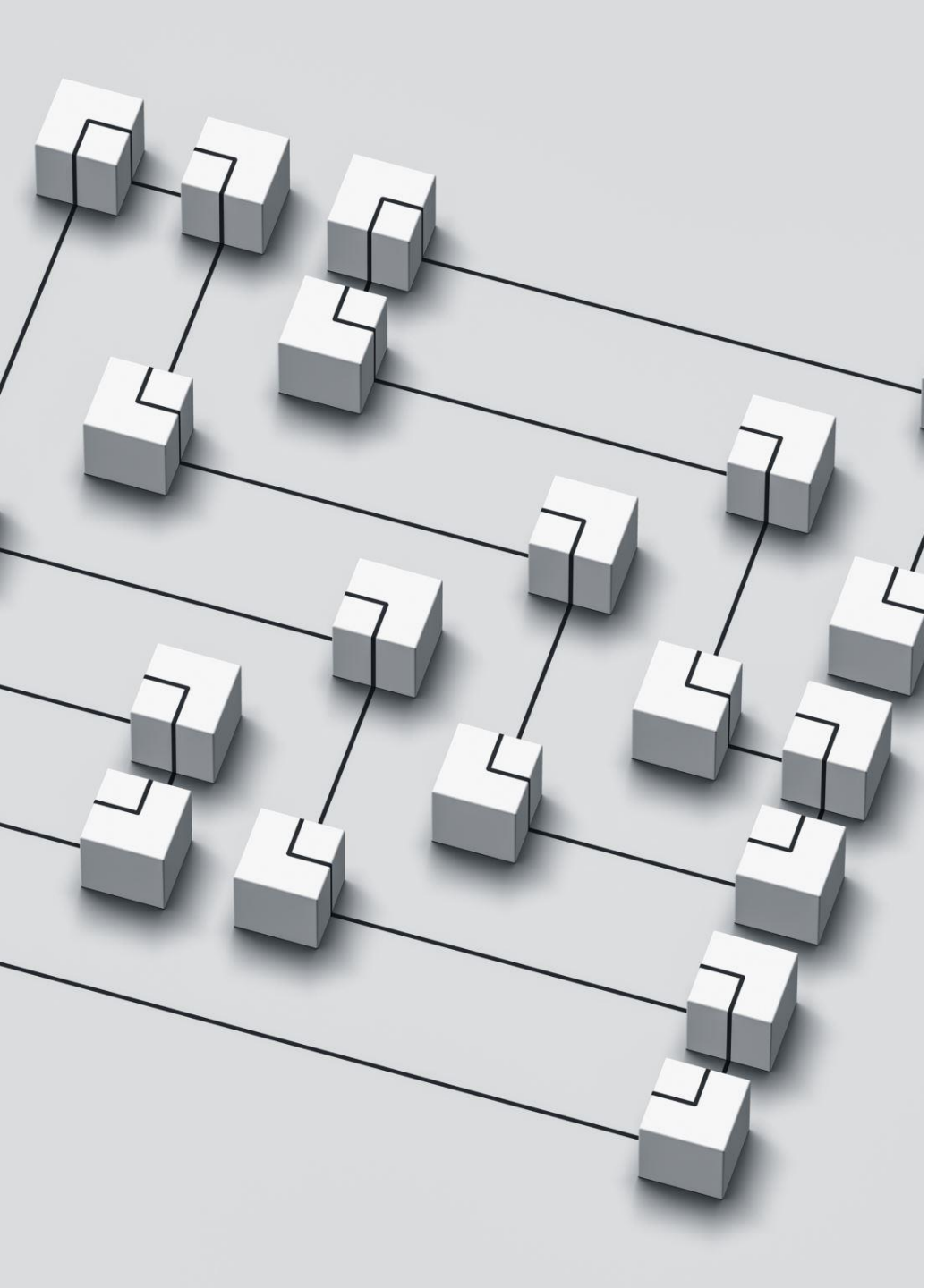
BEYOND THE PROJECT BUBBLE: NO PROJECT EXISTS IN ISOLATION.

PROJECTS AS OPEN SYSTEMS: INTERACT WITH THEIR ENVIRONMENT.

INTERCONNECTED COMPONENTS: INTERNAL PROJECT ELEMENTS ARE INTERDEPENDENT.

HOLISTIC APPROACH: CONSIDER THE PROJECT WITHIN THE BROADER ORGANIZATIONAL AND EXTERNAL CONTEXT.





System View: Inputs, Outputs, Feedback & Environment

System Dynamics: Inputs, Outputs, Feedback & Environment

Inputs: Resources, requirements, organizational policies, market data.

Process: Project management activities (planning, executing, controlling).

Outputs: Deliverables, lessons learned, improved capabilities.

Feedback Loops: Continuous monitoring, user feedback, post-mortems.

Impact of Environment: External factors (economic, political, technological) influence success.

Software Requirement specification

Table of Content

- What is SRS
- Nature of SRS
- Role of SRS
- Characteristics of good SRS
- Template
- SRS template description

What is SRS

- The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

Nature of SRS

The basic issues that SRS writer shall address are the following:

- **1. Functionality:** What the software is supposed to do?
- **2. External Interfaces:** How does the software interact with people, system's hardware, other hardware and other software?
- **3. Performance:** What is the speed, availability, response time, recovery time etc.
- **4. Attributes:** What are the considerations for portability, correctness, maintainability, security, reliability etc.
- **5. Design Constraints Imposed on an Implementation:** Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment etc.

Role of SRS

Since SRS has a specific role to play in software development process, SRS writer should be careful for following points :

- **1.** SRS should correctly define all the software requirements. A software requirement may exist because of the nature of the task to be solved or because of a specific characteristic of the project.
- **2.** SRS should not describe any design or implementation details. These should be described in the design stage of the project.
- **3.** SRS should not impose additional constraints on the software. These are properly specified in other documents such as software quality assurance plan.

Characteristics of good SRS

Following are the characteristics of a good SRS document:

- **Correctness:**
  User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.
- **Completeness:**
  Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.
- **Consistency:**
  Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.
- **Unambiguousness:**
  A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

- **Ranking for importance and stability:**
  There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.
- **Modifiability:**
  SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.
- **Verifiability:**
  A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement starting that the system must be user-friendly is not verifiable and listing such requirements should be avoided.
- **Traceability:**
  One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the

- **Design Independence:**
  There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.
- **Testability:**
  A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.
- **Understandable by the customer:**
  An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.
- **Right level of abstraction:**
  If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

SRS TEMPLATE

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definition, acronyms and abbreviations.
   - 1.4 References
   - 1.5 Overview
2. The overall description
   - 2.1 Product perspective
     - 2.1.1 System interface
     - 2.1.2 Interface
     - 2.2.3 Hardware interface
     - 2.1.4 Communications interface
     - 2.1.5 Memory constraints
     - 2.1.6 Operations
     - 2.1.7 Site adaption requirements

3.Specific Requirements

3.1 External interface

3.2 Functions

3.3 Performance requirements

3.4 Logical database requirements

3.5 Design constraints

3.5.1 Standards compliance

3.6 Software system attribute

3.6.1 Reliability

3.6.2 Availability

3.6.3 Security

3.6.4 Maintainability

3.6.5 Portability

3.7 Organising the specific requirements

3.7.1 System mode

3.7.2 User class

3.7.3 Objects
3.7.4 Features
3.7.5 Features
3.7.6 Response
3.7.7 Functional hierarchy
3.8 Additional comments
4. Change managements process
5. Documents approvals
6. Supporting information

Description

1. Introduction

1.1 **Purpose** → Identify the product whose software requirements are specified in this document, including the revision or release number.

1.2 **Document Conventions** → Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance.

1.3 Scope → . Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it

1.4 Reference → List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, etc.

2. Overall Description

2.1 Product Perspective → Describe the context and origin of the product being specified in this SRS.

2.2 **Product Functions** → Summarize the major functions the product must perform or must let the user perform.

2.3 User Classes and Characteristics → Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience.

2.4 Operating Environment → Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications.

2.5 Design and Implementation Constraints → Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies**.**

2.6 Assumptions and Dependencies → List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints.

3. External Interface Requirements

3.1 User Interfaces → Describe the logical characteristics of each interface between the software product and the users.

3.2 Hardware Interfaces → Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.

3.3 Software Interfaces → Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components.

3.4 Communications Interfaces → Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on.

4. System Features

This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product.

You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.

5. Other Nonfunctional Requirements

If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices.

Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product.

Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product.

6. Other Requirements

Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project**.**

Case Study

- **Software Requirements Specification (SRS)**
- **Project Title:** Online Examination System
  Version: 1.0
  Date: 20 August 2025
  Prepared by: [Your Name / Team]

- **Introduction**
- **1.1 Purpose**
- The purpose of this document is to define the requirements for the Online Examination System (OES). It will describe the system’s functionality, performance, and constraints to ensure clarity among stakeholders, developers, and testers.
- **1.2 Scope**
- The Online Examination System allows educational institutions to conduct exams digitally. The system provides:
- Registration of students and examiners.
- Creation, scheduling, and management of examinations.
- Automatic grading for objective-type questions.
- Manual grading for subjective-type questions.
- Reporting and analytics for administrators.
- This system aims to reduce paperwork, minimize errors, and provide instant results.

- **Definitions, Acronyms, and Abbreviations**
- **OES** – Online Examination System
- **MCQ** – Multiple Choice Questions
- **DBMS** – Database Management System

Product Perspective

- The OES is a web-based application that integrates with existing student information systems. It will follow a modular architecture consisting of:
- User Management Module
- Exam Management Module
- Question Bank Module
- Result Processing Module

- **Product Functions**
- User authentication and role-based access.
- Exam scheduling and notifications.
- Question creation and randomization.
- Automated result generation.
- Reports for teachers and administrators.

- **User Classes and Characteristics**
- **Student:** Appears for exams, views results.
- **Teacher/Examiner:** Creates questions, grades answers, views reports.
- **Administrator:** Manages users, exams, and system settings.
- **2.4 Operating Environment**
- **Frontend:** Web Browser (Chrome, Firefox, Edge).
- **Backend:** Server running Apache Tomcat.
- **Database:** MySQL.
- **OS:** Windows/Linux.
- **2.5 Design and Implementation Constraints**
- Must comply with GDPR for data privacy.
- Support at least 10,000 concurrent users.
- Use HTTPS for secure communication.
- **2.6 Assumptions and Dependencies**
- Users have stable internet access.
- Examiners prepare questions before the exam.

- **3. System Features**
- **3.1 User Registration & Authentication**
- **Description:** Allows users to register, login, and reset passwords.
- **Functional Requirements:**
  - FR1: The system shall provide a registration form with validation.
  - FR2: The system shall authenticate users based on email and password.
- **3.2 Exam Management**
- **Description:** Teachers can create and schedule exams.
- **Functional Requirements:**
  - FR3: The system shall allow teachers to add MCQs and descriptive questions.
  - FR4: The system shall randomize questions for each student.
- **3.3 Result Generation**
- **Description:** Students’ performance is recorded and results are generated automatically.
- **Functional Requirements:**
  - FR5: The system shall auto-grade objective questions.
  - FR6: The system shall allow manual grading for descriptive answers.
  - FR7: The system shall generate a result report for each student.

External Interface Requirements

- **4.1 User Interfaces**
- Responsive web interface.
- Dashboard for each role (Student, Teacher, Admin).
- **4.2 Hardware Interfaces**
- Compatible with PCs, laptops, and tablets.
- **4.3 Software Interfaces**
- Integration with existing student information systems.
- **4.4 Communications Interfaces**
- Secure communication via HTTPS.
- Email/SMS notification integration.

- **Non-Functional Requirements**
- **5.1 Performance Requirements**
- The system shall support 10,000 concurrent users.
- Each exam submission must be saved within 2 seconds.
- **5.2 Security Requirements**
- Passwords shall be stored using encryption (bcrypt).
- Role-based access control must be implemented.
- **5.3 Reliability & Availability**
- System uptime shall be 99.9%.
- Data backup must occur daily.
- **5.4 Usability**
- Intuitive UI with accessibility compliance (WCAG 2.1).

System view of project management

- The **system view of project management** treats a project not as an isolated effort, but as an interconnected part of a larger organizational and environmental ecosystem. This approach emphasizes holistic thinking, strategic alignment, and dynamic interdependencies across all project components.

Core Principles of the System View

- **Holistic Perspective**: Projects are seen as systems with inputs, processes, outputs, and feedback loops.
- **Interconnectedness**: Every decision affects other parts of the organization—resources, stakeholders, operations, and strategy.
- **Adaptability**: Encourages continuous learning and adjustment based on feedback and environmental changes.
- **Strategic Alignment**: Projects must support broader organizational goals and values.

Example: Coffee Corner in a Restaurant

- A system view doesn’t just install a coffee machine—it considers:
- - Staff training
- - Menu integration
- - Marketing strategy
- - Space utilization
- - Customer experience
- - Revenue impact
- All elements must align to achieve the strategic goal of boosting coffee sales

Benefits of the System View

- Better decision-making through contextual awareness
- Enhanced stakeholder engagement
- Improved project adaptability and resilience
- Stronger alignment with business strategy
- Reduced risk of siloed or misaligned efforts

BENNETT UNIVERSITY
THE TIMES GROUP

Function Point Analysis

Introduction

- **FPA measures the functional size** of a software application based on what it does for the user.
- It uses **Function Points** as a unit, similar to km or kg, instead of counting lines of code.
- Developed by **A.J. Albrecht at IBM** in the 1980s to improve software size estimation.
- The **International Function Point User Group (IFPUG)** was formed in 1986 to standardize the method.
- FPA is **technology-independent** and focuses on functionality delivered to the **end user**.

History of FP

MID 70s

First version by Allan Albrecht from IBM



1979

First exposure to the software community



1984

IFPUG is born

The International Function Point Users Group

2009

IFPUG FPs become an ISO Standard

ISO

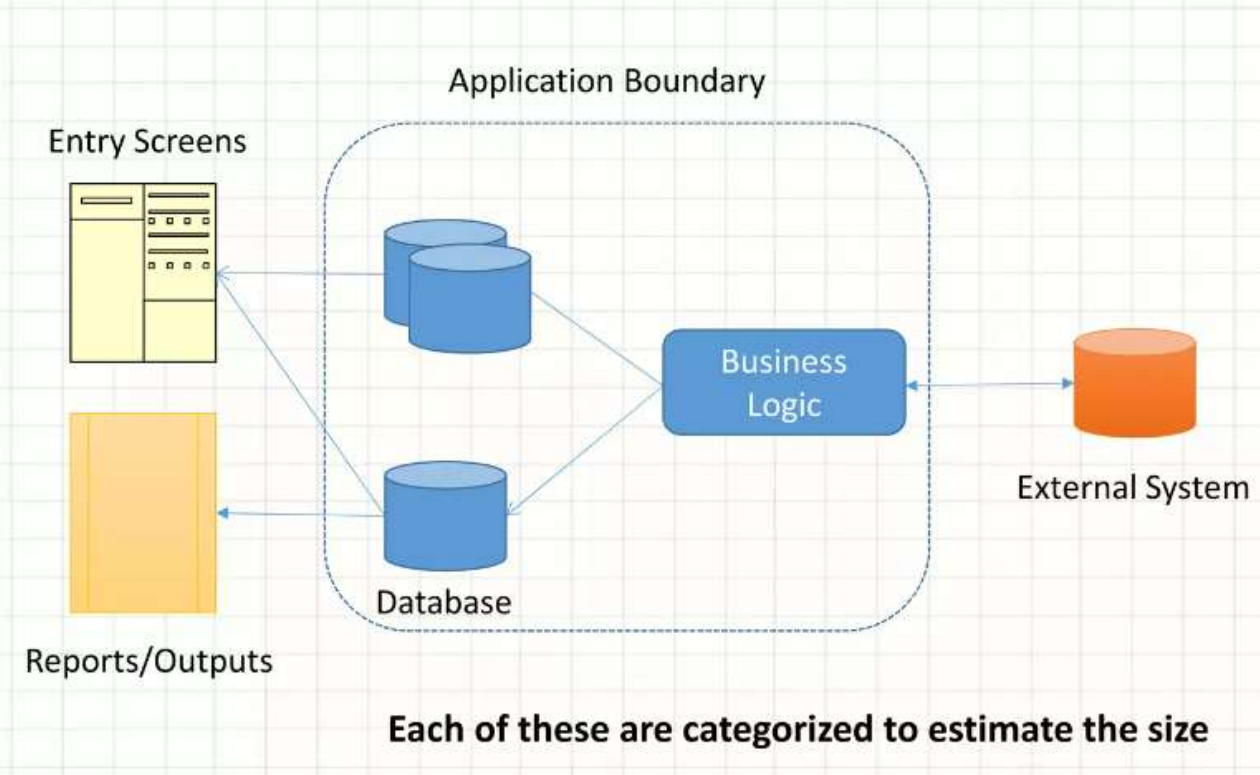
ICS › 35 › 35.080

ISO/IEC 20926:2009

FPA Key Concepts

- The size in FPA is determined by counting the number of inputs, outputs, queries, internal files and external files in the system and adjusting that total for the functional complexity of the system.
  - Data at Rest = Tables/Storage
  - Data in motion = Transactions
- Application Boundary = Within the application to be developed
- Elementary process = A series of steps, which moves data (interwoven EPs form a system), smallest unit of activity that is meaningful to user.

FPA Key Concepts

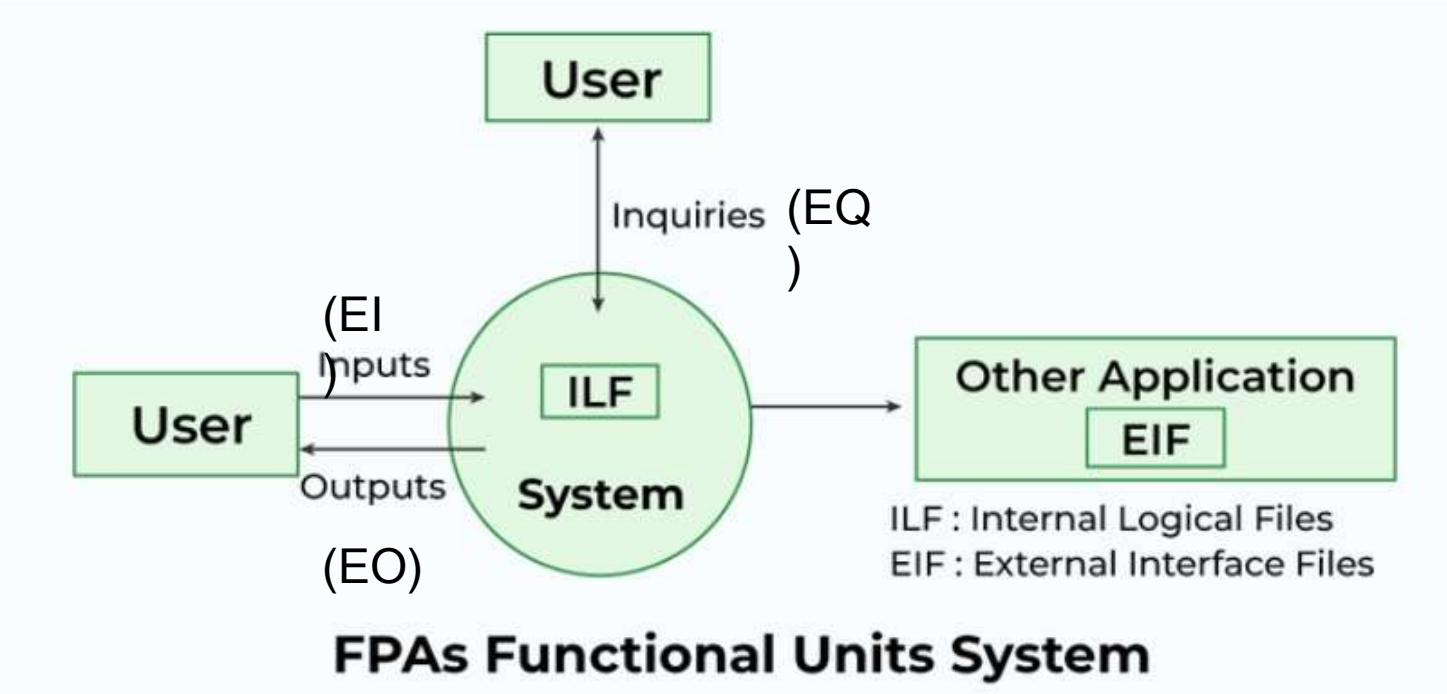


- The application boundary carries user context
- The boundary is not a technical concept
- The data residing inside Business the boundary is data at rest
- The data moving into or Reports/Outputs out of boundary is data in motion
- Each of these are categorized to estimate the size

Mapping of each component to Function Points

Visual Element	FPA Component	Description	Count What?
Entry Screens	**External Input (EI)**	User inputs that maintain ILFs	Each unique input process
Database (Internal)	**Internal Logical File (ILF)**	Data maintained by the app	Each logical data group
External System	**External Interface File (EIF)**	Data maintained elsewhere	Each external data group referenced
Reports/Outputs	**External Output (EO)** or **External Inquiry (EQ)**	Data sent outside boundary	Each unique output process

Components of FPA



- Two Data Function Types (ILF, EIF)
- Three Transactional Function Types. (EI, EO, IQ)

Elementary Processes

Data Files

(Data at rest)



Internal Logical Files (ILFs)

External Interface Files (EIFs)

Transactions

(Data in motion)



External Inputs (EIs)

External Outputs (EOs)

External Queries (EQs)

1. Internal Logical Files (ILF)



Definition: Logical groupings of data in a system, maintained by an end user(and application), are referred to as Internal Logical Files (ILF).



For example, a pilot may enter navigational data through a display in the cockpit prior to departure. The data is stored in a file for use and can be modified during the mission. Therefore, the pilot is responsible for maintaining the file that contains the navigational information.

Case Study : University System

- Data Elements (DETs): - Student ID - Name - Date of Birth - Email - Phone - Address - Enrollment Date - Program - Current Semester - CGPA Record Types (RETs): - Basic Info - Academic Info - Contact Info

Course Catalog file
Data Elements: - Course Code - Course Name - Credits - Department - Prerequisites - Description - Max Enrollment

Internal Logical File3
Data Elements: - Faculty ID - Name - Department - Designation - Qualifications - Email - Office Location

2. External Interface Files (EIF)



Definition: Groupings of data from another system that are used only for reference purposes are defined as External Interface Files (EIF).



Explanation: The second Data Function a system provides an end user is also related to logical groupings of data. In this case the user is not responsible for maintaining the data. The data resides in another system and is maintained by another user or system. The user of the system being counted requires this data for reference purposes only.



For example, it may be necessary for a pilot to reference position data from a satellite or ground-based facility during flight. The pilot does not have the responsibility for updating data at these sites but must reference it during the flight.

Govt Education Database

- Referenced for: - Verifying previous education - Checking accreditation status - Validating transfer credits Data Elements Referenced: - Institution Code - Institution Name - Accreditation Status - Recognized Degrees.

- Banking System
- For fee payment

Transactional Functions

- These functions address the user's capability to access the data contained in ILFs and EIFs. This capability includes maintaining, inquiring and outputting of data. These are referred to as Transactional Functions.

1) External Inputs (EI)

It gives the user the capability to maintain the data in ILF's through adding, changing and deleting its contents. In this case the pilot is utilizing a transaction referred to as an External Input (EI).

For example, a pilot can add, change and delete navigational information prior to and during the mission.

- Processing: - Validate information - Generate Student ID - Create student record in ILF - Send confirmation email

External Input 3

Update Faculty Information

External Input 4

Delete Course

Transactional Functions

2) External Output (EO)

This Transactional Function gives the user the ability to produce outputs.

For example, a pilot has the ability to separately display ground speed, true air speed and calibrated air speed. The results displayed are derived using data that is maintained and data that is referenced. In function point terminology the resulting display is called an External Output (EO).

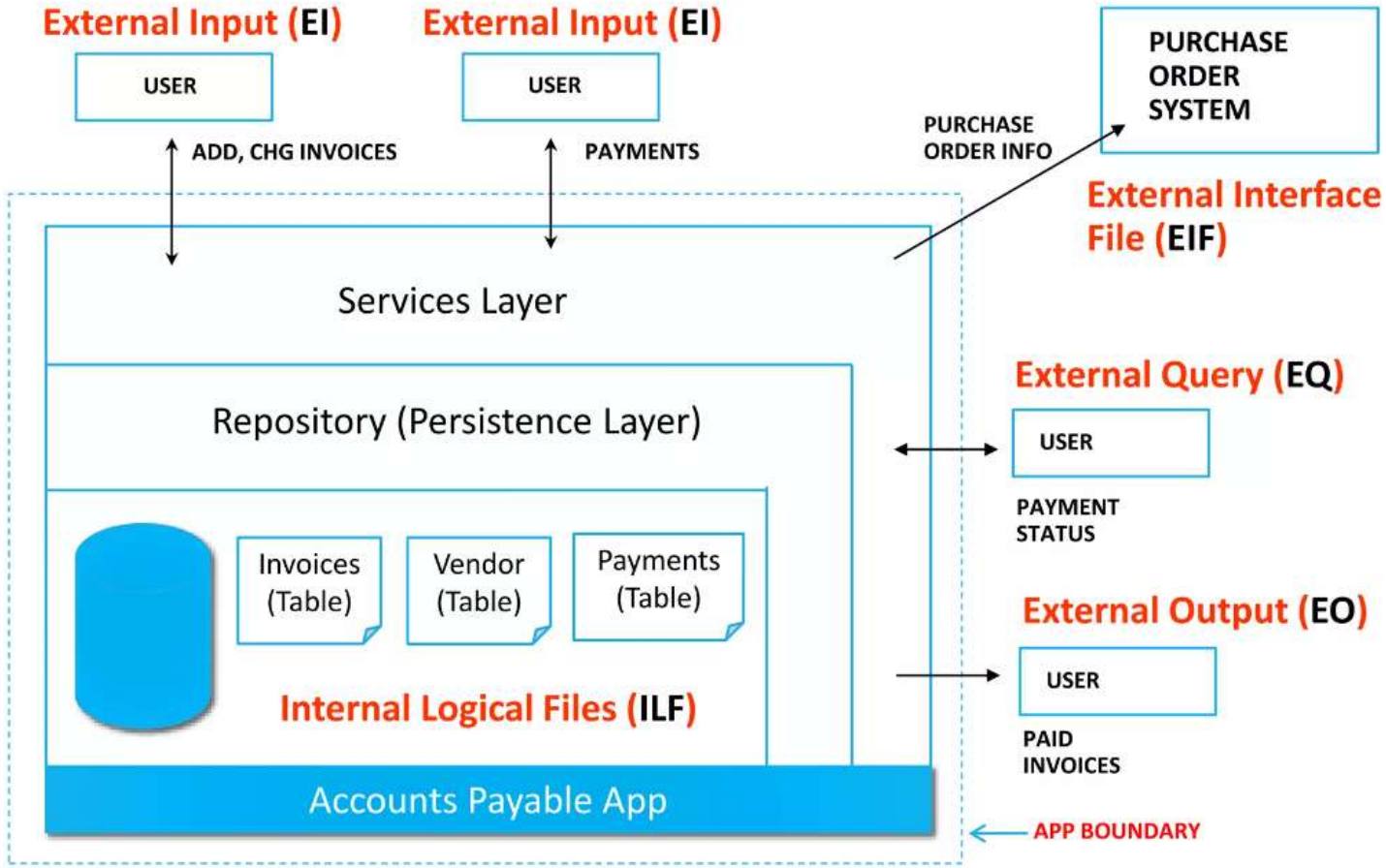
3) External Queries (EQ)

This Transactional Functions addresses the requirement to select and display specific data from files. To accomplish this a user inputs selection information that is used to retrieve data that meets the specific criteria. In this situation there is no manipulation of the data. It is a direct retrieval of information contained on the files. **(without referecing logical files)**

For example, if a pilot displays terrain clearance data that was previously set, the resulting output is the direct retrieval of stored information. These transactions are referred to as External Inquiries (EQ).

- EO1 External output –
- Student GPA report
- Departmental Progress Report
- Fee Payment Invoice

Elementary Processes through E-Commerce example



Calculation of Function Points

FPA= CAF* Count function of UFP

Calculation of CAF:

Step 1: CAF = [0.65 + 0.01 * sum (F_i)]

Where **sum (F_i) = Complexity Adjustment Factor**

F = 14 * Scale

There are 14 software characteristics (shown in next slide) taken to calculate the software complexity and a Scale is assigned to each factor.

Rate each factor on a scale of 0 to 5.

0 - No Influence

1 - Incidental

2 - Moderate

3 - Average

4 - Significant

5 - Essential

Number of factors considered (F_i)
1.Does the system require reliable backup and recovery ?
2.Is data communication required ?
3.Are there distributed processing functions ?
4.Is performance critical ?
5.Will the system run in an existing heavily utilized operational environment ?
6.Does the system require on line data entry ?
7.Does the on line data entry require the input transaction to be built over multiple screens or operations ?
8.Are the master files updated on line ?
9.Is the inputs, outputs, files, or inquiries complex ?
10.Is the internal processing complex ?
11.Is the code designed to be reusable ?
12.Are conversion and installation included in the design ?
13.Is the system designed for multiple installations in different organizations ?
14.Is the application designed to facilitate change and ease of use by the user ?

- Step-2: Calculate Unadjusted Function Point (UFP).

Table 1 : Functional units with weighting factors

Functional Units	Weighting factors		
	Low	Average	High
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
External logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10



All the parameters mentioned in the Table 1 are assigned some weights that have been determined by experience.



The functional complexities are multiplied with the corresponding weights against each function, and the values are added up to determine the UFP (Unadjusted Function Point) of the subsystem.

The procedure for the calculation of Unadjusted Function Point (UFP) is given in table shown below.

Table 2: UFP Calculation Table

Domain Characteristics	Count		Weighting Factor			Count
			low	avg	high	
Number Of User Inputs		*	3	4	6	
Number Of User Outputs		*	4	5	7	
Number Of User Enquiries		*	3	4	6	
Number Of Files		*	7	10	15	
Number Of External Interfaces		*	5	7	10	
Count Total						

Example 1:

Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

```
User Input = 50

User Output = 40

User Inquiries = 35

User Files = 6

External Interface = 4
```

Explanation:

- **Step-1:** As complexity adjustment factor is average (given in question), hence, `scale = 3.`
- `F = 14 * 3 = 42`
- **Step-2:** `CAF = 0.65 + ( 0.01 * 42 ) = 1.07`
- **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.`UFP.`
- `Total Count= (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628`
- **Step-4:** `Function Point = 628 * 1.07 = 671.96`

Example 2

- Study of requirement specification for ABC project has produced following results –

- need for
  - 7 inputs,
  - 10 outputs,
  - 6 inquiries,
  - 17 files &
  - 4 external interfaces.

- Input & external interface FP attributes are of **Average Complexity** & all other FP attributes are of **– Low Complexity**

- Determine – adjusted FPs assuming **Complexity Adjustment Value** is → 32

Calculation of UFP

Domain Characteristics	Count		Weighting Factor			Count
			Simple	Average	Complex	
Number Of User Inputs	7	**X**		**4**		28
Number Of User Outputs	10	**X**	**4**			40
Number Of User Enquiries	6	**X**	**3**			18
Number Of Files	17	**X**	**7**			119
Number Of External Interfaces	4	**X**		**7**		28
Count Total						233

Calculation of FP

FP = Count Total (UFP) * {(0.65 + 0.01) * Sum(Fi)}

= 233 * {(0.65 + 0.01) * 32}

- = 233 * 0.97
- = 226.01
- Hence, adjusted FP = 226.01

Example

An application has the following:

- 10 low external inputs,
- 12 high external outputs,
- 20 low internal logical files,
- 15 high external interface files,
- 12 average external inquiries,
- and a value of complexity adjustment factor of 1.10.
- What are the unadjusted and adjusted function point counts ?



THANK YOU



ANY QUESTIONS

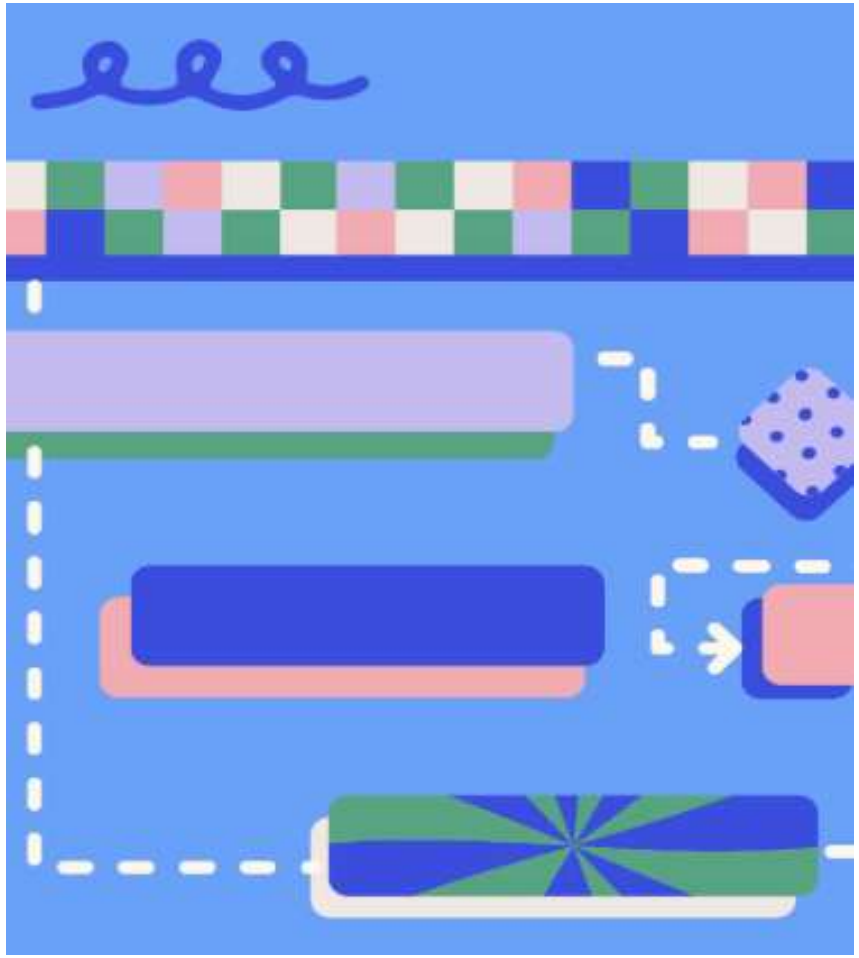
CSET-324: Software Project Management

Activities Sequencing



Defining Activities

- A**ctivity** (or **task):** An element of work normally found on (WBS) that has an expected duration, a cost, and resource requirements.
- Activity definition involves developing a more detailed WBS and supporting explanations to understand all the work to be done so you can develop realistic cost and duration estimates.



Activity Lists

- Tabulation of activities to be included on a project schedule that includes
  - the activity name
  - an activity identifier or number
  - a brief description of the activity



PROJECT ACTIVITY LIST

Project name: Organizing a corporate party	Start date: November 20, 2023
Project number: 6	Estimated completion date: November 30, 2023
Project manager: Jonathan Peters	Client: Michael Jones

Activity number:	Activity name:	Activity description:	Assigned to:	Additional comments:
1	Setting the budget	Deciding on the budget necessary and available for the entire event	Mandy Collins Evelyn Steele	Allocate some money for gifts of appreciation
2	Setting the date			
3	Finding a venue			
4	Deciding on food and drinks			

Activity Attributes

- Provide more information such as
  - predecessors,
  - successors,
  - logical relationships,
  - leads and lags,
  - resource requirements,
  - constraints,
  - imposed dates, and
  - assumptions related to the activity

Milestones

- A significant event that normally has no duration
- Often constitutes several activities to complete a milestone
- Useful tool for setting schedule goals and monitoring progress
- Examples include obtaining customer sign-off on key documents or completion of specific products



Sequencing Activities

- Involves reviewing activities and determining dependencies
- A **dependency** or **relationship** is the sequencing of project activities or tasks
- You *must* determine dependencies in order to use critical path analysis

Example: Dependencies

Consider your team has been assigned to build a College Event Management App before the upcoming fest—2 months from now. It must handle event creation, student registration, notifications, and schedules. You have a team of 12 weeks to deliver it.

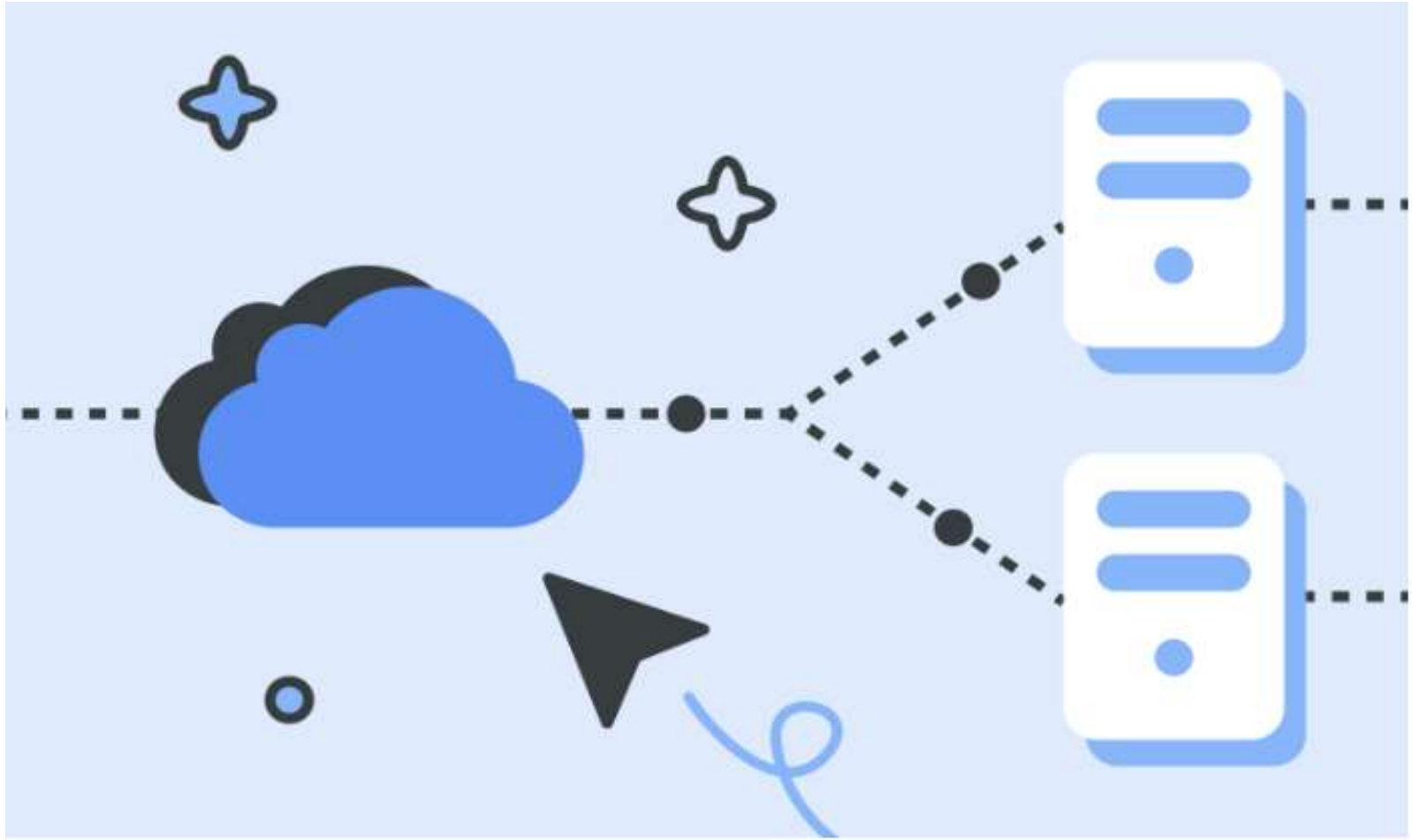
Sample Dependencies

Task ID	Task	Duration (weeks)	Depends On
A	Requirements Gathering	1	–
B	UI/UX Design	2	A
C	Front-End Development	4	B
D	Back-End Development	4	B
E	Integration & Testing	2	C, D
F	Deployment	1	E
G	Documentation & Training	1	E
H	Maintenance & Feedback Loop	1	F

Three types of Dependencies

- **Mandatory dependencies:** inherent in the nature of the work being performed on a project, sometimes referred to as hard logic
- **Discretionary dependencies:** defined by the project team, sometimes referred to as soft logic and should be used with care since they may limit later scheduling options
- **External dependencies:** involve relationships between project and non-project activities

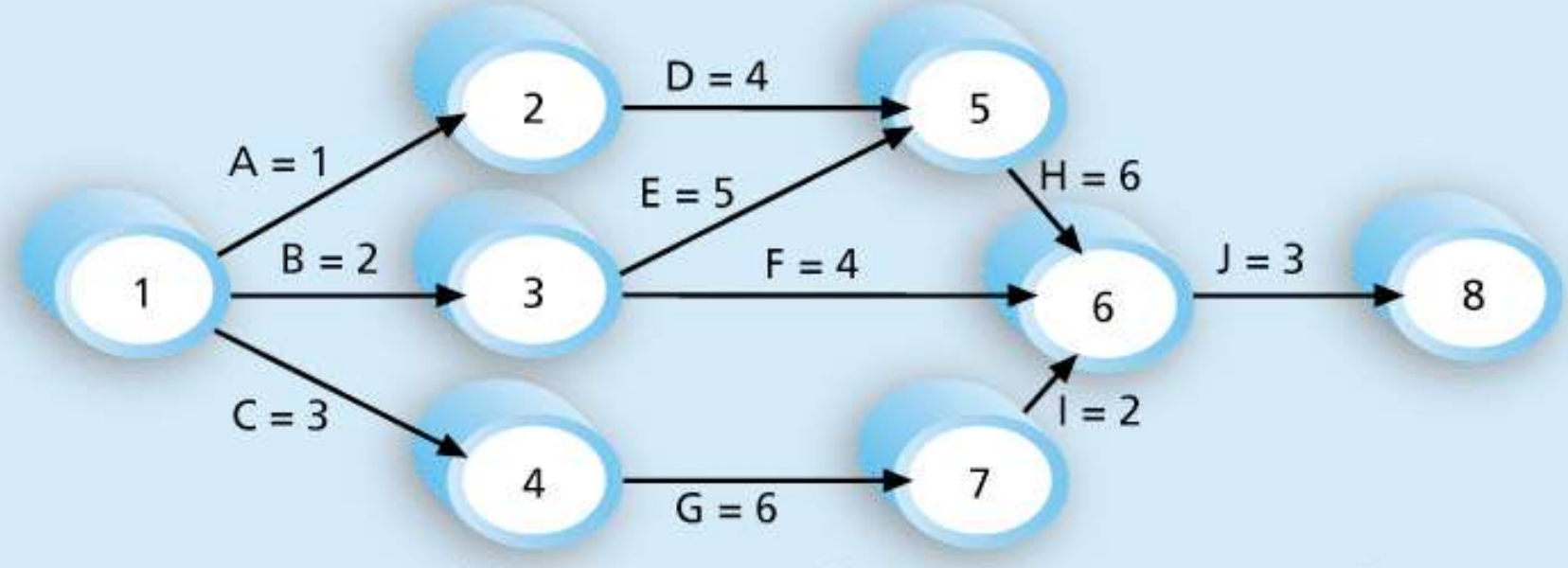
Network Diagrams



Network Diagrams

- Preferred technique for showing activity sequencing
- N**etwork diagram:** A schematic display of the logical relationships among, or sequencing of, project activities
- Two main formats are the arrow and precedence diagramming methods

Network Diagram for Project X



Note: Assume all durations are in days; A=1 means Activity A has a duration of 1 day.


BENNETT
UNIVERSITY
THE TIMES GROUP

1. Arrow Diagramming Method (ADM)



Also called activity-on-arrow (AOA) network diagrams.



Activities are represented by arrows.



Nodes or circles are the starting and ending points of activities.

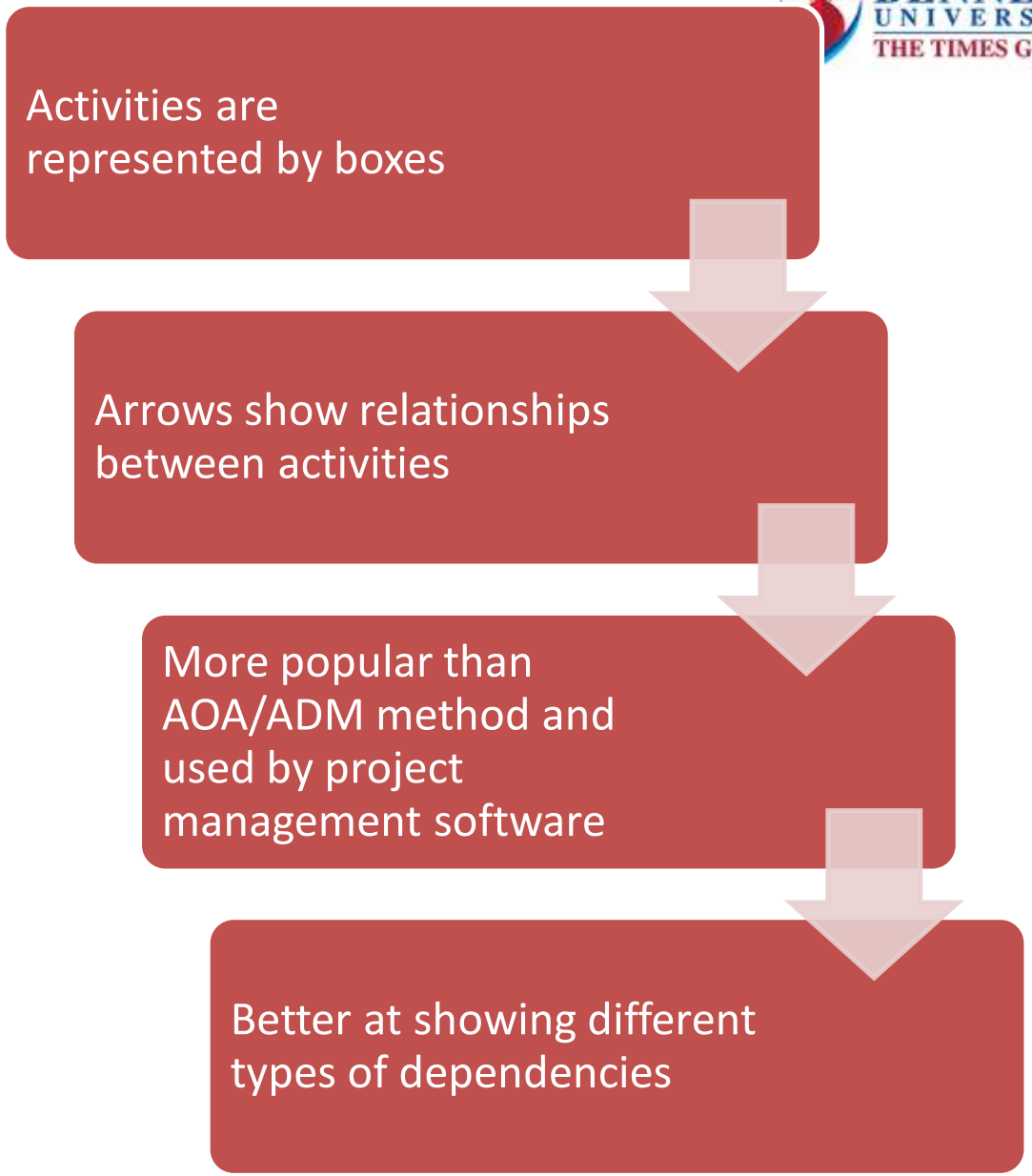


Can only show finish-to-start dependencies.

Process for Creating ADM/AOA Diagrams

1. Find all of the activities that start at node 1. Draw their finish nodes and draw arrows between node 1 and those finish nodes. Put the activity letter or name and duration estimate on the associated arrow
2. Continuing drawing the network diagram, working from left to right. Look for bursts and merges. **Bursts** occur when a single node is followed by two or more activities. A **merge** occurs when two or more nodes precede a single node
3. Continue drawing the project network diagram until all activities are included on the diagram that have dependencies
4. As a rule of thumb, all arrowheads should face toward the right, and no arrows should cross on an AOA network diagram

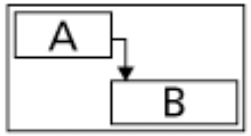
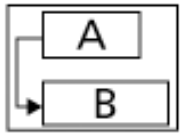
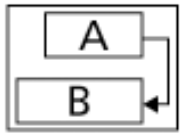
2. Precedence Diagramming Method (PDM)



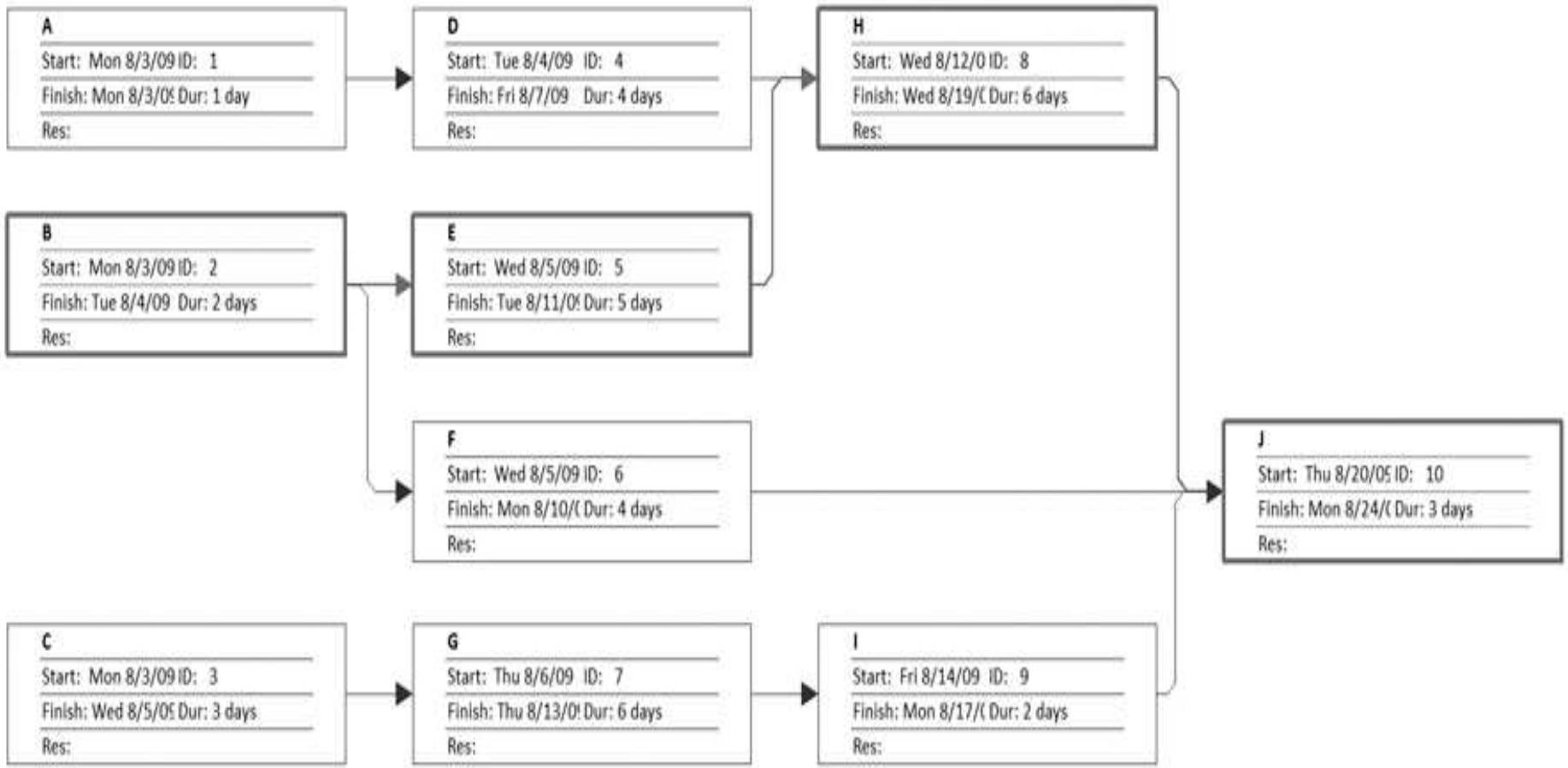
Task Dependency Types

Task dependencies

The nature of the relationship between two linked tasks. You link tasks by defining a dependency between their finish and start dates. For example, the "Contact caterers" task must finish before the start of the "Determine menus" task. There are four kinds of task dependencies in Microsoft Project.

Task dependency	Example	Description
Finish-to-start (FS)		Task (B) cannot start until task (A) finishes.
Start-to-start (SS)		Task (B) cannot start until task (A) starts.
Finish-to-finish (FF)		Task (B) cannot finish until task (A) finishes.
Start-to-finish (SF)		Task (B) cannot finish until task (A) starts.

Sample PDM Network Diagram



Activity Recourse Estimation

- Before estimating activity durations, you must have a good idea of the quantity and type of resources that will be assigned to each activity; **resources** are people, equipment, and materials
- Consider important issues in estimating resources
  - How difficult will it be to do specific activities on this project?
  - What is the organization's history in doing similar activities?
  - Are the required resources available?
- A **resource breakdown structure** is a hierarchical structure that identifies the project's resources by category and type

Activity Duration Estimation

- **Duration** includes the actual amount of time worked on an activity *plus* elapsed time
- **Effort** is the number of workdays or work hours required to complete a task
- Effort does not normally equal duration
- People doing the work should help create estimates, and an expert should review them

Three-Point Estimates

- Instead of providing activity estimates as a discrete number, such as four weeks, it's often helpful to create a **three-point estimate**
  - an estimate that includes an optimistic, most likely, and pessimistic estimate, such as three weeks for the optimistic, four weeks for the most likely, and five weeks for the pessimistic estimate
- Three-point estimates are needed for PERT and Monte Carlo simulations

BENNETT UNIVERSITY
THE TIMES GROUP



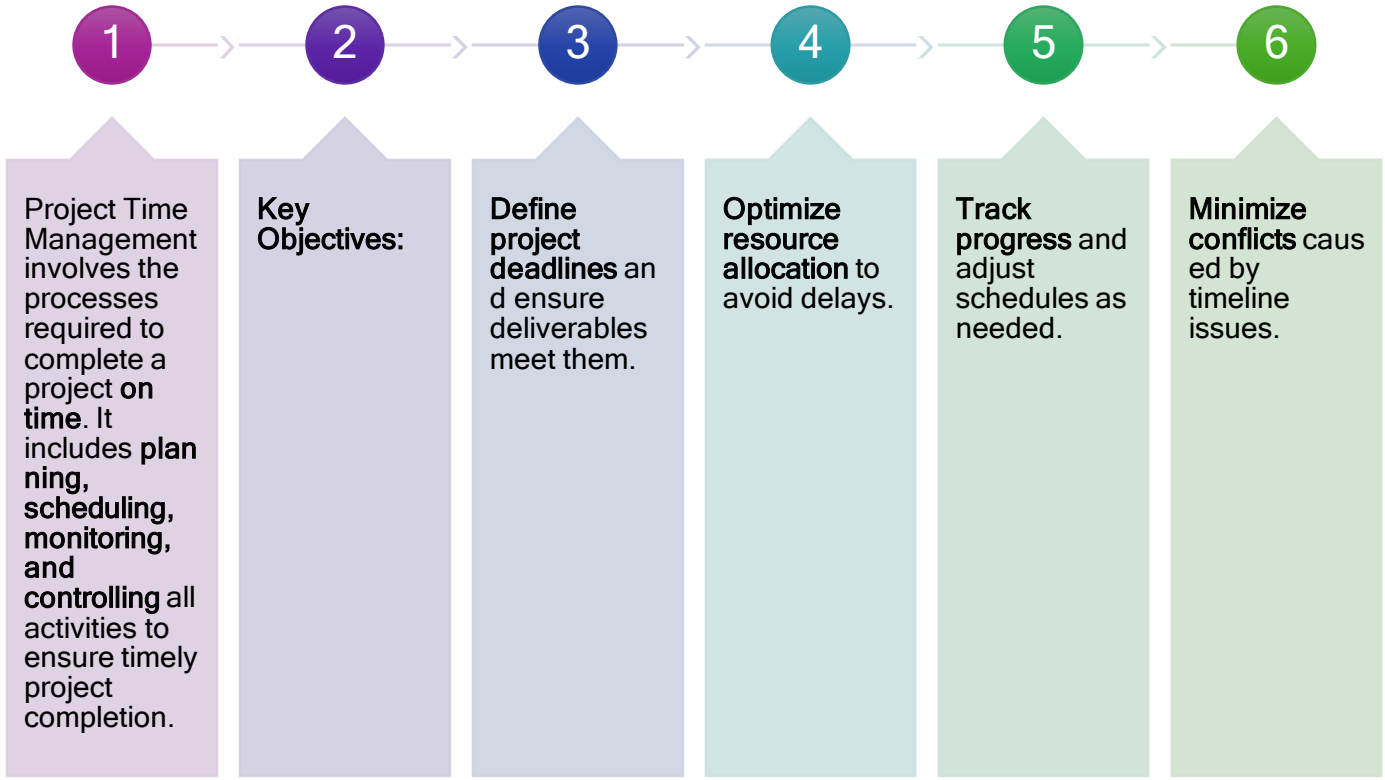
Project Time Management

Importance of Project Schedules

- Managers often cite delivering projects on time as one of their biggest challenges
- Time has the least amount of flexibility; it passes no matter what happens on a project
- Schedule issues are the main reason for conflicts on projects, especially during the second half of projects



Project Time Management



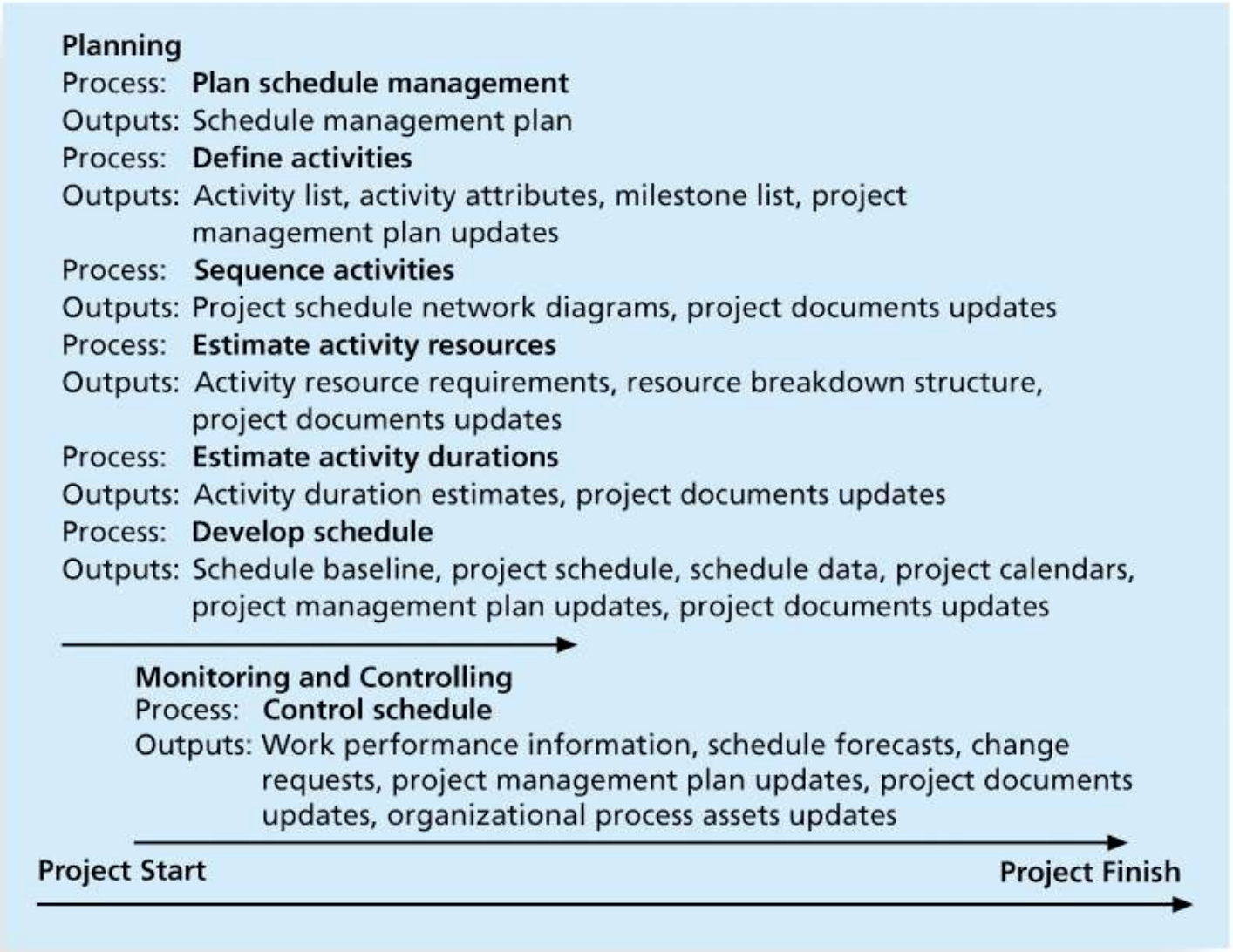
Project Time Management Processes

Planning	Planning schedule management: determining the policies, procedures, and documentation that will be used for planning, executing, and controlling the project schedule
Defining	Defining activities: identifying the specific activities that the project team members and stakeholders must perform to produce the project deliverables
Sequencing	Sequencing activities: identifying and documenting the relationships between project activities

Project Time Management Processes

Estimating	Estimating activity durations: estimating the number of work periods that are needed to complete individual activities
Developing	Developing the schedule: analyzing activity sequences, activity resource estimates, and activity duration estimates to create the project schedule
Controlling	Controlling the schedule: controlling and managing changes to the project schedule

Figure 1. Project Time Management Summary





THANK YOU



ANY QUESTIONS

BENNETT UNIVERSITY
THE TIMES GROUP

Project Scope Management

Project Scope Management



"The processes required to ensure that the project includes all the work required, and only the work required, to complete the project successfully"

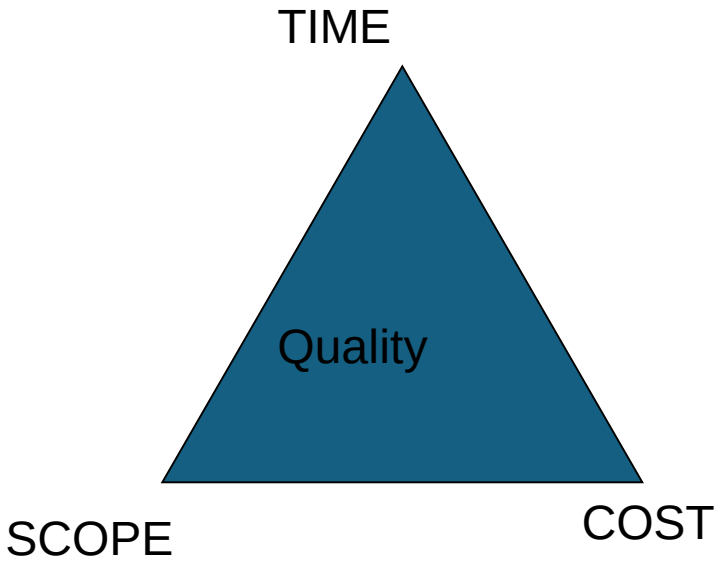


Answers the question – "What will the project produce in the end".

Why Do We Manage Scope?

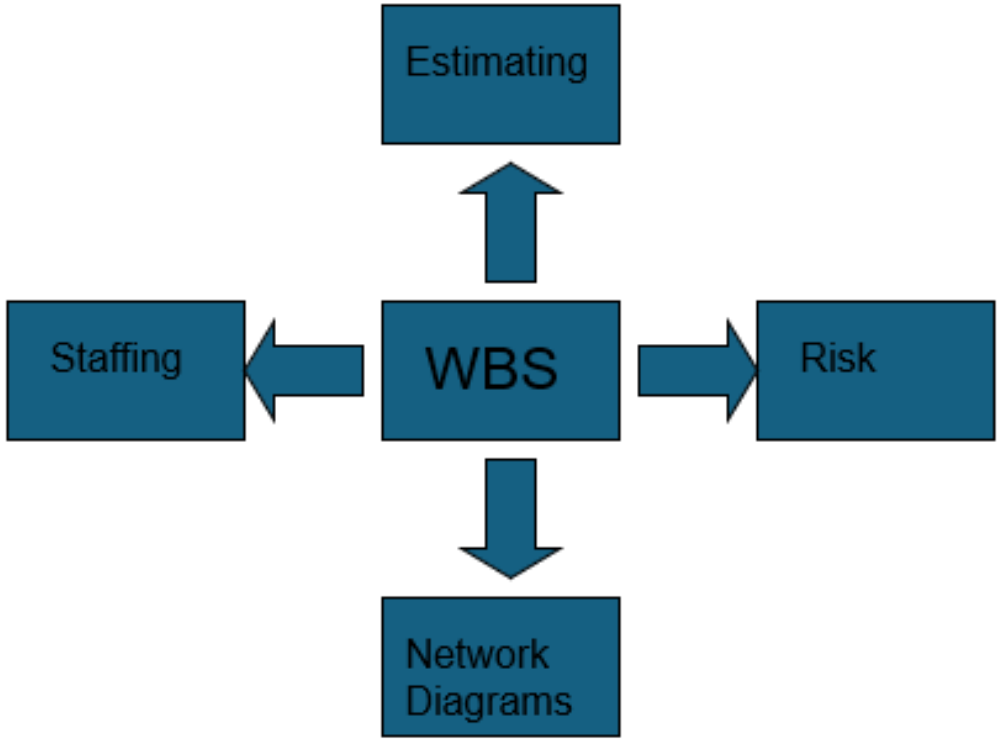
Can’t manage schedule and budget if scope is out of control (Triple Constraint)

Scope docs are used to manage expectations



Scope Management Key Points

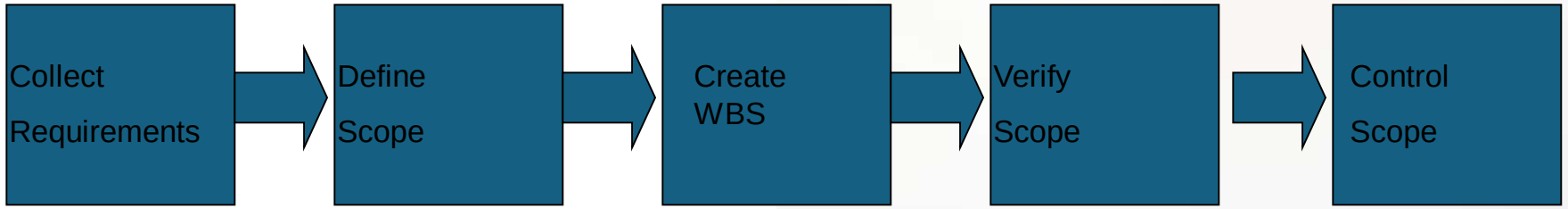
- What is scope management
  - Checking to ensure that one is completing work
  - Saying No to additional work not in the charter
  - Preventing extra work/gold plating
- Work Breakdown Structure (WBS)
  - Foundation of the project, all planning and controlling is based on the WBS
  - Identifies all work to be performed, if it is not in the WBS it does not need to be done
  - Graphical picture of work



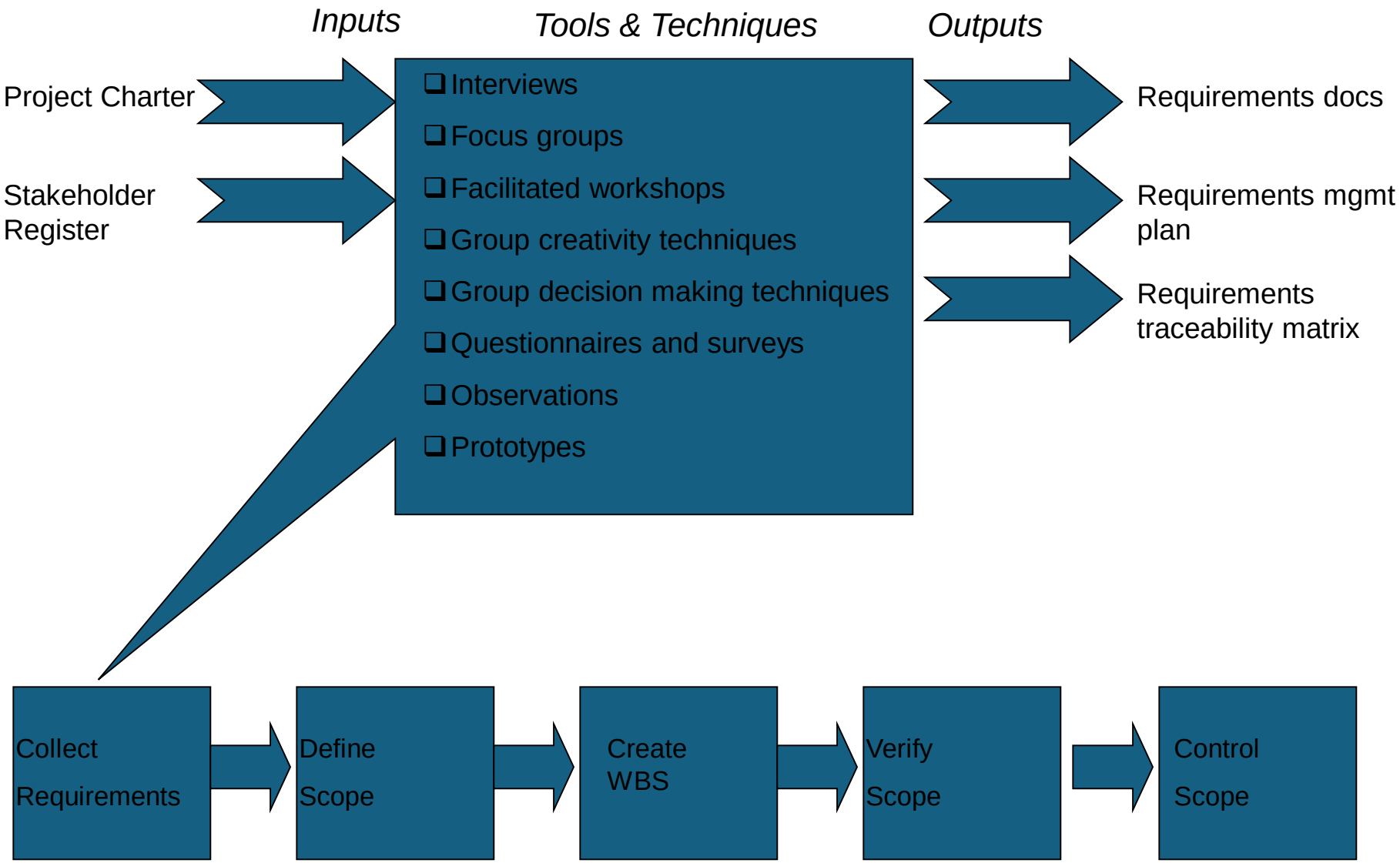
How Do We Manage Scope?

- Five processes
  - Collect Requirements
  - Define Scope
  - Create WBS
  - Verify Scope
  - Control Scope





Collect Requirements

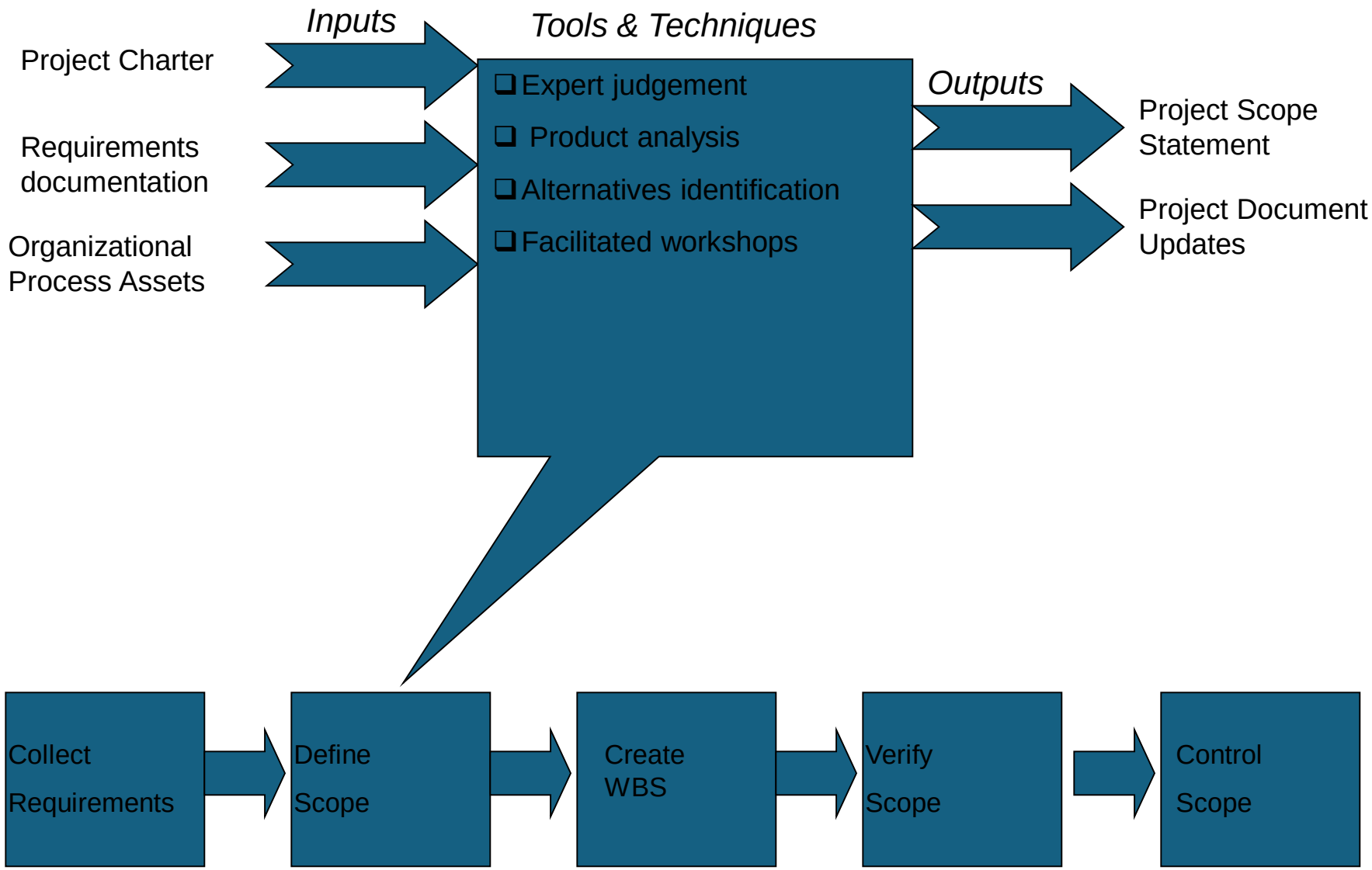


Collect Requirements

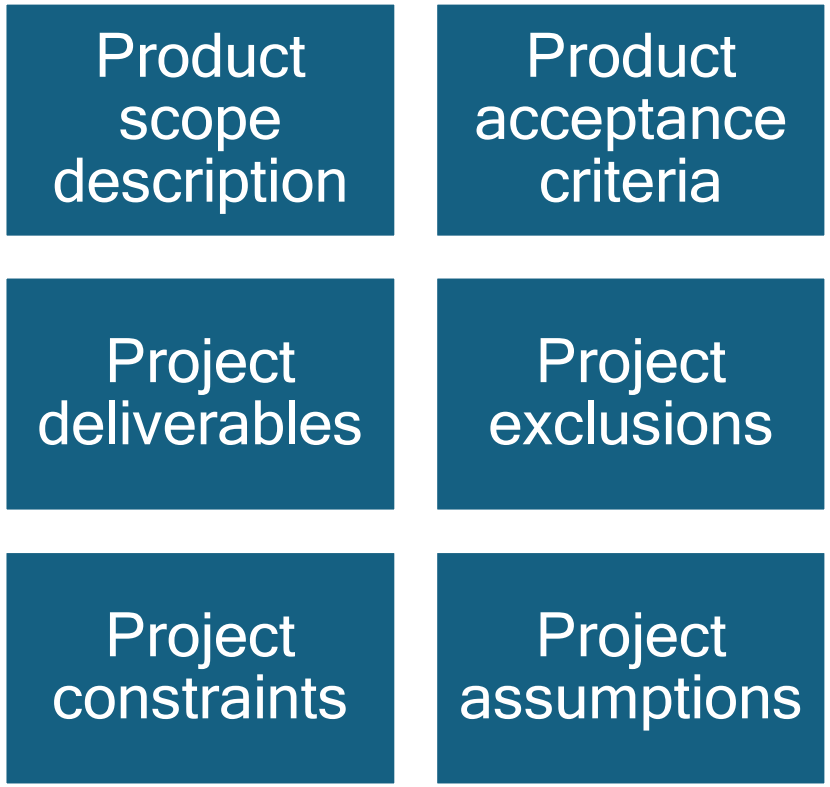
Determine product of the project requirements

Make sure all requirements support the business need of the project as described in the charter

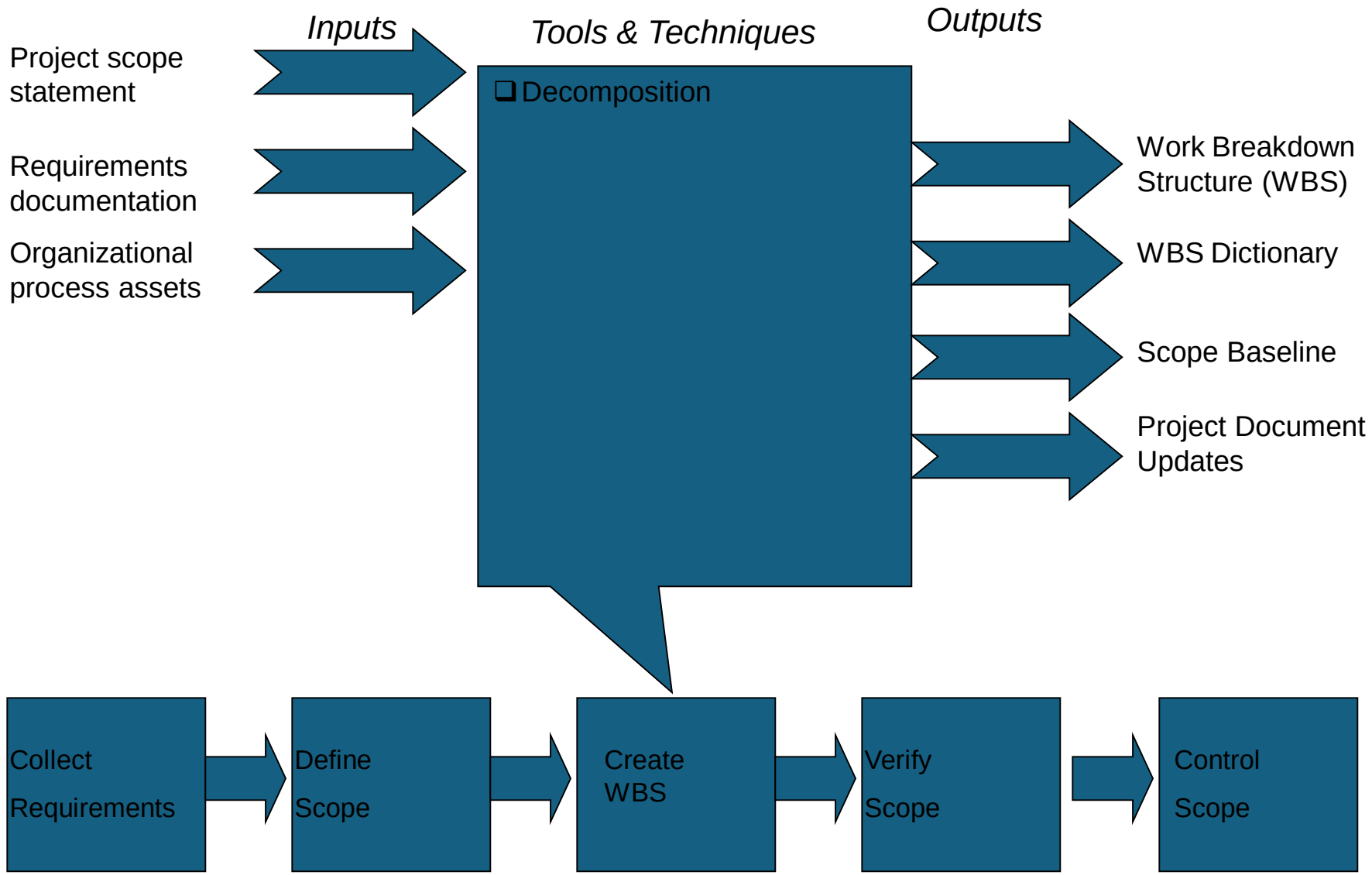
Define Scope



Project Scope Statement



Create WBS


Inputs
Tools & Techniques
Outputs
Project scope statement
Requirements documentation
Organizational process assets
❑ Decomposition
Work Breakdown Structure (WBS)
WBS Dictionary
Scope Baseline
Project Document Updates
Collect Requirements
Define Scope
Create WBS
Verify Scope
Control Scope

Work Breakdown Structure

Start with major project deliverables or phases

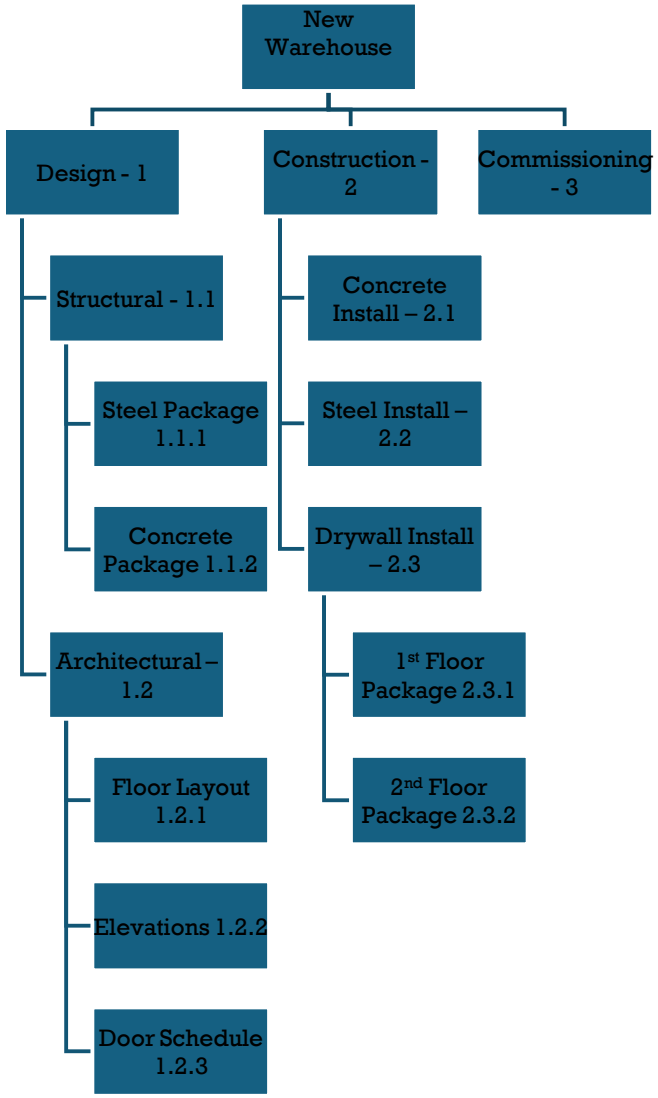
"Decomposition" is breaking down the deliverables into more manageable parts

Not all branches need the same level of decomposition!

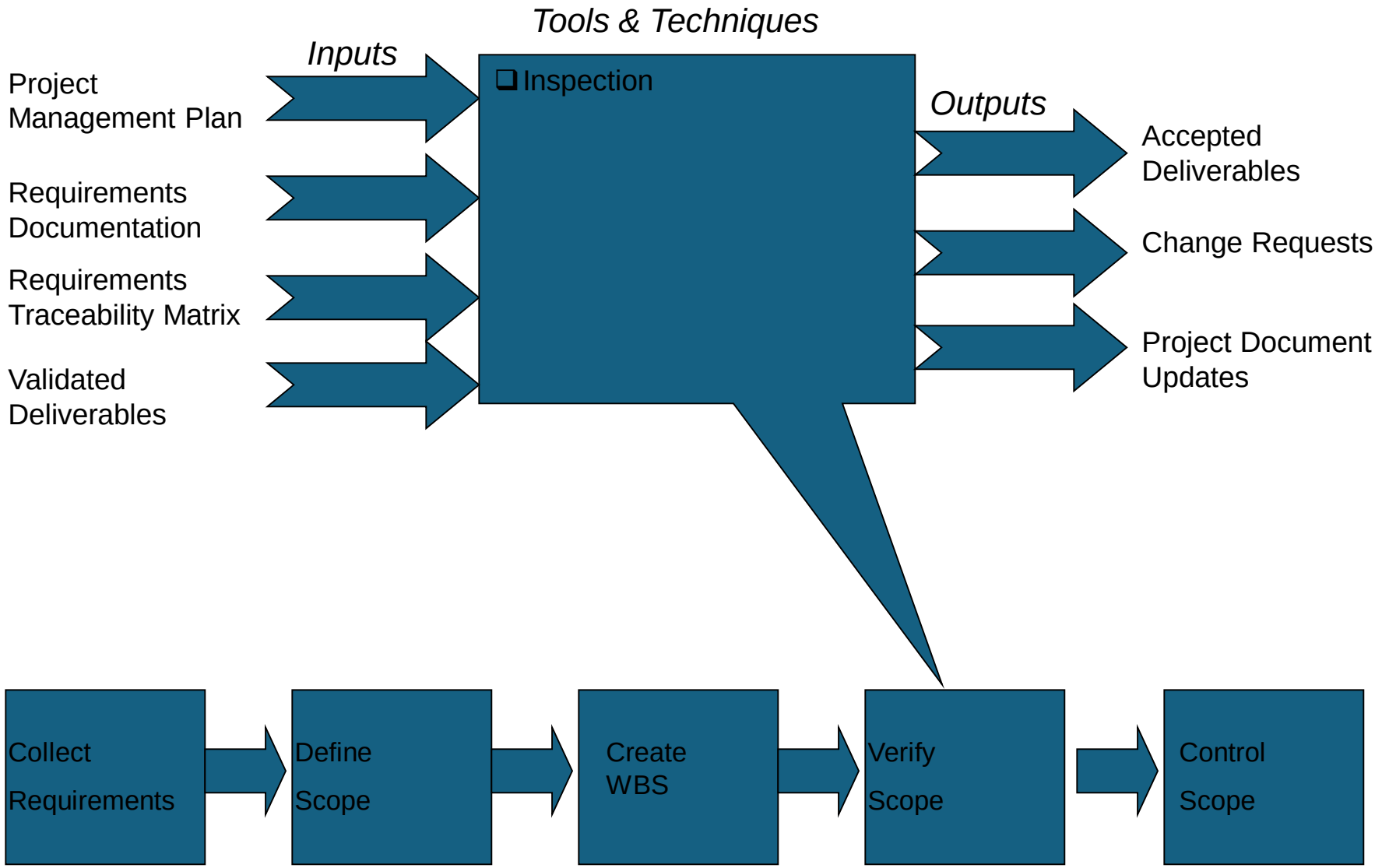
Decompose into "Work Packages"

8-80 Hour Rule ⇒ Don't break work down too much

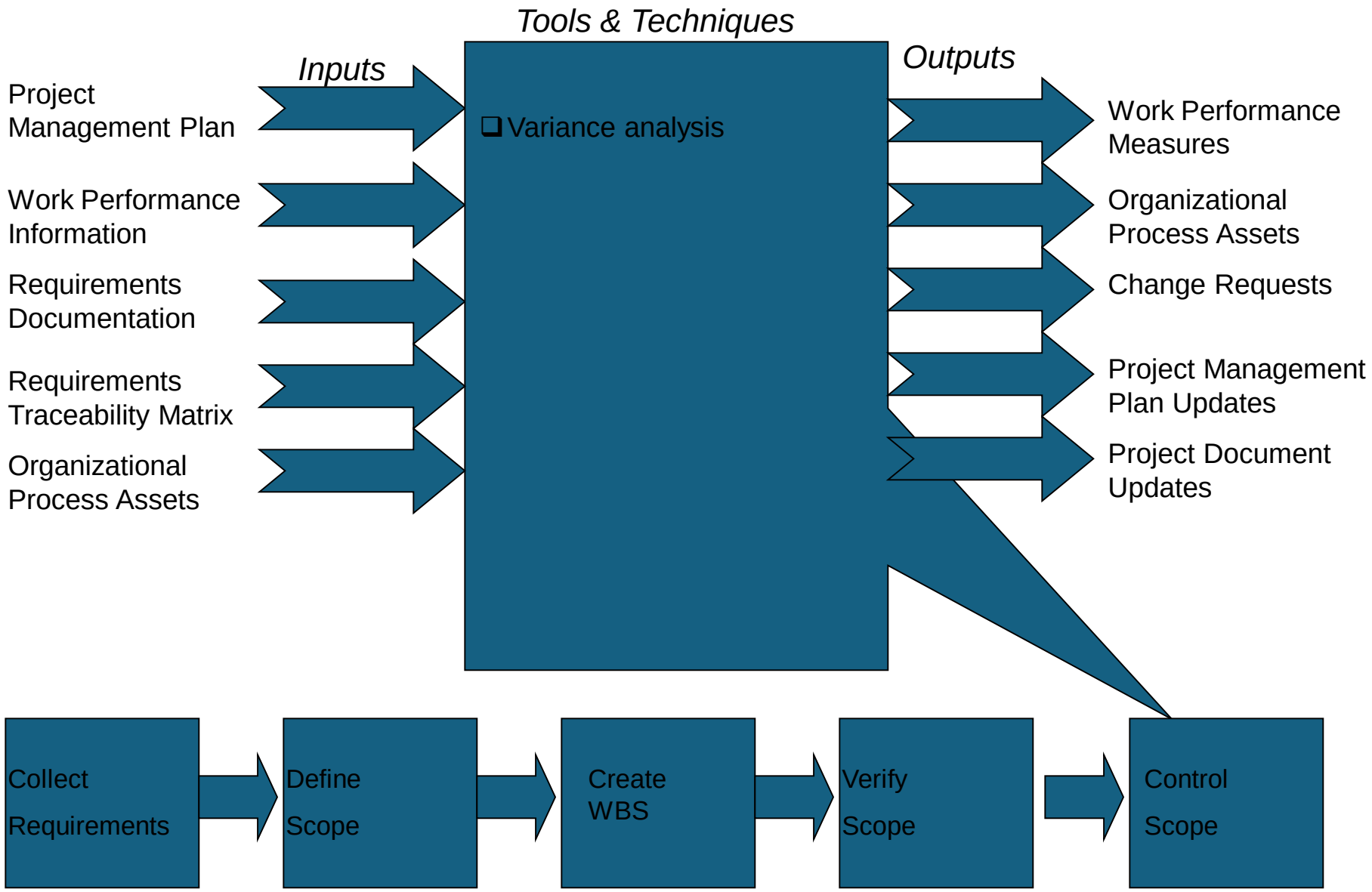
Work Breakdown Structure



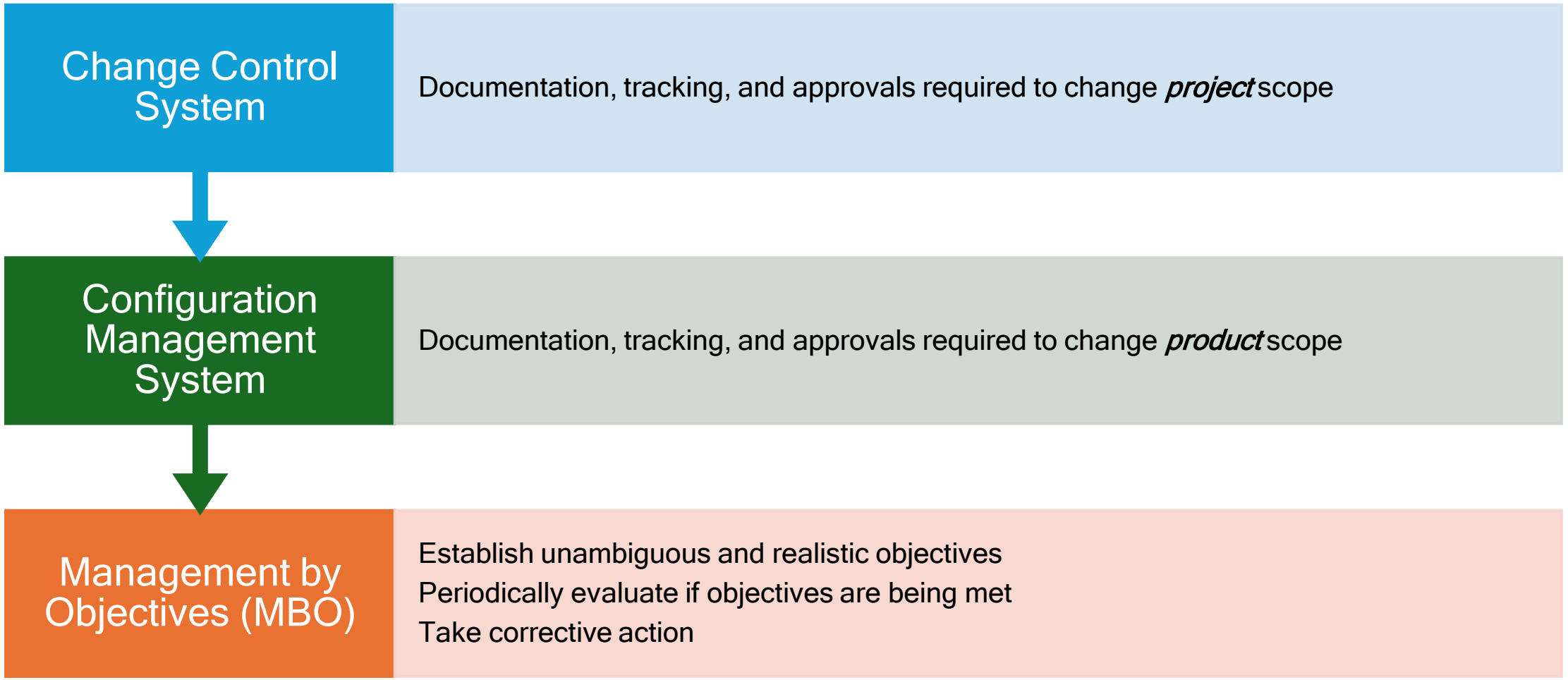
Verify Scope



Control Scope



Scope Definitions





THANK YOU



ANY QUESTIONS

Iterative steps for planning

- 1. Planning & Requirements
- - Define project goals and scope.
- - Identify key stakeholders and constraints.
- - Gather initial requirements and success criteria.
- - Establish iteration length (e.g., 1-4 weeks).

- 2. Analysis & Design
- - Analyze business needs and technical feasibility.
- - Design initial architecture or solution framework.
- - Prioritize features or user stories from the backlog.
- - Identify dependencies and risks.

- **3. Implementation**
- Develop the first iteration or prototype.
- Assign tasks based on team capacity and skills.
- Use version control and CI/CD pipelines if applicable.
- Maintain documentation for traceability.

- 4. Testing
- - Conduct unit, integration, and user acceptance testing.
- - Validate against the “Definition of Done.”
- - Gather feedback from stakeholders and users.
- - Log bugs and performance issues.

- 5. Evaluation & Review
- - Assess iteration outcomes vs. goals.
- - Conduct sprint review and retrospective.
- - Identify what worked, what didn’t, and areas for improvement.
- - Update backlog and adjust priorities.

- 6. Iterate & Improve
- - Refine requirements based on feedback.
- - Plan the next iteration with updated insights.
- - Continue cycles until the final product meets expectations.

- This cycle aligns with frameworks like Scrum, Kanban, and PDCA (Plan-Do-Check-Act). It promotes adaptability, continuous improvement, and stakeholder engagement throughout the project lifecycle.

Project Plan documentation methods

- 1. Centralized Repository Setup
- - Use a document management system (e.g., SharePoint, Confluence, Notion).
- - Ensure version control and access permissions.
- - Tag and index documents for easy retrieval.

- 2. Phase-Based Documentation
- Organize documents by project lifecycle stages: | Phase | Key Documents | |---|---| | Initiation | Project Charter, Business Case | | Planning | Project Plan, WBS, Risk Management Plan | | Execution | Status Reports, Task Assignments, Change Requests | | Monitoring & Control| Issue Logs, Performance Reports | | Closure | Final Report, Lessons Learned, Closure Checklist |

- 3. Documentation Plan Blueprint
- Include:
- - Purpose & Scope of documentation
- - Audience (internal teams, stakeholders)
- - Content Outline (topics, formats)
- - Procedures for updates and reviews
- - Reference Materials (diagrams, templates)

- 4. Collaborative Review Cycles
- - Schedule periodic reviews with team leads.
- - Use feedback tools (e.g., comments, annotations).
- - Track revisions and approvals.

- **5. Publishing & Accessibility**
- Host documents on a searchable portal.
- Use metadata, tags, and cross-references.
- Ensure mobile and offline access if needed.

6. Maintenance & Updates

- Assign ownership for each document.
- Set update frequency (e.g., weekly for issue logs).
- Archive outdated versions systematically.

BENNETT UNIVERSITY
THE TIMES GROUP

Software Review and Feedback

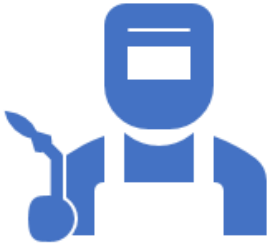
When are reviews needed?

- A *review* is any activity in which a work product is distributed to reviewers who examine it and give feedback.
  - Reviews are useful not only for finding and eliminating defects, but also for gaining consensus among the project team, securing approval from stakeholders, and aiding in professional development for team members.
  - Reviews help teams find defects soon after they are injected making them cost less to fix than they would cost if they were found in test.
  - All work products in a software project should be either reviewed or tested.
    - Software requirements specifications, schedules, design documents, code, test plans, test cases, and defect reports should all be reviewed.

Types of Review: Inspections



Inspections are moderated meetings in which reviewers list all issues and defects they have found in the document and log them so that they can be addressed by the author.



The goal of the inspection is to repair all of the defects so that everyone on the inspection team can approve the work product.

Commonly inspected work products include software requirements specifications and test plans.

Types of Review: Inspections

- Running an inspection meeting:
  1. A work product is selected for review and a team is gathered for an inspection meeting to review the work product.
  2. A moderator is chosen to moderate the meeting.
  3. Each inspector prepares for the meeting by reading the work product and noting each defect.
  4. In an inspection, a defect is any part of the work product that will keep an inspector from approving it.
  5. Discussion is focused on each defect, and coming up with a specific resolution.
     - It's the job of the inspection team to do more than just identify the problems; they must also come up with the solutions.
  6. The moderator compiles all of the defect resolutions into an *inspection log*

Inspection Log Example

Document:	Contract Royalty Subsystem, vision and scope doc.
Version::	1.3
# of Issues:	16
Review Date:	March 16, 2003

Attendees	**Read Document**	**Time Spent Preparing**
Mike (project manager)	Y	Author
Barbara (VP)	Y	1.0 hours
Quentin (requirements analyst)	Y	2.0 hours
Sophie (senior QA engineer)	Y	3.0 hours
Jill (senior programmer)	Y	0.5 hours

Issue No.	**Section / Page**	**Identified By**	**Issue**
1	Global	Quentin	The term "standard contract" should be replaced with "pro-forma contract"
2	Section 3.1.1 Line 165	Sophie	The contents of the cells in the table are out of order. It looks like some cells were moved down.
3	Section 3.1.2 Line 190	Jill	Specify the look up is by contract number and artist name.
4	Section 3.3b Line 623	Sophie	Tile of the section needs to be changed to "Deletion File (Maintenance). To be consistent with section 3.2.1 #1

Types of Review: Deskchecks

- A *deskcheck* is a simple review in which the author of a work product distributes it to one or more reviewers.
  - The author sends a copy of the work product to selected project team members. The team members read it, and then write up defects and comments to send back to the author.

Types of Review: Deskchecks

- Unlike an inspection, a deskcheck does not produce written logs which can be archived with the document for later reference.
- Deskchecks can be used as predecessors to inspections.
  - In many cases, having an author of a work product pass his work to a peer for an informal review will significantly reduce the amount of effort involved in the inspection.

Types of Review: Walkthroughs

- A *walkthrough* is an informal way of presenting a technical document in a meeting.
  - Unlike other kinds of reviews, the author runs the walkthrough: calling the meeting, inviting the reviewers, soliciting comments and ensuring that everyone present understands the work product.
  - Walkthroughs are used when the author of a work product needs to take into account the perspective of someone who does not have the technical expertise to review the document.
  - After the meeting, the author should follow up with individual attendees who may have had additional information or insights. The document should then be corrected to reflect any issues that were raised.

Types of Review: Code Review

- A *code review* is a special kind of inspection in which the team examines a sample of code and fixes any defects in it.
  - In a code review, a defect is a block of code which does not properly implement its requirements, which does not function as the programmer intended, or which is not incorrect but could be improved
    - For example, it could be made more readable or its performance could be improved

Types of Review: Code Review

- It's important to review the code which is most likely to have defects. This will generally be the most complex, tricky or involved code.
- Good candidates for code review include:
  - A portion of the software that only one person has the expertise to maintain
  - Code that implements a highly abstract or tricky algorithm
  - An object, library or API that is particularly difficult to work with
  - Code written by someone who is inexperienced or has not written that kind of code before, or written in an unfamiliar language
  - Code which employs a new programming technique
  - An area of the code that will be especially catastrophic if there are defects

Types of Review: Pair Programming

- *Pair programming* is a technique in which two programmers work simultaneously at a single computer and continuously review each others' work.
- Although many programmers were introduced to pair programming as a part of Extreme Programming, it is a practice that can be valuable in any development environment.
- Pair programming improves the organization by ensuring that at least two programmers are able to maintain any piece of the software.

Types of Review: Pair Programming

- In pair programming, two programmers sit at one computer to write code. Generally, one programmer will take control and write code, while the other watches and advises.
  - Some teams have found that pair programming works best for them if the pairs are constantly rotated; this helps diffuse the shared knowledge throughout the organization. Others prefer to pair a more junior person with a more senior for knowledge sharing.
- The project manager should not try to force pair programming on the team; it helps to introduce the change slowly, and where it will meet the least resistance.
  - It is difficult to implement pair programming in an organization where the programmers do not share the same nine-to-five (or ten-to-six) work schedule.
  - Some people do not work well in pairs, and some pairs do not work well together.



THANK YOU



ANY QUESTIONS

BENNETT UNIVERSITY
THE TIMES GROUP

Software Measurement and Control

Outline

- Measurement
- Measures
- Direct and Indirect software measures
- Software Measurement
  - Size-oriented Metrics
  - Function-oriented Metrics
  - Object-oriented Metrics
  - Use-case-oriented Metrics

What is Measurement?

- **It is the act of obtaining a measure**
- It is a mechanism of objective evaluation
  - enables managers and practitioners to improve software process
  - assists in planning, tracking and controlling a software project and assessing product quality

What is a Measure?

- It provides a quantitative indication of the size of some product or process attribute

  OR

- quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of product or process

Software Measurement Categories

Direct Measures	Indirect Measures
• Of *software process* are • Cost and effort • Of *product* are • Line of Code (LOC) • Execution Speed • Defects over specified period of time	• Of *product* are • Functionality • Quality • Complexity • Efficiency • Reliability • Maintainability • .. Other Abilities

Size Oriented Metrics

- Derived by normalizing quality and/or productivity measures by considering the *size* of software that has been produced.
- LOC used as a normalizing factor for other measures
- Widely used in software industry but their validity and applicability is debated

Example Set of Size-oriented metrics

- Errors per KLOC (thousand lines of code)
- Defects per KLOC
- $ per LOC
- Pages of Documentation per KLOC
- Errors per person-month
- LOC per person-month
- $ per Page of documentation

Function-oriented Metrics

These use a measure of the functionality delivered by application as a normalization value.

Most widely used function-oriented metric is function point (FP)

FP's computation – based on characteristics of software information domain and complexity.

Function Point

- First proposed by Albrecht; can be used to measure functionality delivered by a system
- Like LOC, FP is also controversial
  - Proponents claim
    - It is prog. Language independent, hence ideal for conventional and non-procedural languages
    - Based on data that can be known early phases of projects
  - Opponents claim
    - Computation is based on subjective rather than objective data
    - counts of information domain may be difficult to collect
    - FP is just a number and has no physical meaning

Computing function points (cont..)

- Following empirical relationship is used to compute Function Point (FP)

$$FP = \text{total count} \times [0.65 + 0.01 \times \Sigma(F_i)]$$

F_i (i=1 to 14) are VAF (Value adjustment factors) or simply ‘adjustment values’

Function Points

- **Inputs**: Data entering the system.
- **Outputs**: Data exiting the system.
- **Inquiries**: User requests for data displayed on screen.
- **Internal Logical Files (ILFs)**: Data maintained within the application.
- **External Interface Files (EIFs)**: Data files accessed by the system but not maintained by it.

- **Q1. Productivity & Cost/KLOC**
  A project delivered **180,000 LOC** in **30 PM** at a total cost of **₹36,00,000**.
  (i) LOC/PM, (ii) Cost per KLOC, (iii) Cost per LOC.

- Q2. **Schedule from Productivity**
  Planned size **48 KLOC**. Org productivity **4 KLOC/PM**. Team size **6** developers.
  Compute PM required and calendar months.

- **Q1.** 6,000 LOC/PM; ₹20,000/KLOC; ₹20/LOC.

- **Q2.** PM = 48/4 = **12 PM**; Months = 12/6 = **2 months**.

- **Worked example (mini library system)**
- **Boundary:** Library system (our app) interacts with a publisher DB (external).
- **Data functions**
- ILF **Books**: DET=12, RET=2 → **Low** → 7
- ILF **Members**: DET=10, RET=1 → **Low** → 7
- ILF **Loans**: DET=8, RET=1 → **Low** → 7
- EIF **Publishers**: DET=7, RET=1 → **Low** → 5
  Subtotal (data) = 7+7+7+5 = **26**
- **Transactional functions**
- EI **Add/Update Book**: FTR=2 (Books, Publishers), DET=10 → **Avg** → 4
- EI **Register Member**: FTR=1, DET=8 → **Low** → 3
- EI **Record Loan**: FTR=3 (Loans, Books, Members), DET=6 → **Avg** → 4
- EO **Overdue Report**: FTR=3, DET=10 → **Avg** → 5
- EQ **Search Catalog**: FTR=2, DET=6 → **Avg** → 4
  Subtotal (txns) = 4+3+4+5+4 = **20**
- **UFP = 26 + 20 = 46**
- Assume DI = 30 ⇒ **VAF = 0.65 + 0.01×30 = 0.95**
  Adjusted FP = 46 × 0.95 = 43.7 ≈ 44 FP

Productivity is 10 hours/FP,

estimated effort =

Object-oriented Metrics

- Used for object-oriented projects
- Set of metrics for OO projects:
  - Number of scenario scripts
    - Scenario scripts are detailed sequence of steps about user and application interaction
    - Scenario scripts are directly correlated to application size and no. of test cases
  - Number of key classes
    - Key classes are independent components
    - They Indicate amount of development effort and potential reuse
  - Number of support classes
    - These indicate amount of development effort and potential reuse
  - Average number of support classes per key class
  - Number of sub systems (aggregation of classes): If identified, it is easier to lay out the schedule

Use-case oriented Metrics

- Use-cases describe user visible functions and features
- They are defined early in software process and can be used as normalization measure before significant activities are initiated
- They are independent of programming language
- No. of use cases directly proportional to
  - size of application in LOC &
  - no. of test cases that will be designed
- There is no standard size of a use case as they are created at different levels of abstraction
  - For this reason, it is a suspect as a normalization measure

Web Engineering Project Metrics



Deliver a combination of **content** and **functionality** to the end user.



Traditional software metrics are difficult to apply directly to WebApps. However, specific **WebApp metrics** help estimate **effort, complexity, and quality**.



Early Estimations Possible:
Metrics can be assessed early in the lifecycle to estimate:

- Effort required
- Defects likely
- Testing focus areas

Customization Index (C):
Let:

- **Nsp** = Number of Static Web Pages
- **Ndp** = Number of Dynamic Web Pages

$$\text{Customization Index (C)} = \frac{Ndp}{Nsp + Ndp}$$

Web Engineering Project Metrics

- **Key WebApp Measures:**

Metric	Description
Static Web Pages (Nsp)	Pages with fixed content, low complexity, low effort.
Dynamic Web Pages (Ndp)	Pages with user/custom-triggered content; higher complexity.
Internal Page Links	Indicate navigational structure and coupling within the WebApp.
Persistent Data Objects	Databases/files accessed; increase data complexity and implementation effort.
External Systems Interfaced	Backroom business application integrations; increase system complexity.
Static Content Objects	Text/images/audio/video that remain unchanged; affect size and loading.
Dynamic Content Objects	Content generated based on user actions; raises content generation complexity.
Executable Functions	Scripts/applets providing services to users; increase modeling and coding work.

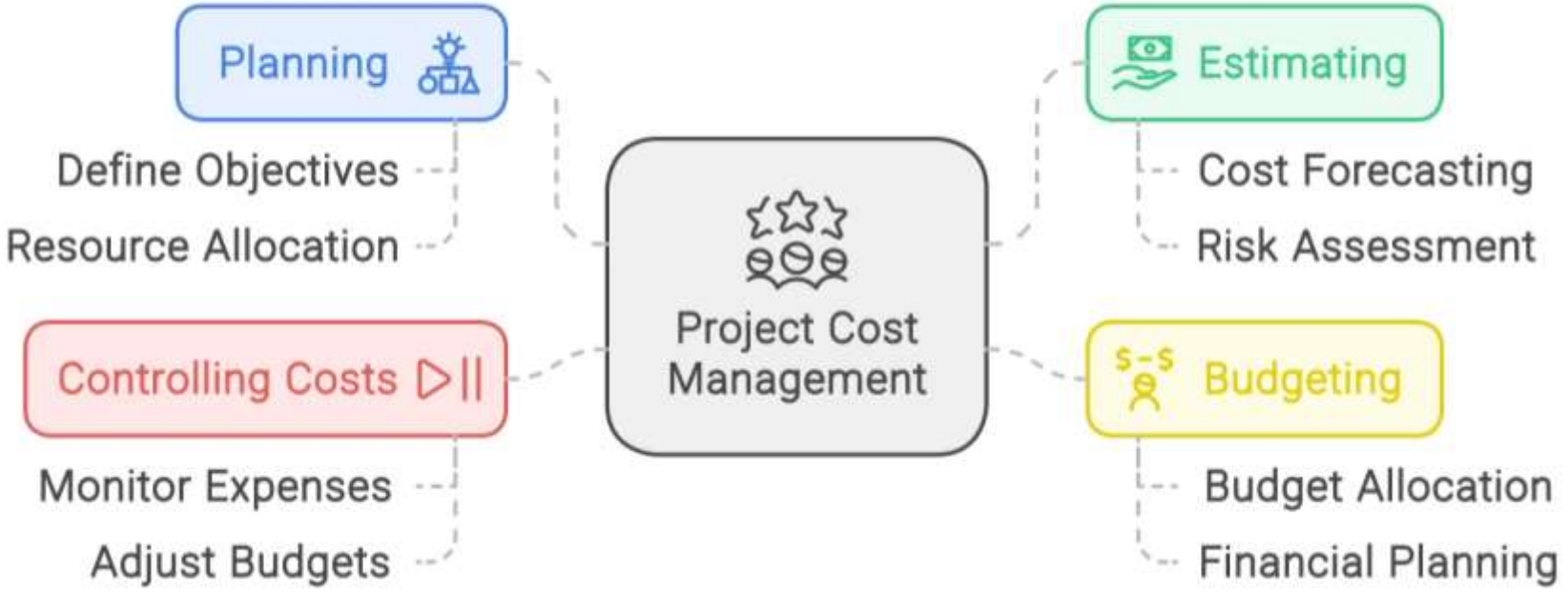


THANK YOU



ANY QUESTIONS

Cost Estimation



Principles of Cost Management

- **Cost Management** involves **planning, estimating, budgeting, financing, funding, managing, and controlling** project costs.

- Objective: Ensure the project is completed within the **approved budget**.

- Importance:
  - Prevents budget overruns
  - Enhances stakeholder trust
  - Enables informed decision-making
  - Improves project success rate



Three Key Processes in Cost Management



1. Cost Estimation

Approximation of project costs (resources, labor, hardware, software, etc.).

2. Cost Budgeting

Aggregating estimated costs to establish a cost baseline.

3. Cost Control

Monitoring cost performance, updating the budget, and controlling changes.

Cost Estimation

- **Cost Estimation** is the process of **approximating the monetary resources** required to complete all project activities.

- Components of Cost Estimation

  - Direct Costs – Developer salaries, hardware, software licenses.
  - Indirect Costs – Electricity, administration, office rent.
  - Contingency Costs – Cost buffer for unforeseen events.
  - Overhead Costs – Company-wide shared costs (e.g., HR, training).

Cost Estimation Techniques

- Algorithmic cost modelling
- Expert judgement
- Estimation by analogy
- Parkinson's Law
- Pricing to win

Algorithmic Cost Modelling

- A formulaic approach based on historical cost information and which is generally based on the size of the software

Expert Judgement

- One or more experts in both software development & application domain use their experience to predict software costs. Process iterates until some consensus is reached.
- Advantages: Relatively cheap estimation method. Can be accurate if experts have direct experience of similar systems
- Disadvantages: Very inaccurate if there are no experts!

Estimation by analogy

- The cost of a project is computed by comparing the project to a similar project in the same application domain
- Advantages: Accurate if project data available
- Disadvantages: Impossible if no comparable project has been tackled. Needs systematically maintained cost database

Parkinson's Law

- The project costs whatever resources are available
- Advantages: No overspend
- Disadvantages: System is usually unfinished

Pricing to win

- The project costs whatever the customer has to spend on it.
- Advantages: You get the contract
- Disadvantages: The probability that the customer gets the system he or she wants is small. Costs do not accurately reflect the work required

Top-down and bottom-up estimation

- Any of these approaches may be used top-down or bottom-up
- Top-down
  - Start at the system level and assess the overall system functionality and how this is delivered through sub-systems
- Bottom-up
  - Start at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate

Top-down estimation

- Usable without knowledge of the system architecture and the components that might be part of the system
- Takes into account costs such as integration, configuration management and documentation
- Can underestimate the cost of solving difficult low-level technical problems

Bottom-up estimation

- Usable when the architecture of the system is known and components identified
- Accurate method if the system has been designed in detail
- May underestimate costs of system level activities such as integration and documentation

Estimation methods

- Each method has strengths and weaknesses
- Estimation should be based on several methods
- If these do not return approximately the same result, there is insufficient information available
- Some action should be taken to find out more in order to make more accurate estimates
- Pricing to win is sometimes the only applicable method

Experience-based estimates

- Estimating is primarily experience-based
- However, new methods and technologies may make estimating based on experience inaccurate
  - Object oriented rather than function-oriented development
  - Client-server systems rather than mainframe systems
  - Off the shelf components
  - Component-based software engineering
  - CASE tools and program generators

Pricing to win

- This approach may seem unethical & unbusinesslike
- However, when detailed information is lacking it may be the only appropriate strategy
- The project cost is agreed on the basis of an outline proposal and the development is constrained by that cost
- A detailed specification may be negotiated or an evolutionary approach used for system development

Project Estimation Techniques

The major project planning phase include

Finding project parameters.

- **Project size:** What would be the size of the code written say, in number of lines, files, modules?
- **Cost:** How much would it cost to develop a software? A software may be just pieces of code, but one has to pay to the managers, developers, and other project personnel.

Cont…

- **Duration:** How long would it be before the software is delivered to the clients?
- **Effort:** How much effort from the team members would be required to create the software?

Algorithmic Cost modelling

- Cost is estimated as a mathematical function of product, project and process attributes whose values are estimated by project managers
  - Effort = A * $Size^{B}$ * M
  - A is an organisation-dependent constant, B reflects the disproportionate effort for large projects and M is a multiplier reflecting product, process and people attributes
- Most commonly used product attribute for cost estimation is code size
- Most models are basically similar but with different values for A, B and M

Estimation accuracy

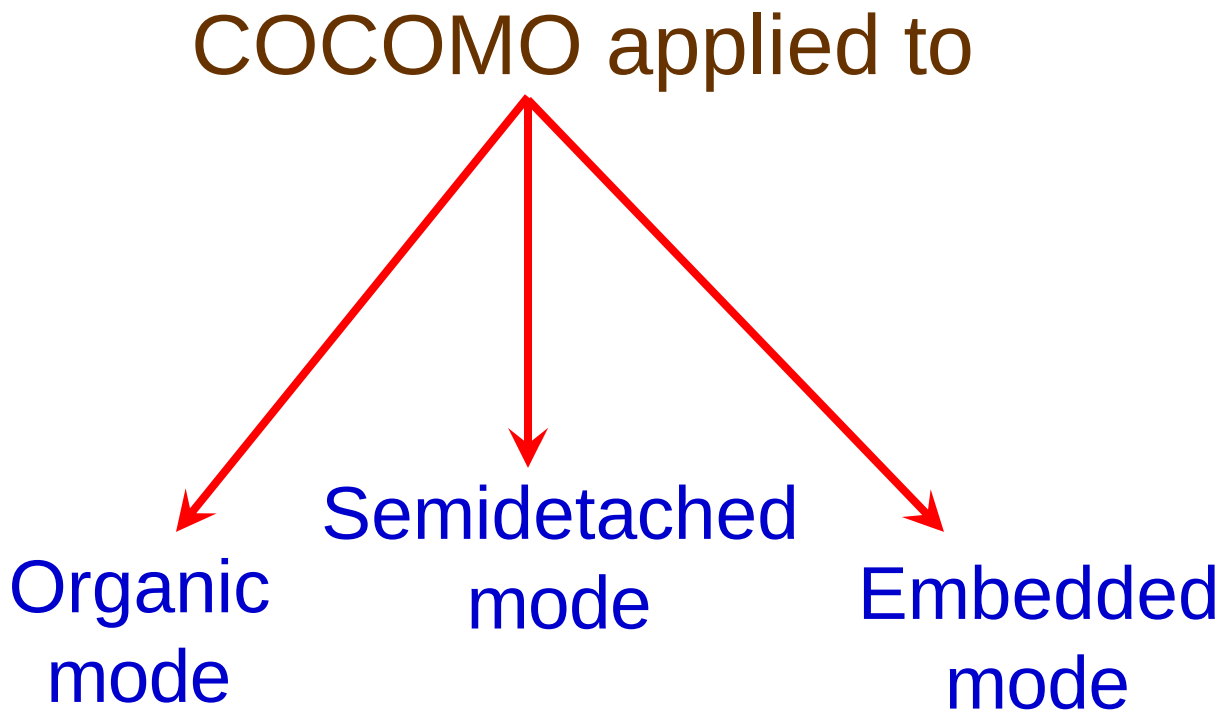
- The size of a software system can only be known accurately when it is finished
- Several factors influence the final size
  - Use of COTS and components
  - Programming language
  - Distribution of system
- As the development process progresses then the size estimate becomes more accurate

COCOMO model

- The **COstructive COst Model** (COCOMO) is the most widely used software estimation model.
- The COCOMO model predicts the **effort** and **duration** of a project based on inputs relating to the size of the resulting systems and a number of "**cost drives**" that affect productivity.

COCOMO Models

- COCOMO is defined in terms of three different models:
  - the **Basic model**,
  - the **Intermediate model**, and
  - the **Detailed model**.
- The more complex models account for more factors that influence software projects, and make more accurate estimates.


COCOMO applied to
Organic mode
Semidetached mode
Embedded mode

Software Project Planning

Mode	Project size	Nature of Project	Innovation	Deadline of the project	Development Environment
Organic	Typically 2-50 KLOC	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached	Typically 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/ customer Interfaces required

The comparison of three COCOMO modes

What each coefficient means

- $a \rightarrow$ **Effort multiplier**
  - Base factor that sets the overall scale of effort.
  - Larger a= more effort per KLOC.
- $b \rightarrow$ **Effort exponent**
  - Controls how effort grows as project size increases.
  - If $b > 1$, effort grows faster than linear with size (captures complexity).

Software Project Planning

Basic Model

Basic COCOMO model takes the form

$$E = a_b (KLOC)^{b_b}$$

$$D = c_b (E)^{d_b}$$

where E is effort applied in Person-Months, and D is the development time in months. The coefficients a_b, b_b, c_b and d_b are given in table 4 (a).

Software Project Planning

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Basic COCOMO coefficients

Software Project Planning

When effort and development time are known, the average staff size to complete the project may be calculated as:

Average staff size $$(SS) = \frac{E}{D} Persons$$

When project size is known, the productivity level may be calculated as:

Productivity $$(P) = \frac{KLOC}{E} KLOC / PM$$

Software Project Planning

Example:

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes i.e., organic, semidetached and embedded.

Software Project Planning

Solution

The basic COCOMO equation take the form:

$$E = a_b (KLOC)^{b_b}$$

$$D = c_b (KLOC)^{d_b}$$

Estimated size of the project = 400 KLOC

(i) Organic mode

E = $2.4(400)^{1.05}$ = 1295.31 PM

D = $2.5(1295.31)^{0.38}$ = 38.07 PM

Software Project Planning

(ii) Semidetached mode

$E = 3.0(400)^{1.12} = 2462.79$ PM

$D = 2.5(2462.79)^{0.35} = 38.45$ PM

(iii) Embedded mode

$E = 3.6(400)^{1.20} = 4772.81$ PM

$D = 2.5(4772.8)^{0.32} = 38$ PM

Software Project Planning

Example:

A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project.

Software Project Planning

Solution

The semi-detached mode is the most appropriate mode; keeping in view the size, schedule and experience of the development team.

Hence $E = 3.0(200)^{1.12} = 1133.12$ PM

$D = 2.5(1133.12)^{0.35} = 29.3$ PM

Average staff size $(SS) = \frac{E}{D} Persons$

$$= \frac{1133.12}{29.3} = 38.67 Persons$$

Software Project Planning

Productivity $= \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765\, KLOC / PM$

$$P = 176\ LOC / PM$$

- $c \rightarrow$ **Time multiplier**
  - Base factor that sets the overall scale of development time.
  - Usually **2.5** for all three modes.
- $d \rightarrow$ **Time exponent**
  - Determines how development time grows with project effort.

Software Project Planning

Intermediate Model

Cost drivers

(i) Product Attributes

- Required s/w reliability
- Size of application database
- Complexity of the product

(ii) Hardware Attributes

- Run time performance constraints
- Memory constraints
- Virtual machine volatility
- Turnaround time

Software Project Planning

(*iii*) Personal Attributes

- Analyst capability
- Programmer capability
- Application experience
- Virtual m/c experience
- Programming language experience

(*iv*) Project Attributes

- Modern programming practices
- Use of software tools
- Required development Schedule

Software Project Planning

BENNETT UNIVERSITY
THE TIMES GROUP

Multipliers of different cost drivers

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	--
DATA	--	0.94	1.00	1.08	1.16	--
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME	--	--	1.00	1.11	1.30	1.66
STOR	--	--	1.00	1.06	1.21	1.56
VIRT	--	0.87	1.00	1.15	1.30	--
TURN	--	0.87	1.00	1.07	1.15	--

Software Project Planning

BENNETT UNIVERSITY
THE TIMES GROUP

Cost Drivers	RATINGS					
	Very low	Low	Nominal	High	Very high	Extra high
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	--
AEXP	1.29	1.13	1.00	0.91	0.82	--
PCAP	1.42	1.17	1.00	0.86	0.70	--
VEXP	1.21	1.10	1.00	0.90	--	--
LEXP	1.14	1.07	1.00	0.95	--	--
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	--
TOOL	1.24	1.10	1.00	0.91	0.83	--
SCED	1.23	1.08	1.00	1.04	1.10	--

Multiplier values for effort calculations

Software Project Planning

Intermediate COCOMO equations

$$E = a_i (KLOC)^{b_i} * EAF$$

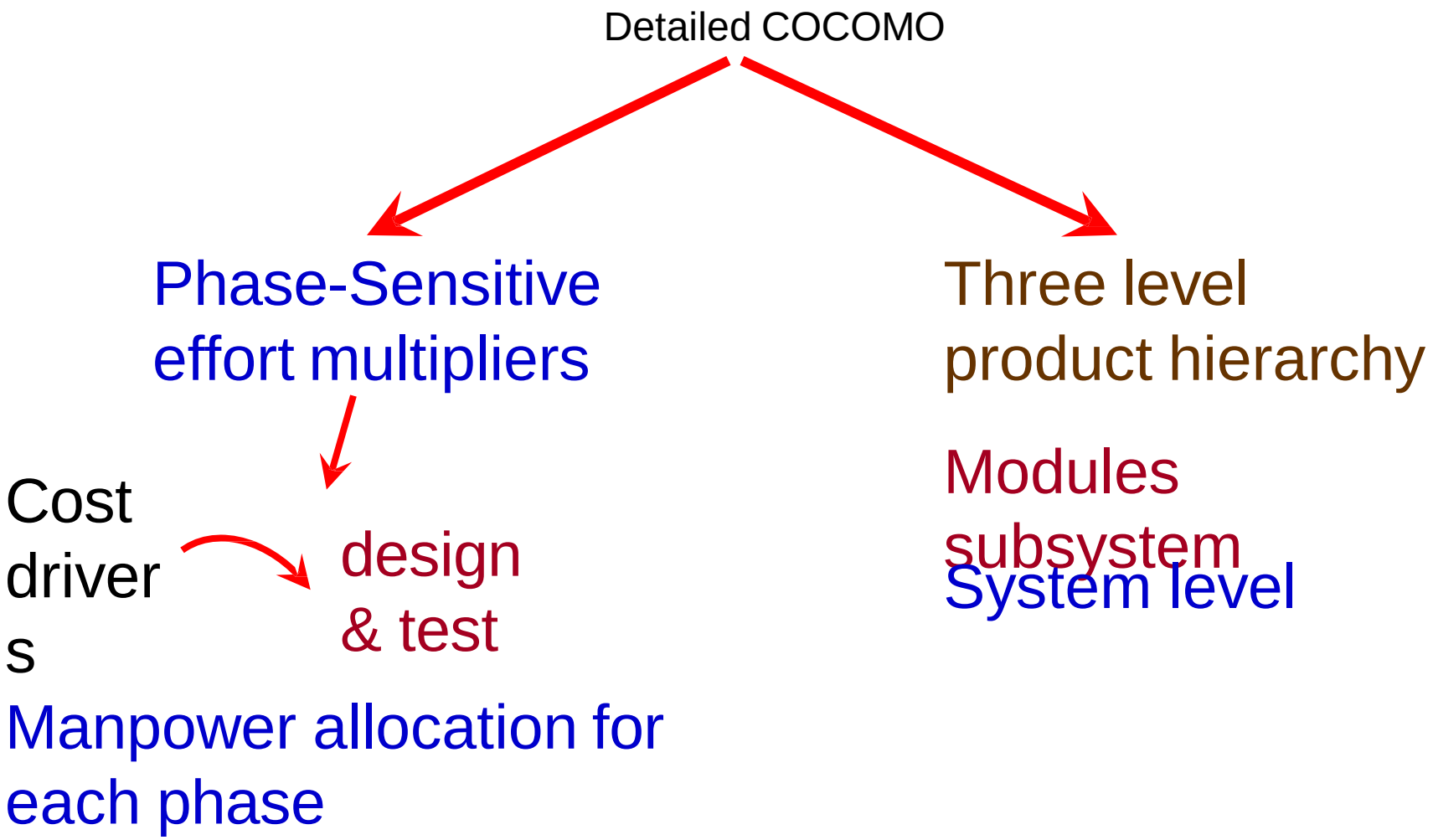
$$D = c_i (E)^{d_i}$$

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Coefficients for intermediate COCOMO

Software Project Planning

Detailed COCOMO Model



Software Project Planning

Development Phase

Plan / Requirements

EFFORT	:	6% to 8%
DEVELOPMENT TIME	:	10% to 40%

% depend on mode & size

Software Project Planning

Design

Effort	:	16% to 18%
Time	:	19% to 38%

Programming

Effort	:	48% to 68%
Time	:	24% to 64%

Integration & Test

Effort	:	16% to 34%
Time	:	18% to 34%

Principle of the effort estimate

Size equivalent

As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design document (DD%), code (C%) and integration (I%) to be modified are estimated. Then, an adjustment factor, A, is calculated by means of the following equation.

A = 0.4 DD + 0.3 C + 0.3 I

The size equivalent is obtained by

S (equivalent) = (S x A) / 100

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

Software Project Planning

BENNETT UNIVERSITY
THE TIMES GROUP

Lifecycle Phase Values of μ_p

Mode & Code Size	Plan & Requirements	System Design	Detailed Design	Module Code & Test	Integration & Test
Organic Small S≈2	0.06	0.16	0.26	0.42	0.16
Organic medium S≈32	0.06	0.16	0.24	0.38	0.22
Semidetached medium S≈32	0.07	0.17	0.25	0.33	0.25
Semidetached large S≈128	0.07	0.17	0.24	0.31	0.28
Embedded large S≈128	0.08	0.18	0.25	0.26	0.31
Embedded extra large S≈320	0.08	0.18	0.24	0.24	0.34

Effort and schedule fractions occurring in each phase of the lifecycle

Software Project Planning

BENNETT UNIVERSITY
THE TIMES GROUP

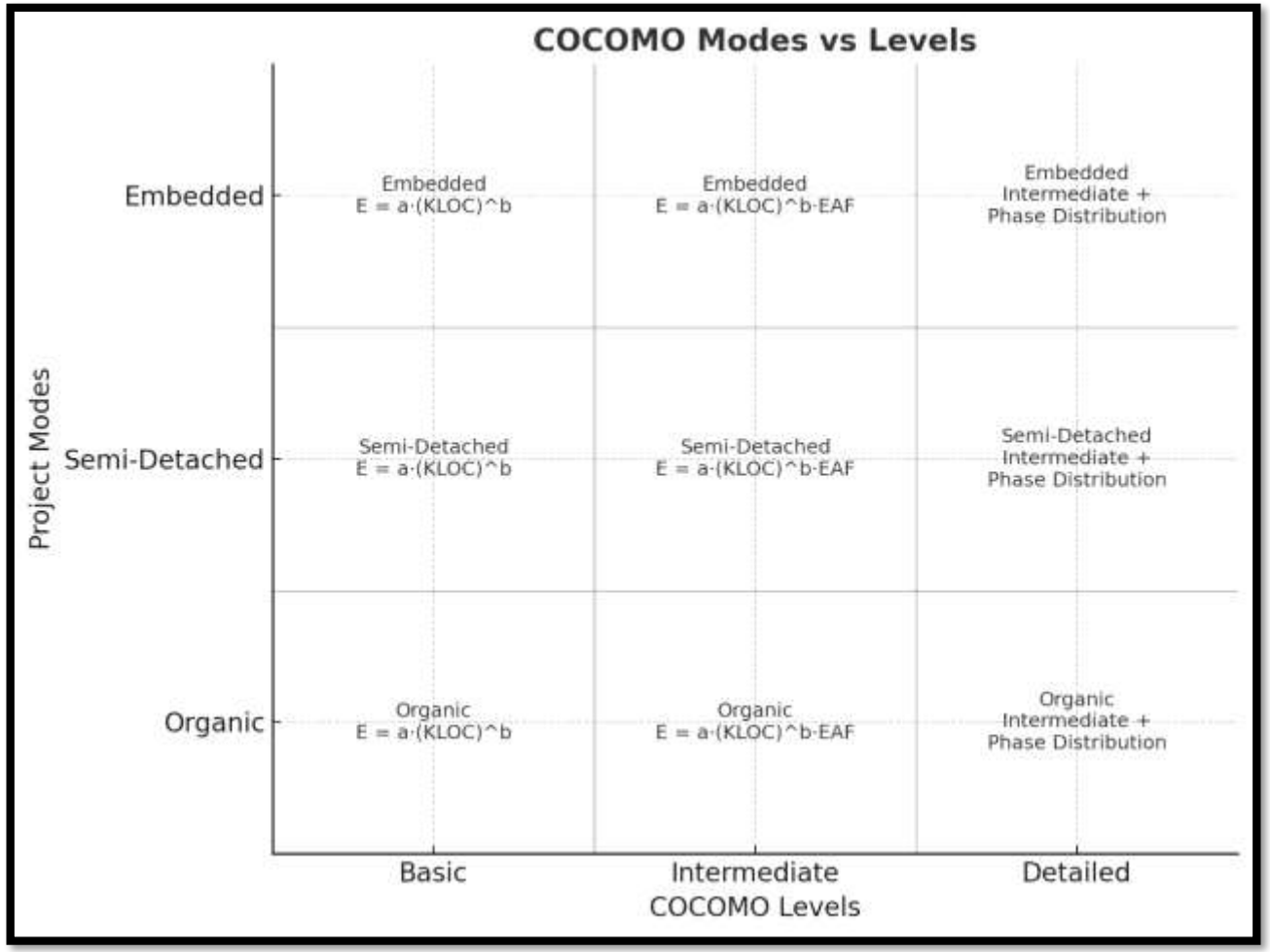
Lifecycle Phase Values of τ_p

Mode & Code Size	Plan & Requirements	System Design	Detailed Design	Module Code & Test	Integration & Test
Organic Small S≈2	0.10	0.19	0.24	0.39	0.18
Organic medium S≈32	0.12	0.19	0.21	0.34	0.26
Semidetached medium S≈32	0.20	0.26	0.21	0.27	0.26
Semidetached large S≈128	0.22	0.27	0.19	0.25	0.29
Embedded large S≈128	0.36	0.36	0.18	0.18	0.28
Embedded extra large S≈320	0.40	0.38	0.16	0.16	0.30

Effort and schedule fractions occurring in each phase of the lifecycle

Organic / Semi-Detached / Embedded = **modes** (types of projects).
•**Basic / Intermediate / Detailed** = **levels** (how much detail you include).
•They are **not separate models**, but **cross-linked**:
•A project can be *semi-detached* and estimated with *basic* COCOMO, or with *intermediate* COCOMO, or with *detailed* COCOMO.

COCOMO modes vs levels



Software Project Planning

Distribution of software life cycle:

1. Requirement and product design
   (a) Plans and requirements
   (b) System design

2. Detailed Design
   (a) Detailed design

3. Code & Unit test
   (a) Module code & test

4. Integrate and Test
   (a) Integrate & Test

Software Project Planning

Example:

A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers: Very highly capable with very little experience in the programming language being used

Or

Developers of low quality but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool ?

Software Project Planning

Solution

This is the case of embedded mode and model is intermediate COCOMO.

Hence $E = a_i (KLOC)^{d_i}$

$= 2.8\ (400)^{1.20} = 3712$ PM

Case I: Developers are very highly capable with very little experience in the programming being used.

EAF = 0.82 x 1.14 = 0.9348

E = 3712 x .9348 = 3470 PM

D = $2.5\ (3470)^{0.32}$ = 33.9 M

Developers are of low quality → This maps to the **Personnel Capability (PCAP)** cost driver.

•Low PCAP multiplier ≈ **1.29** (meaning: more effort required).

•**Developers have a lot of experience with the programming language** → This maps to the **Language Experience (LEXP)** cost driver.

•High LEXP multiplier ≈ **0.95** (meaning: less effort required).

Developers are **low quality** → Personnel Capability (PCAP) multiplier =

•Developers have **high programming language experience** → Language Experience (LEXP) multiplier

Software Project Planning

Case II: Developers are of low quality but lot of experience with the programming language being used.

EAF = 1.29 x 0.95 = 1.22

E = 3712 x 1.22 = 4528 PM

D = $2.5 (4528)^{0.32}$ = 36.9 M

Case II requires more effort and time. Hence, low quality developers with lot of programming language experience could not match with the performance of very highly capable developers with very litter experience.

Software Project Planning

Example:

Consider a project to develop a full screen editor. The major components identified are:

I. Screen edit

II. Command Language Interpreter

III. File Input & Output

IV. Cursor Movement

V. Screen Movement

The size of these are estimated to be 4k, 2k, 1k, 2k and 3k delivered source code lines. Use COCOMO to determine

1. Overall cost and schedule estimates (assume values for different cost drivers, with at least three of them being different from 1.0)
2. Cost & Schedule estimates for different phases.

Software Project Planning

Solution

Size of five modules are:

Screen edit	= 4 KLOC
Command language interpreter	= 2 KLOC
File input and output	= 1 KLOC
Cursor movement	= 2 KLOC
Screen movement	= 3 KLOC
Total	**= 12 KLOC**

Software Project Planning

Let us assume that significant cost drivers are

i. Required software reliability is high, i.e.,1.15

ii. Product complexity is high, i.e.,1.15

iii. Analyst capability is high, i.e.,0.86

iv. Programming language experience is low,i.e.,1.07

v. All other drivers are nominal

EAF = 1.15x1.15x0.86x1.07 = 1.2169

Software Project Planning

(a) The initial effort estimate for the project is obtained from the following equation

$$E = a_i (KLOC)^{bi} \times EAF$$

$$= 3.2(12)^{1.05} \times 1.2169 = 52.91 \text{ PM}$$

Development time $D = C_i(E)^{di}$

$$= 2.5(52.91)^{0.38} = 11.29 \text{ M}$$

(b) Using the following equations and referring Table 7, phase wise cost and schedule estimates can be calculated.

$$E_p = \mu_p E$$

$$D_p = \tau_p D$$

Software Project Planning

Since size is only 12 KLOC, it is an organic small model. Phase wise effort distribution is given below:

System Design	= 0.16 x 52.91 = 8.465 PM
Detailed Design	= 0.26 x 52.91 = 13.756 PM
Module Code & Test	= 0.42 x 52.91 = 22.222 PM
Integration & Test	= 0.16 x 52.91 = 8.465 Pm

Now Phase wise development time duration is

System Design	= 0.19 x 11.29 = 2.145 M
Detailed Design	= 0.24 x 11.29 = 2.709 M
Module Code & Test	= 0.39 x 11.29 = 4.403 M
Integration & Test	= 0.18 x 11.29 = 2.032 M

Software Project Management

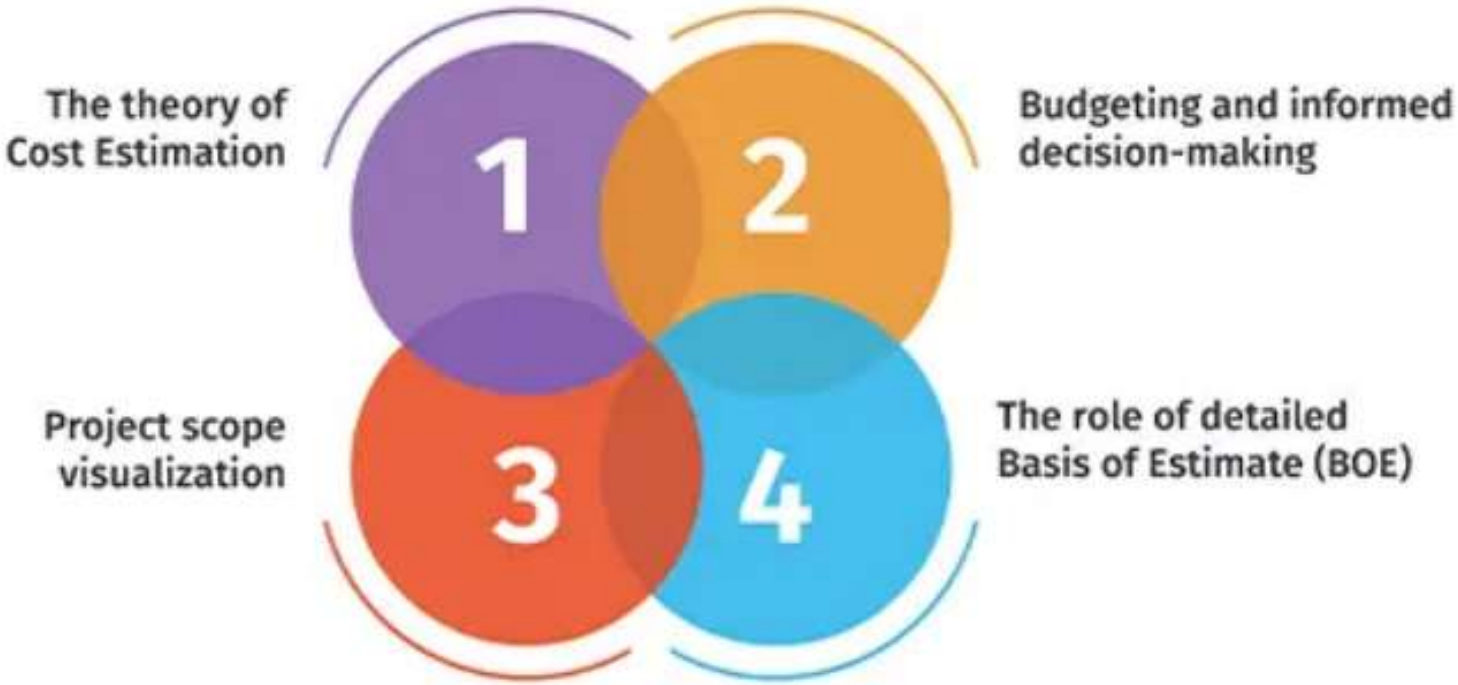
(CSET212)

Surbhi Sharma
(Assistant Professor, SCSET)

Contents

- Principles of Cost Management,
- Cost Estimating,
- Types of Estimates,
- Estimation Tools & Techniques

Principles of Cost Management



Introduction to Cost Management

Involves the following to complete the project within the approved budget:

- planning,
- estimating,
- budgeting,
- financing,
- funding,
- managing, and
- controlling costs

Cost Planning

- Resource requirement analysis
- Cost of tools, licenses, and infrastructure
- Team size and their hourly rates
- Timeline and workload balancing
- **Example:**
  If a project requires 5 developers for 4 months at ₹60,000/month, then:

Total developer cost = 5 × 4 × ₹60,000 = ₹12,00,000

Cost Estimating

Cost Estimating

- Cost estimating is the process of approximating the monetary resources required to complete all project activities.
- **Purpose:**
  - To prepare realistic budgets
  - To evaluate financial feasibility
  - To form the basis for cost control and scheduling

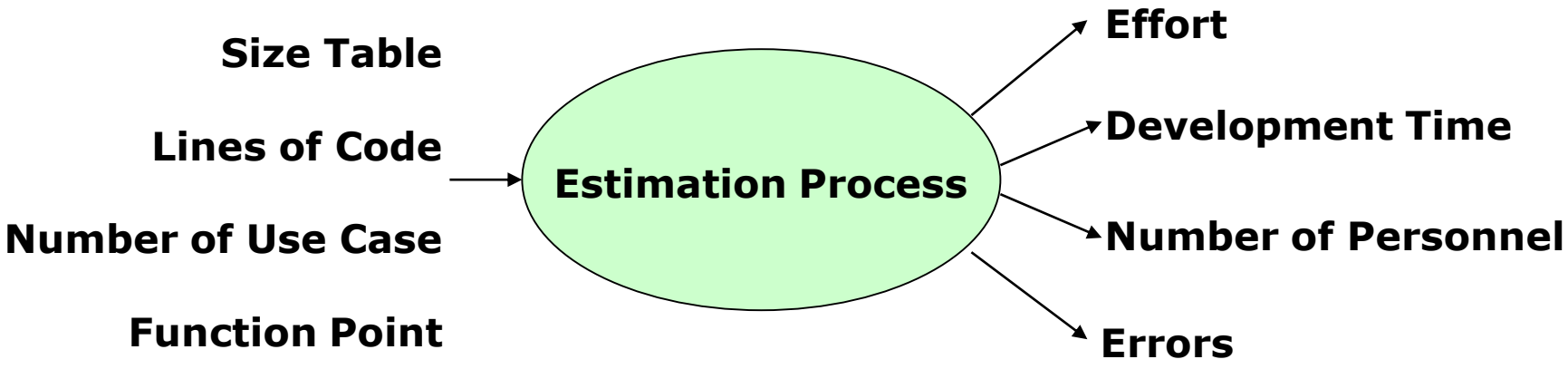
Objectives of Cost Estimating

- Quantify costs of resources, labor, tools, licenses, and infrastructure
- Define a baseline for project budgeting
- Anticipate financial risks through buffer and reserve estimates
- Aid in funding approvals and resource allocations

Why Cost Estimation is Important

- Save time and money, Select the most profitable project, and Ensures smoother project controls management
- **Solid project planning**
- **Improved resource allocation**
- **Accurate budgeting and control**
- **Risk mitigation**
- **Competitive bidding**
- **Project transparency**

Cost Estimation Process



Effort

- Effort Equation
  - **PM = C * (KDSI)^{n} (**person-months)
    - where **PM** = number of person-month (=152 working hours),
    - **C** = a constant,
    - **KDSI** = thousands of "delivered source instructions" (DSI) and
    - **n** = a constant.

Productivity

- Productivity equation
  - **(DSI) / (PM)**
    - where **PM** = number of person-month (=152 working hours),
    - **DSI** = "delivered source instructions"

Schedule

- Schedule equation
  - **TDEV = C * (PM)^{n} (**months)
    - where TDEV = number of months estimated for software development.

Average Staffing

- Average Staffing Equation
  - **(PM) / (TDEV)** **(**FSP)
    - where FSP means Full-time-equivalent Software Personnel.

Cost Estimation Process

Cost = SizeOfTheProject x Productivity

Inputs to Cost Estimation

Input	Description
Project Scope Statement	Defines the features and boundaries of work
Work Breakdown Structure	Hierarchical decomposition of tasks
Resource Requirements	Skills, effort, tools, and timelines
Historical Data	Past projects' cost and schedule
Risk Register	Cost-related risk items and their contingencies
Enterprise Environmental Factors	Cost indices, exchange rates, market rates

Cost Categories

Category	Description	Examples
Direct Cost	Attributable directly to the project	Salaries, software licenses
Indirect Cost	Overheads shared across projects or departments	Rent, electricity, HR
Recurring	Repeat over the duration of the project	Developer wages, hosting
Non-Recurring	One-time expenses	Equipment purchase, training
Fixed	Do not vary with project size	Contract fees
Variable	Scale with project scope	Travel, cloud compute usage

Types of Cost Estimates

Types of Cost Estimates

- Cost estimates in software projects refer to the approximations of monetary resources required at different phases of the project lifecycle. Based on project progress and available data, **different types of estimates** are used with varying levels of accuracy.
- Major Types of Cost Estimates
  - Rough Order of Magnitude (ROM) Estimate
  - Budget Estimate
  - Definitive Estimate

Rough Order of Magnitude (ROM) Estimate

- **Also Known As:** Ballpark Estimate
  Purpose: For initial feasibility or bidding
  Accuracy Range: ±75% (very approximate)
  When Used: Project initiation phase (before detailed requirements)
- **Example:**
  - Based on historical data, a similar project cost ₹10 lakhs
  - ROM Estimate = ₹10L ± 75%
  - Range: ₹2.5L to ₹17.5L
- **Use Case:** “Should we pursue this project opportunity?”

Budget Estimate

- **Purpose:** For intermediate planning and partial approval
  Accuracy Range: ±25%
  When Used: After requirements and WBS (Work Breakdown Structure) are defined
- **Example:**
  - Initial estimates suggest ₹12,00,000 for development
  - Budget Estimate = ₹12L ± 25% → ₹9L to ₹15L
- **Use Case:** “How much should we request as a budget?”

Definitive Estimate

- **Purpose:** For final cost baseline and procurement
  Accuracy Range: ±10%
  When Used: After scope, design, and resource plans are fully finalized
- **Example:**
- Cost calculation:
  - UI/UX: ₹80,000
  - Development: ₹6,00,000
  - Testing: ₹1,20,000
  - Deployment: ₹50,000
- **Definitive Estimate = ₹8,50,000 ± 10%** → ₹7,65,000 to ₹9,35,000
- **Use Case:** “Lock the budget and prepare for execution.”

Estimate Type	Accuracy	Timing	Purpose
Rough Order of Magnitude (ROM)	±75%	Project Initiation	Go/no-go decisions, proposals
Budget Estimate	±25%	Planning Phase	Budget approval, forecasting
Definitive Estimate	±10%	Execution Preparation	Contracting, final cost baseline

Estimation Tools & Techniques

Estimation Tools & Techniques

- Estimation techniques are methods used to forecast the cost, duration, and resources required to complete project tasks.
- Estimation forms the foundation of **budgeting**, **scheduling**, and **resource allocation** in software project management.

Estimation Techniques

- Expert Judgment
- Analogous Estimating
- Parametric Estimating
- Bottom-Up Estimating
- Three-Point Estimating (PERT Method)
- Reserve Analysis

Expert Judgment

- Involves consulting experienced project managers, developers, or domain experts for effort estimation.
- **Use Case:**
  When there is no historical data or similar project available.
- **Example:**
  A senior developer estimates a module will take 40 hours based on previous similar work.
- **Limitation:**
  Subjective and prone to bias.

Analogous Estimating

- Uses actual values from a **previous, similar project** as a reference for the current one.
- **Example:**
  If a previous project took 3 months for 5 screens, and the current one needs 10 screens → Estimated time = **6 months**
- **Pros:** Quick and low effort
  Cons: Low accuracy if similarity is weak

Parametric Estimating

Uses statistical relationships between historical data and project variables.

Formula:

Effort = Rate × Quantity

Example:

- Cost per UI screen = ₹5,000
- Project has 20 screens

Total cost = ₹5,000 × 20 = ₹1,00,000

Advantage: More accurate if historical data is strong

Limitation: Assumes linear scalability

Bottom-Up Estimating

- Each work package or task is estimated individually, then rolled up to get total.
- **Steps:**
  - Break the project into tasks (WBS)
  - Estimate effort per task
  - Aggregate the total
- **Useful For:** High-accuracy budgeting
  Drawback: Time-consuming

Task	Hours
UI Design	30
Backend Development	80
Testing	40
Total	**150** hrs

Three-Point Estimating (PERT Method)

- Program Evaluation and Review Technique estimates based on uncertainty using three scenarios:
  - Optimistic time (O)- Minimum possible time (best-case)
  - Most Likely time(M)- Normal/expected time
  - Pessimistic time (P)- Maximum possible time (worst-case)
- **Formula:**
  Expected Time (TE) = (O + 4M + P) / 6
- Example:
  - O = 10 days, M = 14 days, P = 22 days
  - TE = (10 + 4×14 + 22)/6 = (10 + 56 + 22)/6 = **14.67 days**
- **Benefit:** Considers risk and uncertainty

Reserve Analysis

- Adds contingency reserves (time/money) for unknown risks.
- **Types:**
  - Contingency Reserve: Known-unknowns (~10–15%)
  - Management Reserve: Unknown-unknowns (~5%)

  Example:
  Estimated Cost = ₹5,00,000
  Contingency Reserve (10%) = ₹50,000
  Management Reserve (5%) = ₹25,000
  Total Budget = ₹5,75,000

Technique	Accuracy	Time Required	Suitable For
Expert Judgment	Medium	Low	Early-stage estimates
Analogous Estimation	Low	Low	Projects with history
Parametric Estimation	High	Medium	Repetitive projects
Bottom-Up Estimation	Very High	High	Mature planning phase
Three-Point Estimation	High	Medium	Risk-uncertain tasks
Reserve Analysis	N/A	Low	Budget buffering

Software Project Management

Surbhi Sharma
(Assistant Professor, SCSET)

Contents

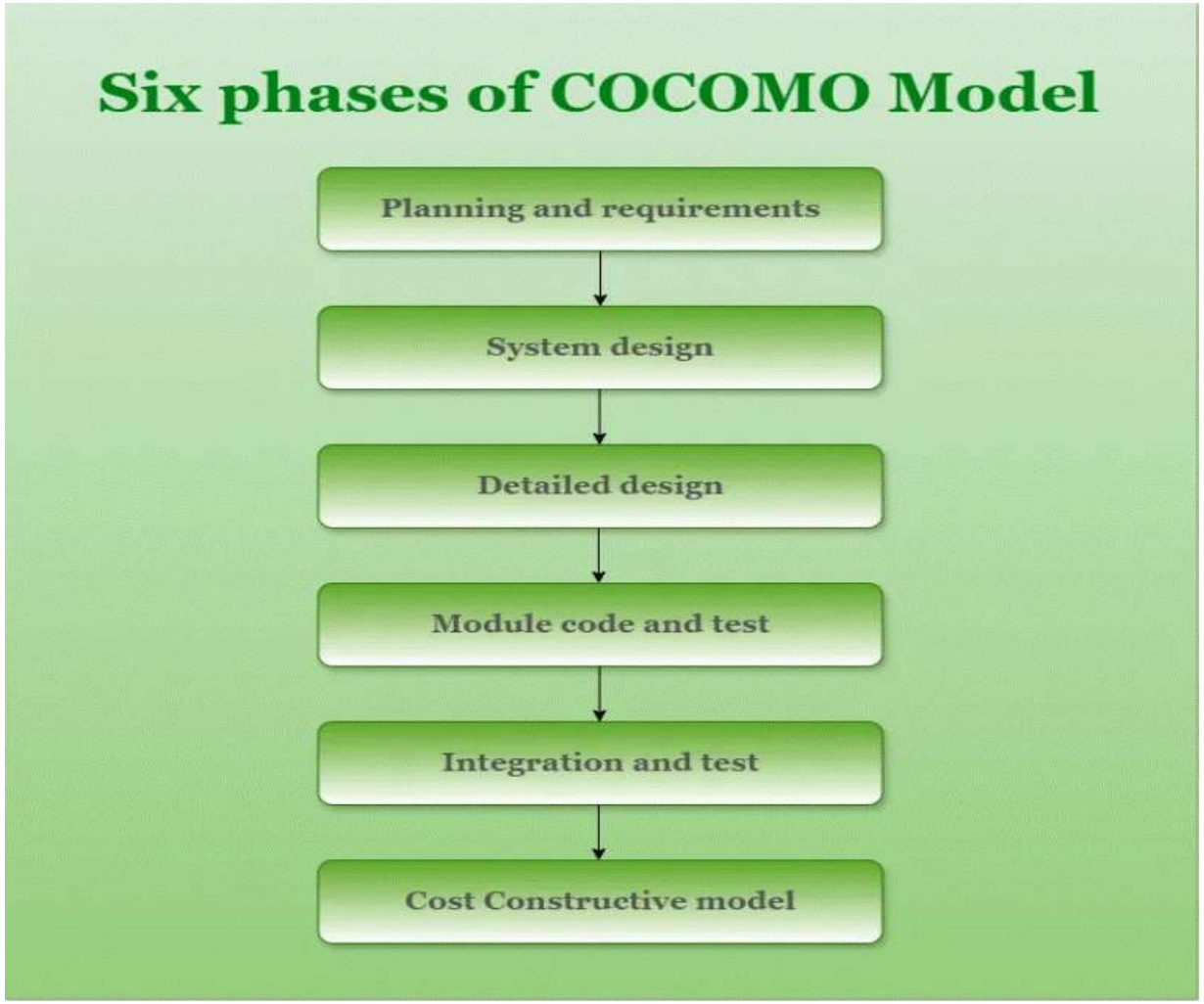
- COCOMO Model

COCOMO Model

The Constructive Cost Model (COCOMO)

- COCOMO is one of the most widely used software estimation models in the world
- It was developed by Barry Boehm in 1981
- COCOMO predicts the effort and schedule for a software product development based on inputs relating to the **size** of the software and a number of **cost drivers** that affect productivity

Six Phases of COCOMO Model



COCOMO Models

- COCOMO has three different models that reflect the **complexity**:
  - the Basic Model
  - the Intermediate Model
  - and the Detailed Model

Basic Model

- Applicable to small to medium sized software projects
- Use for a quick and rough estimates
- Three modes of software development are considered
  - Organic
  - Semi-detached
  - Embedded

Aspects	Organic	Semidetached	Embedded
Project Size	2 to 50 KLOC	50-300 KLOC	300 and above KLOC
Complexity	Low	Medium	High
Team Experience	Highly experienced	Some experienced as well as inexperienced staff	Mixed experience, includes experts
Environment	Flexible, fewer constraints	Somewhat flexible, moderate constraints	Highly rigorous, strict requirements
Effort Equation	$E = 2.4(400)^{1.05}$	$E = 3.0(400)^{1.12}$	$E = 3.6(400)^{1.20}$
Example	Simple payroll system	New system interfacing with existing systems	Flight control software

Basic Model

- The **Basic COCOMO model** computes effort as a function of program size.

The formulae to calculate these entities are:

$$Effort(E) = a(kLOC)^b$$

$$Time(T) = c(E)^d$$

$$People\ required = \frac{E}{T}$$

The effort is measured in person-months and time in months. The constants $a, b, c, and\ d$ vary for each model type. The following are the constant values for the basic model:

Mode	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

Parameter	Organic	Semi-detached	Embedded
C	2.5	2.5	2.5
D	0.38	0.35	0.32

Basic COCOMO Model: Equation

Mode	***Effort***	***Schedule***
Organic	$E=2.4*(KDSI)^{1.05}$	$TDEV=2.5*(E)^{0.38}$
Semidetached	$E=3.0*(KDSI)^{1.12}$	$TDEV=2.5*(E)^{0.35}$
Embedded	$E=3.6*(KDSI)^{1.20}$	$TDEV=2.5*(E)^{0.32}$

Basic COCOMO Model: Example

E.g. : We have determined our project fits the characteristics of **Semi-Detached** mode

- We estimate our project will have **32,000** Delivered Source Instructions. Using the formulas, we can estimate:
- **Effort** = $3.0*(32)^{1.12}$ = 146 man-months
- **Schedule** = $2.5*(146)^{0.35}$ = 14 months
- **Productivity** = 32,000 DSI / 146 MM
  = 219 DSI/MM
- **Average Staffing** = 146 MM /14 months
  = 10 FSP

Example

Suppose a project was estimated to be made in 400 kLOC. Lets calculate its effort, time, and the number of people required while considering the project is of organic type:

$$Effort(E) = 2.4(400\ kLOC)^{1.05} = 1295.31\ person - months$$

$$Time(T) = 2.5(1295.31)^{0.38} = 30.07\ months$$

$$People\ required = \frac{1295.31}{30.07} = 43.07\ persons$$

Intermediate Model

- The intermediate model is an extension of the basic model and includes a set of cost drivers to calculate the estimates with better accuracy. The effort factor includes the effort adjustment factor (EAF) that is calculated with the cost drivers

$$Effort(E) = a(kLOC)^b * EAF$$

$$Time(T) = c(E)^d$$

The effort is measured in person-months and time in months. The constants $a, b, c, and\ d$ vary for each model type. The following are the constant values for the basic model:

Project Type	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Cost drivers fall under 4 categories:

- **Product attributes** (e.g., complexity, database size)
- **Hardware attributes** (e.g., memory constraints)
- **Personnel attributes** (e.g., experience)
- **Project attributes** (e.g., use of tools)

Cost Drivers	Very Low	Low	Nominal	High	Very High
Product Attributes					
Required Software Reliability	0.75	0.88	1.00	1.15	1.40
Size of Application Database		0.94	1.00	1.08	1.16
Complexity of The Product	0.70	0.85	1.00	1.15	1.30
Hardware Attributes					
Runtime Performance Constraints			1.00	1.11	1.30
Memory Constraints			1.00	1.06	1.21
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30
Required turnabout time		0.94	1.00	1.07	1.15
Personnel attributes					
Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	
Project Attributes					
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82
Use of software tools	1.24	1.10	1.00	0.91	0.83
Required development schedule	1.23	1.08	1.00	1.04	1.10

- The project manager is to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, appropriate cost driver values are taken from the above table. These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor).

Example: Suppose a project was estimated to be made in 400 kLOC. let's calculate its effort, time, and the number of people required while considering the project is of organic type and has a nominal complexity. The developer has a high virtual machine experience. The value of the nominal complexity of a project is 1.00, and the high virtual experience of the developer is 0.90, according to the values mentioned in table:

$$EAF = 1.00 * 0.90 = 0.9$$

$$Effort(E) = 3.2(400\ kLOC)^{1.05} * 0.9 = 1554.37\ person - months$$

$$Time(T) = 2.5(1554.37)^{0.38} = 40.80\ months$$

$$People\ required = \frac{1554.37}{40.80} = 38.09\ persons$$

Detailed COCOMO

- **Detailed COCOMO** incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process.
- The detailed model uses different effort multipliers for each cost driver attribute.
- In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

Typical Phase-wise Distribution of Effort

Phase	% of Total Effort
Requirements & Planning	6%
System Design	16%
Detailed Design	26%
Coding & Unit Testing	42%
Integration & Testing	16%

Model	Inputs Required	Complexity	Use Case
Basic COCOMO	KLOC only	Simple	Quick estimates
Intermediate COCOMO	KLOC + 15 cost drivers	Medium	More accurate estimate
Detailed COCOMO	KLOC + cost drivers + phase-wise details	High	Full project planning

Introduction to AGILE

Discussion Points

- *History*
- *What is Agile*
- *How does it works*
- *Why we use Agile Methods*
- *Agile Manifesto*
- *Agile*
- *Methodologies*

  Scrum

History

- In 2001, this new management paradigm began to pick up momentum, agile was formalized when 17 pioneers of the agile methodology met at the Snowbird Ski Resort in Utah and issued the Agile Manifesto.

1.Kent Beck 2.Mike Beedle 3. Arie van Bennekum

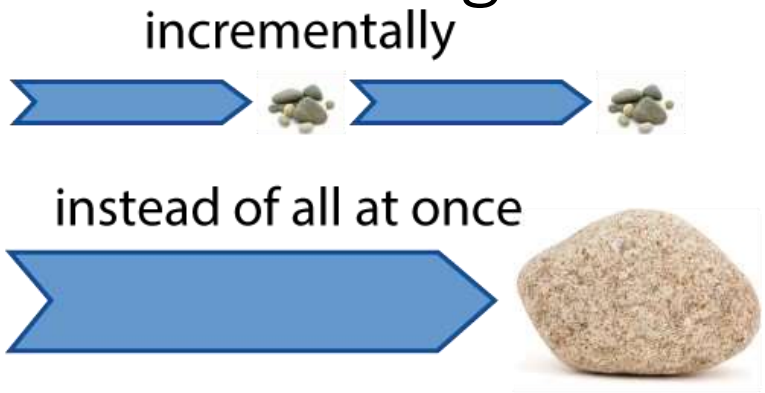
4. Alistair Cockburn 5.Ward Cunningham 6.Martin Fowler

7.James Grenning 8.Jim Highsmith 9.Andrew Hunt
10.Ron Jeffries 11.Jon Kern 12.Brian Marick

13.Robert C. Martin 14.Steve Mellor 15.Ken Schwaber

16.Jeff Sutherland 17.Dave Thomas

What is Agile?



- Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end.
- It works by breaking projects down into little bits of user functionality called user stories, prioritizing them, and then continuously delivering them in short two week cycles called iterations.

How does it work?

- **You make a list:**

  • Sitting down with your customer you make a list of features they would like to see in their software. We call these things user stories and they become the To Do list for your project.

- **You size things up:**

  • You size(estimate) your stories relatively to each other, coming up with a guess as to how long you think each user story will take.

- **You set some priorities:**

  • Like most lists, there always seems to be more to do than time allows. So you ask your customer to prioritize their list so you get the most important stuff done first, and save the least important for last.

- System Features
- 3.1 User Registration & Authentication
- **Description:** Allows users to register, login, and reset passwords.
- **Functional Requirements:**
  - FR1: The system shall provide a registration form with validation.
  - FR2: The system shall authenticate users based on email and password.

Cont...

You start executing:

- Then you start delivering some value. You start at the top. Work your way to the bottom. Building, iterating, and getting feedback from your customer as you go.

  You update the plan as you go :

  Then as you and your customer starting delivering one of two things is going to happen. You'll discover:

- You're going fast enough. All is good. Or,
- You have too much to do and not enough time.

  At this point you have two choices. You can either a) do less and cut scope (recommended). Or you can b) push out the date and ask for more money.

Why Use Agile Methods

- *Improve Customer Involvement*
- *Increase Quality*
- *Simplify*
- *Releases Drive*

  Down Risk

Advantages of Agile model:

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

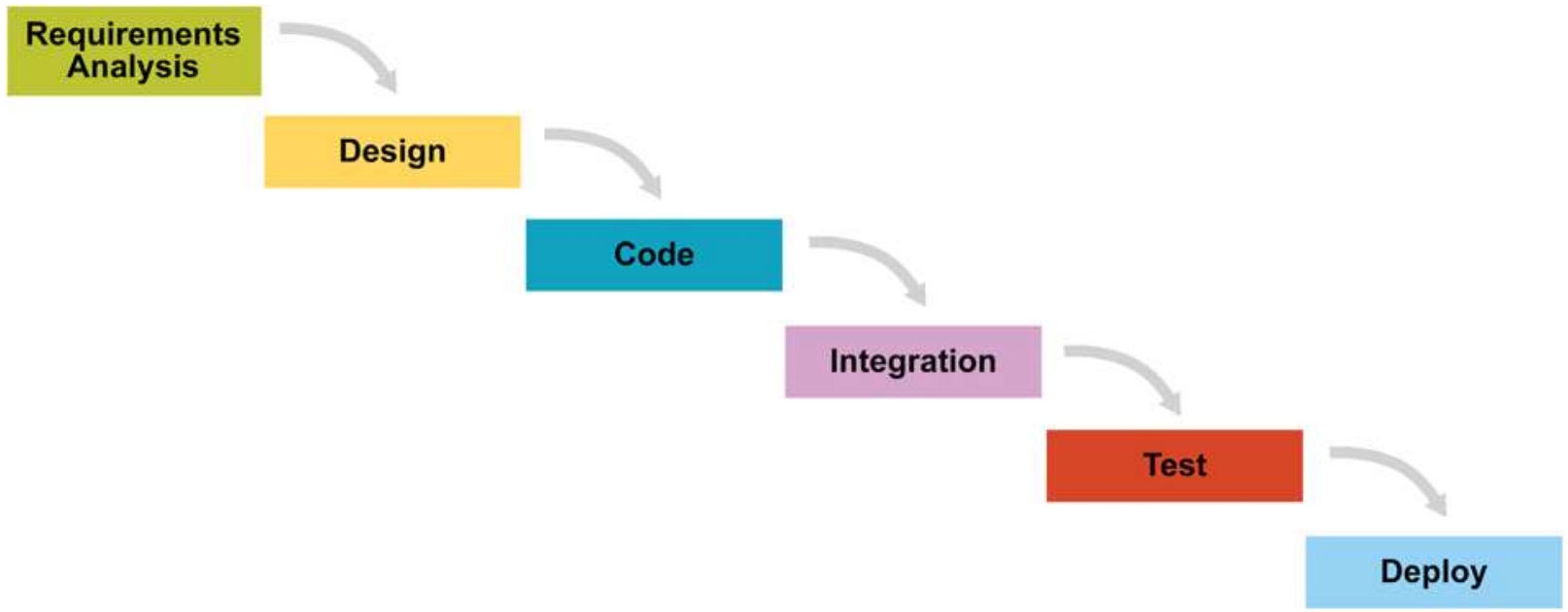
Disadvantages of Agile model:

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.

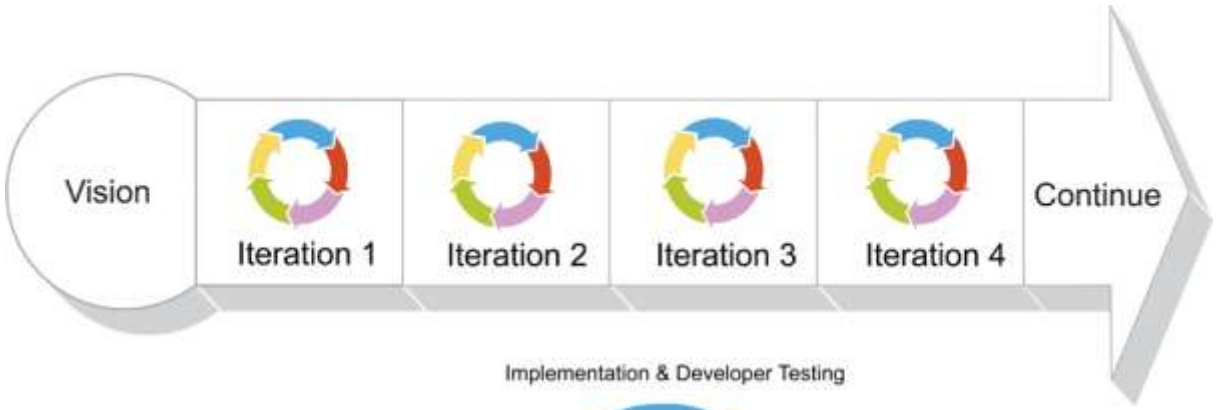
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.

- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

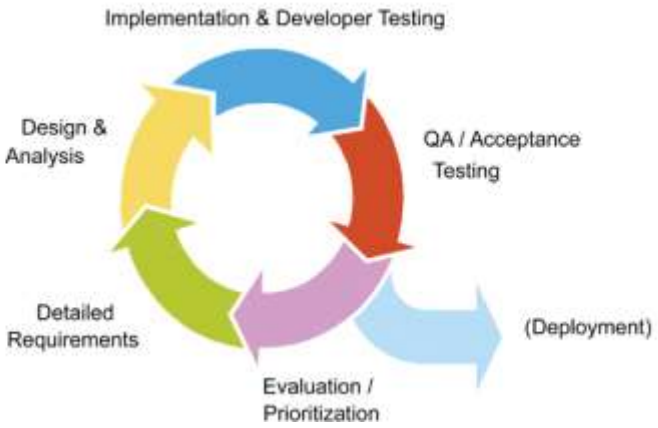
WATERFALL vs AGILE



WATERFALL vs AGILE



Iteration Detail

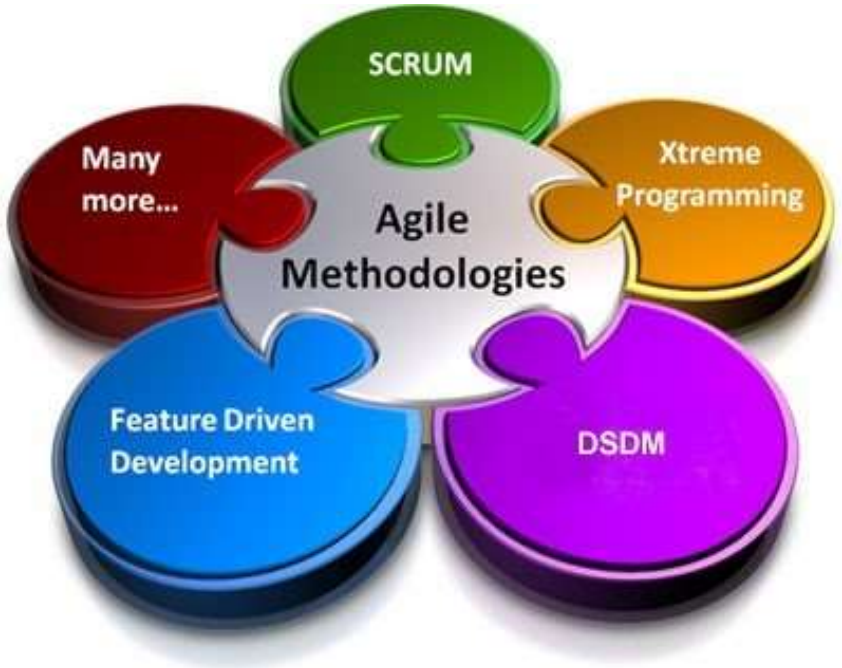


AGILE MANIFESTO

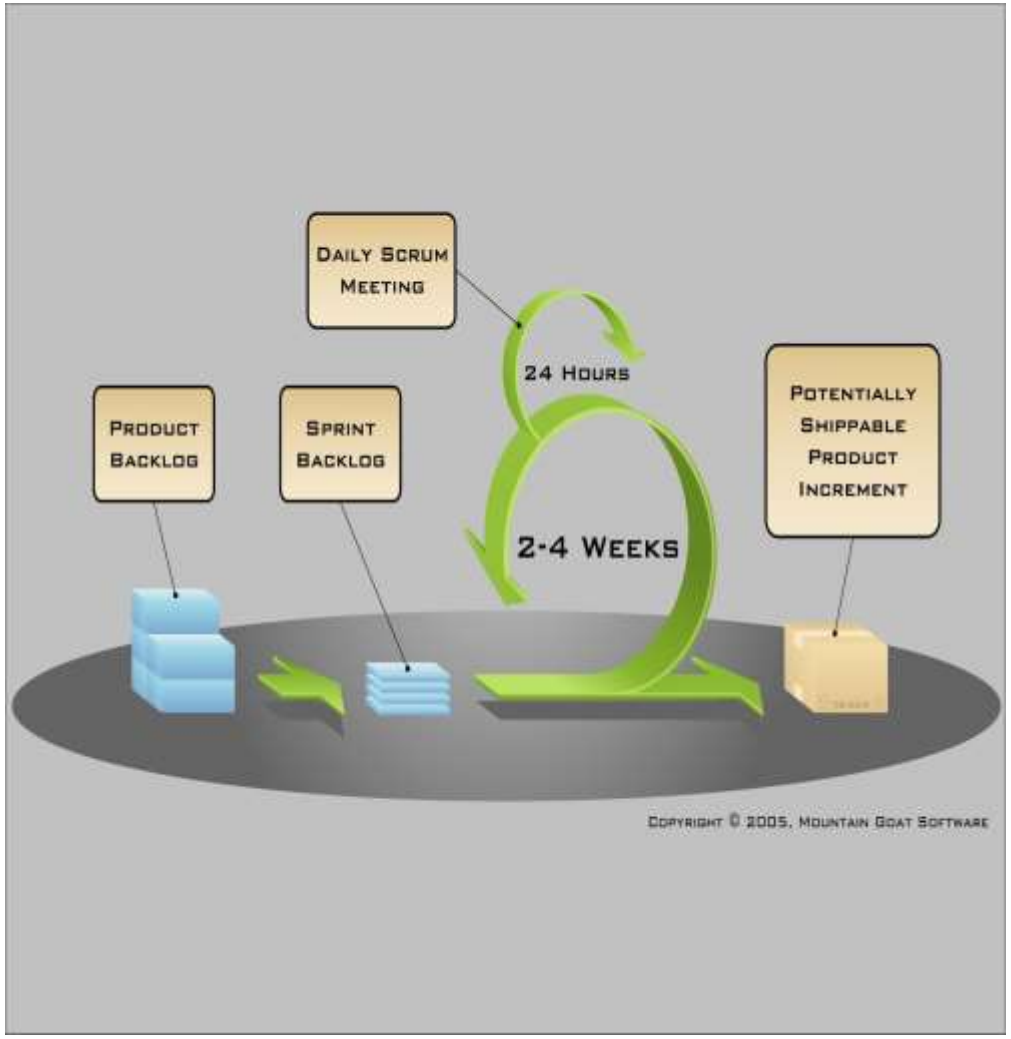


Agile Methodologies

- *Scrum*
- *Extreme Programming (XP)*
- *Dynamic Systems Development Method (DSDM)*
- *Feature-Driven Development (FDD)*
- *Lean and Kanban Software Development*
- *Crystal*



SCRUM



- Ken Schwaber, Mike Beedle, Jeff Sutherland and others have contributed significantly to the evolution of Scrum over the last decade.
- Scrum is a lightweight agile project management framework with broad applicability for managing and controlling iterative and incremental projects of all types.

SCRUM ELEMENTS-ROLES



Product Owner



- Single person responsible for maximizing the return on investment (ROI) of the development effort
- Responsible for product vision
- Constantly re-prioritizes the Product Backlog, adjusting any longterm expectations such as release plans
- Final arbiter of requirements questions
- Accepts or rejects each product
- increment
- Decides whether to ship
- Decides whether to continue
- development Considers stakeholder interests

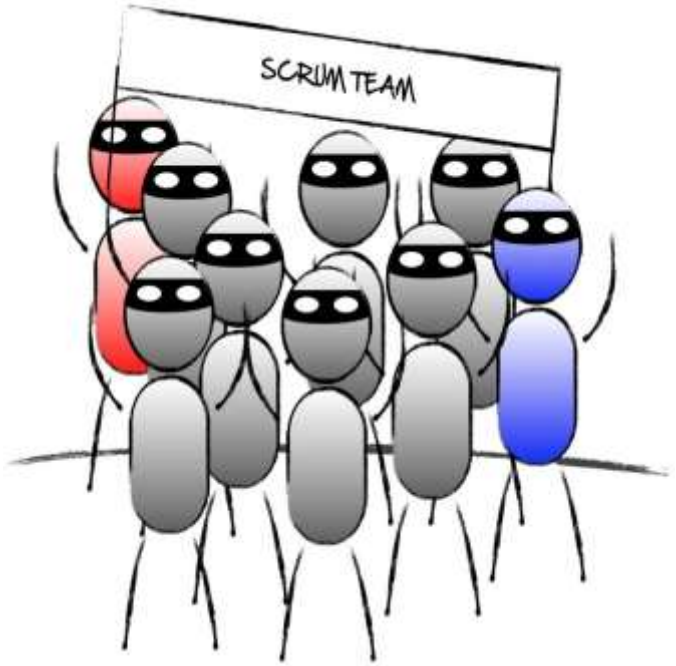
  May contribute as a team

ScrumMaster



- Scrum master is a core of scrum team
- Doing scrum planning for a particular sprint
- Manage dependencies and impediments
- Is a part of scrum team not a manager
- Responsible for scrum ceremonies (like daily stand-up ,demo, retrospective)

Scrum Development Team

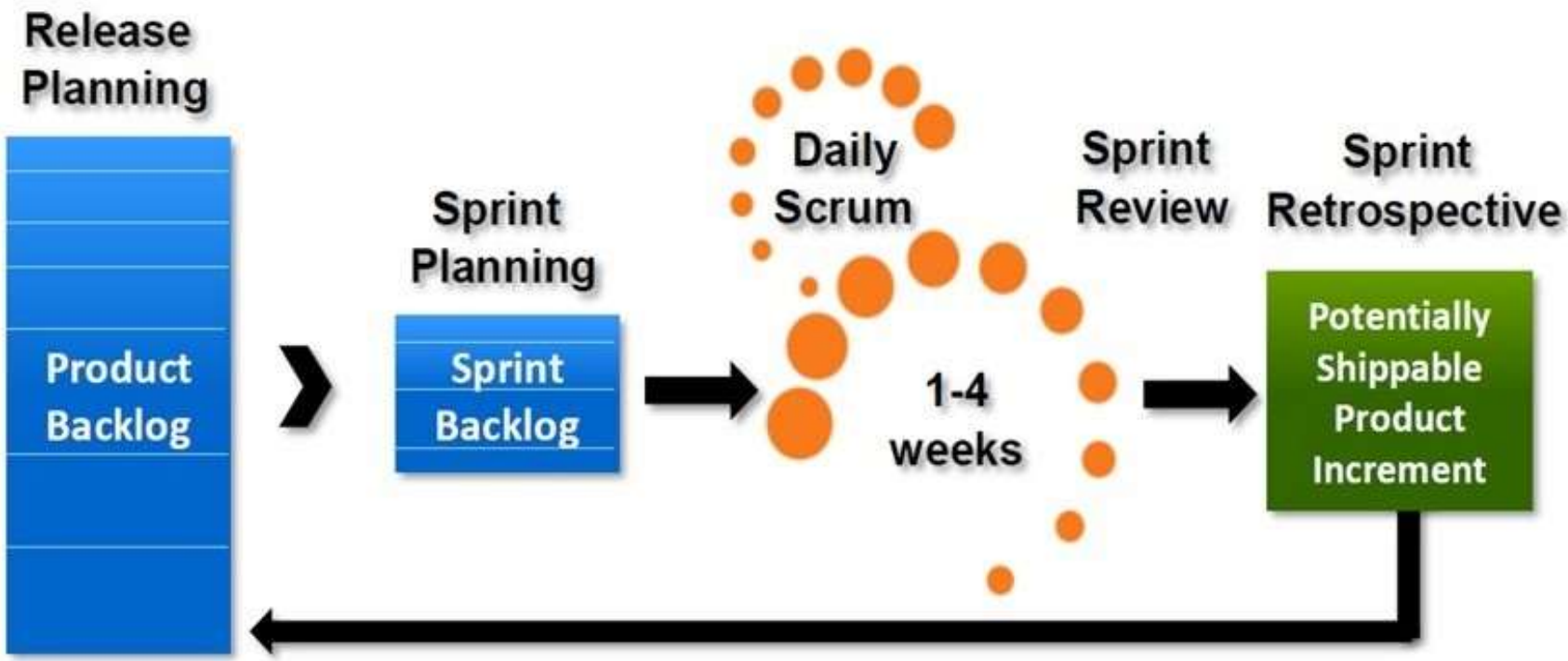


- Scrum Team works towards acheiving goal of the sprint. They work on scrum baklog item and take part in scrum ceremonies.
- Scrum team take part in daily standup and discuss 3 things:
  1. What was done yesterday?
  2. What's the plan for today?
  3. Any impediments?

THE PROCESS

- Sprint Planning Meeting
- Sprint
- Daily Scrum
- Sprint Review Meeting

PROCESS


Release Planning
Product Backlog
Sprint Planning
Sprint Backlog
Daily Scrum
1-4 weeks
Sprint Review
Sprint Retrospective
Potentially Shippable Product Increment

SPRINT

- A month-long iteration, during which is incremented a product functionality
- NO outside influence can interference with the Scrum team during the Sprint
- Each Sprint begins with the Daily Scrum Meeting

Sprint Planning Meeting

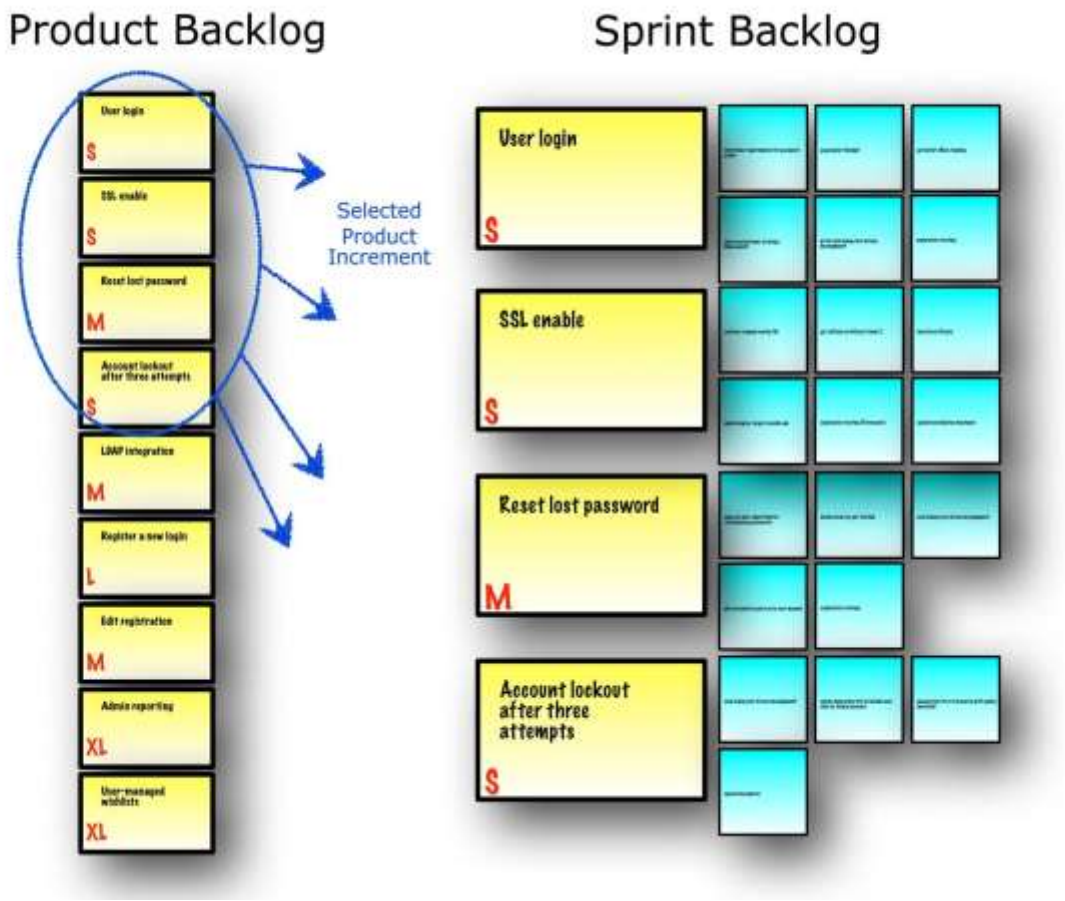
- A collaborative meeting in the beginning of each Sprint between the Product Owner, the Scrum Master and the Team

- Takes 8 hours and consists of 2 parts (“before lunch and after lunch”)

- The Product Owner and team negotiate which Product Backlog Items they will attempt to convert to working product during the Sprint. The Product Owner is responsible for declaring which items are the most important to the business.

- The team is responsible for selecting the amount of work they feel they can implement without accruing technical debt. The team “pulls” work from the Product Backlog to the Sprint Backlog.

SPRINT PLANNING



Parts of Sprint Planning Meeting

- 1st Part:
- Creating Product Backlog
- Determining the Sprint Goal.
- Participants: Product Owner, Scrum Master, Scrum Team
- 2nd Part:
- Participants: Scrum Master, Scrum Team
- Creating Sprint Backlog

Difference between Product Backlog & Sprint Backlog

- Managed by product owner and contains a high-level view of all the work that your team must complete to create the product. Your product owner ranks the user stories in the product backlog and provides sufficient detail during the sprint planning meeting so that your team can estimate and implement each user story.
- In contrast, your team creates the sprint backlog, which contains a detailed list of all the tasks that your team must complete to finish the user stories for the sprint. In the product backlog, your team estimates user stories with the relative unit of story points. In the sprint backlog, your team estimates tasks in hours.
- Your product owner updates the product backlog every week, but your team updates the sprint backlog at least daily.
- Your product owner maintains the same product backlog throughout the project, but your team creates a new sprint backlog for each sprint.

Pre-Project/Kickoff Meeting

A special form of Sprint Planning Meeting

Meeting before the begin of the Project

DAILY SCRUM


What did I do yesterday that helped the Development Team meet the Sprint Goal?
What will I do today to help the Development Team meet the Sprint Goal?
Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

SPRINT REVIEW MEETING

- Is held at the end of each Sprint
- Business functionality which was created during the Sprint is demonstrated to the Product Owner
- Informal, should not distract Team members of doing their work

SCRUM ARTIFACTS

- Product Backlog
- Sprint Backlog
- Burn down Charts

Product Backlog

- Requirements for a system, expressed as a prioritized list of Backlog Items
- Is managed and owned by a Product Owner
- Spreadsheet (typically)
- Usually is created during the Sprint Planning Meeting Can be changed and re-prioritized before each PM

Sprint Backlog

- No more then 300 tasks in the list
- If a task requires more than 16 hours, it should be broken down
- Team can add or subtract items from the list. Product Owner is not allowed to do it.

Burn down Charts

- Are used to represent “work done”.
- Are wonderful Information

Radiators 3 Types:

- Sprint Burn down Chart (progress of the Sprint)
- Release Burn down Chart (progress of release)
- Product Burn down chart (progress of the Product)

BENNETT UNIVERSITY
THE TIMES GROUP

CSET-324: Software Project Management

Schedule Development

Schedule Development

- Uses results of the other time management processes to determine the start and end date of the project
- Ultimate goal is to create a realistic project schedule that provides a basis for monitoring project progress for the time dimension of the project
- Important tools and techniques include Gantt charts, critical path analysis, and critical chain scheduling, and PERT analysis

BENNETT UNIVERSITY
THE TIMES GROUP

Schedule Development



Inputs

1. Project Management Plan
2. Project Documents
3. Agreements
4. Enterprise Environmental Factors
5. Organizational Process Assets

Tools & Techniques

1. Project Management Plan
2. Critical Path Method
3. Resource Optimization
4. Data Analysis
5. Leads and Lags
6. Schedule Compression
7. Project Management Information System
8. Agile Release Planning

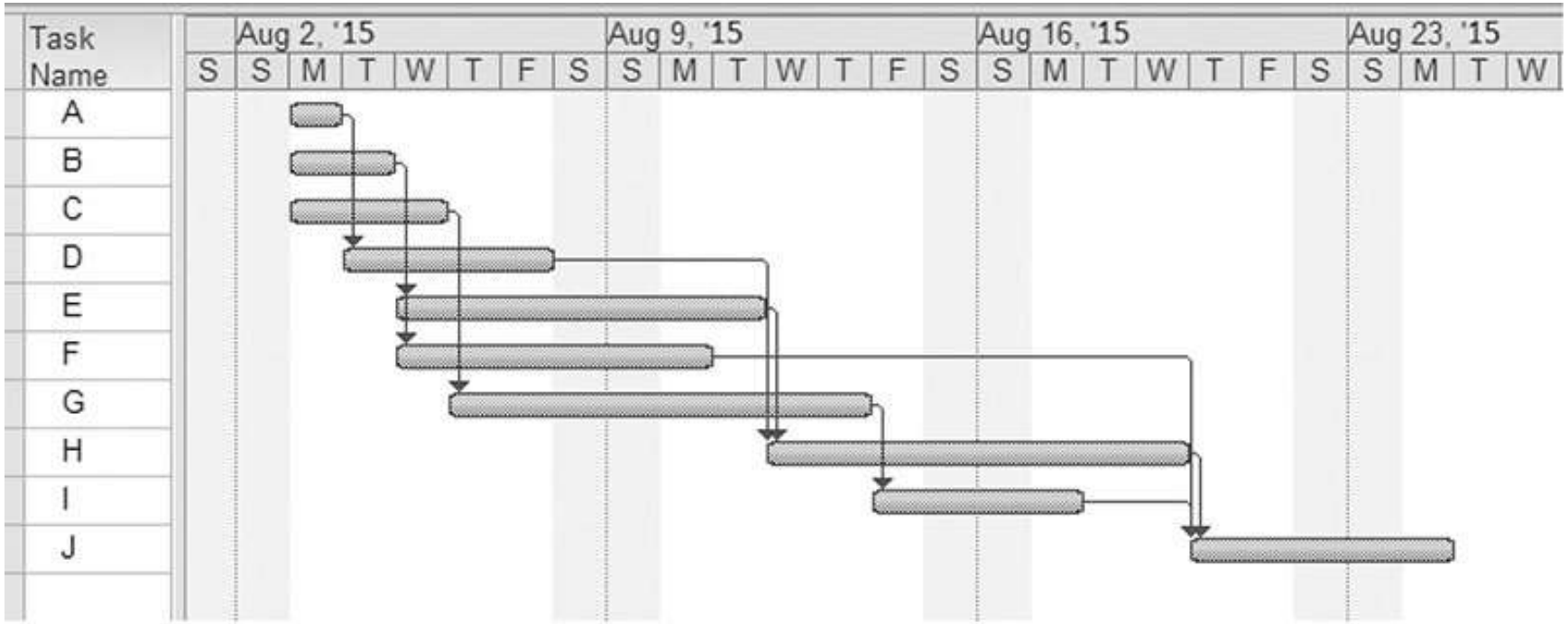
Outputs

1. Schedule Baseline
2. Project Schedule
3. Schedule Data
4. Project Calendars
5. Change Requests
6. Project Management Plan Updates
7. Project Documents Updates

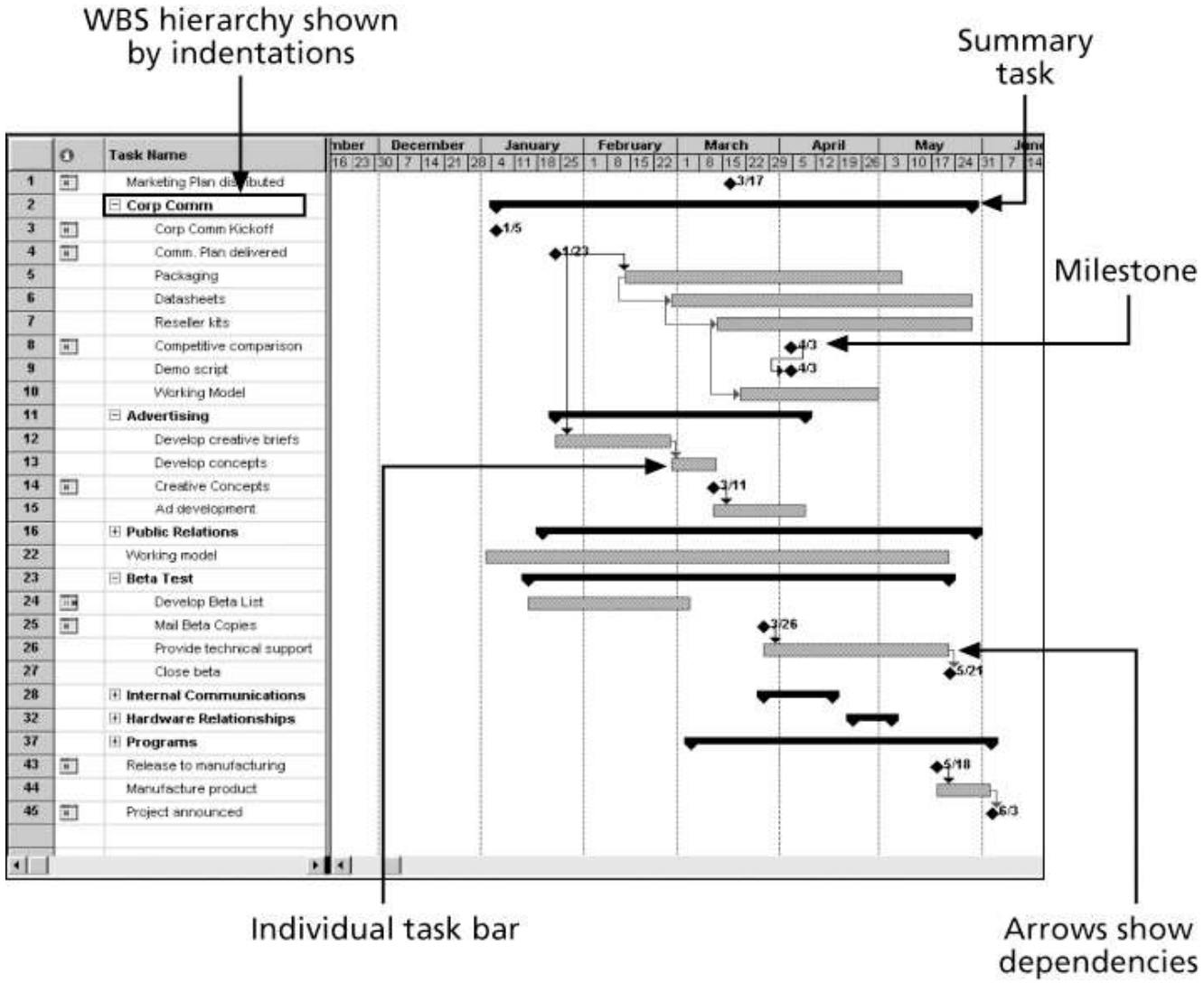
Gantt Charts

- **Gantt charts** provide a standard format for displaying project schedule information by listing project activities and their corresponding start and finish dates in a calendar format
- Symbols include:
  - A black diamond: a milestones
  - Thick black bars: summary tasks
  - Lighter horizontal bars: durations of tasks
  - Arrows: dependencies between tasks

Gantt Chart for Project X



Gantt Chart for Software Launch Project



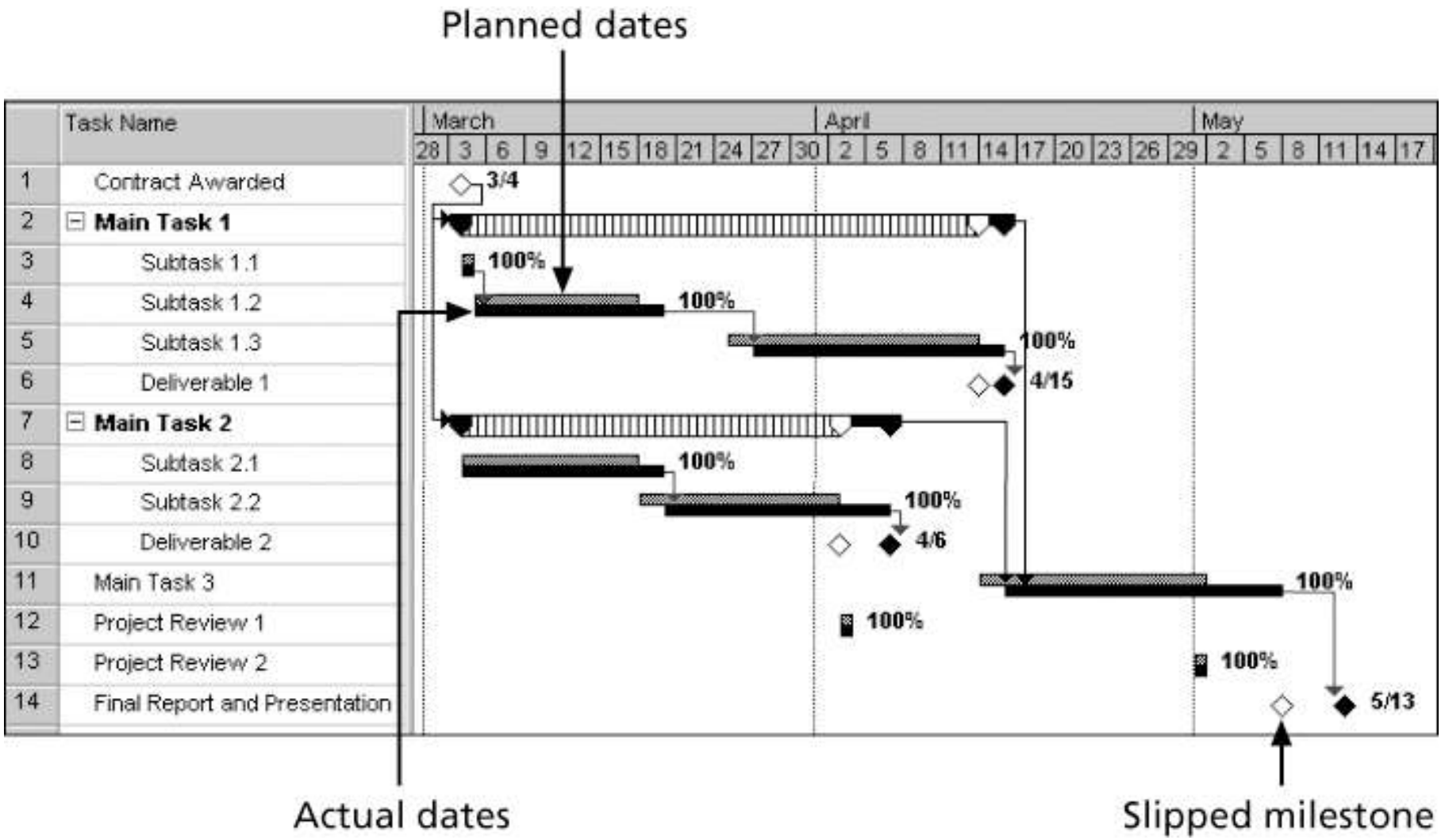
Adding Milestones to Gantt Charts

- Many people like to focus on meeting milestones, especially for large projects
- Milestones emphasize important events or accomplishments on projects
- Normally create milestone by entering tasks with a zero duration, or you can mark any task as a milestone

SMART Criteria

- Milestones should be
  - **S**pecific
  - **M**easurable
  - **A**ssignable
  - **R**ealistic
  - **T**ime-framed

Sample Tracking Gantt Chart



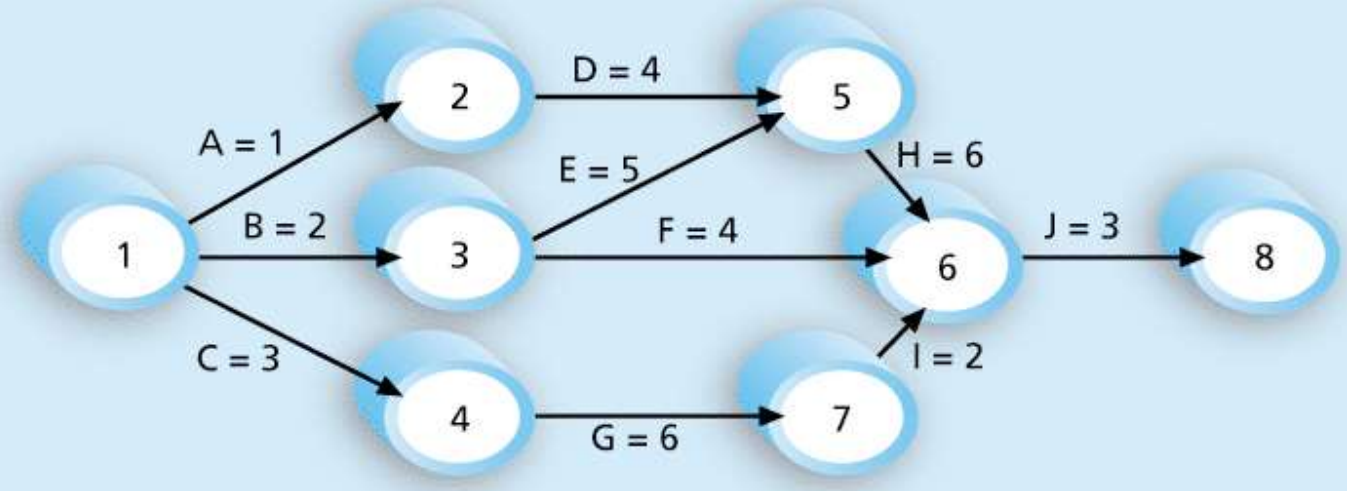
Critical Path Method (CPM)

- **CPM** is a network diagramming technique used to predict total project duration
- A **critical path** for a project is the series of activities that determines the *earliest time* by which the project can be completed
- The critical path is the *longest path* through the network diagram and has the least amount of slack or float
- **Slack** or **float** is the amount of time an activity may be delayed without delaying a succeeding activity or the project finish date

Calculating the Critical Path

- First develop a good network diagram
- Add the duration estimates for all activities on each path through the network diagram
- The longest path is the critical path
- If one or more of the activities on the critical path takes longer than planned, the whole project schedule will slip *unless* the project manager takes corrective action

Determining the Critical Path for Project X



Note: Assume all durations are in days.

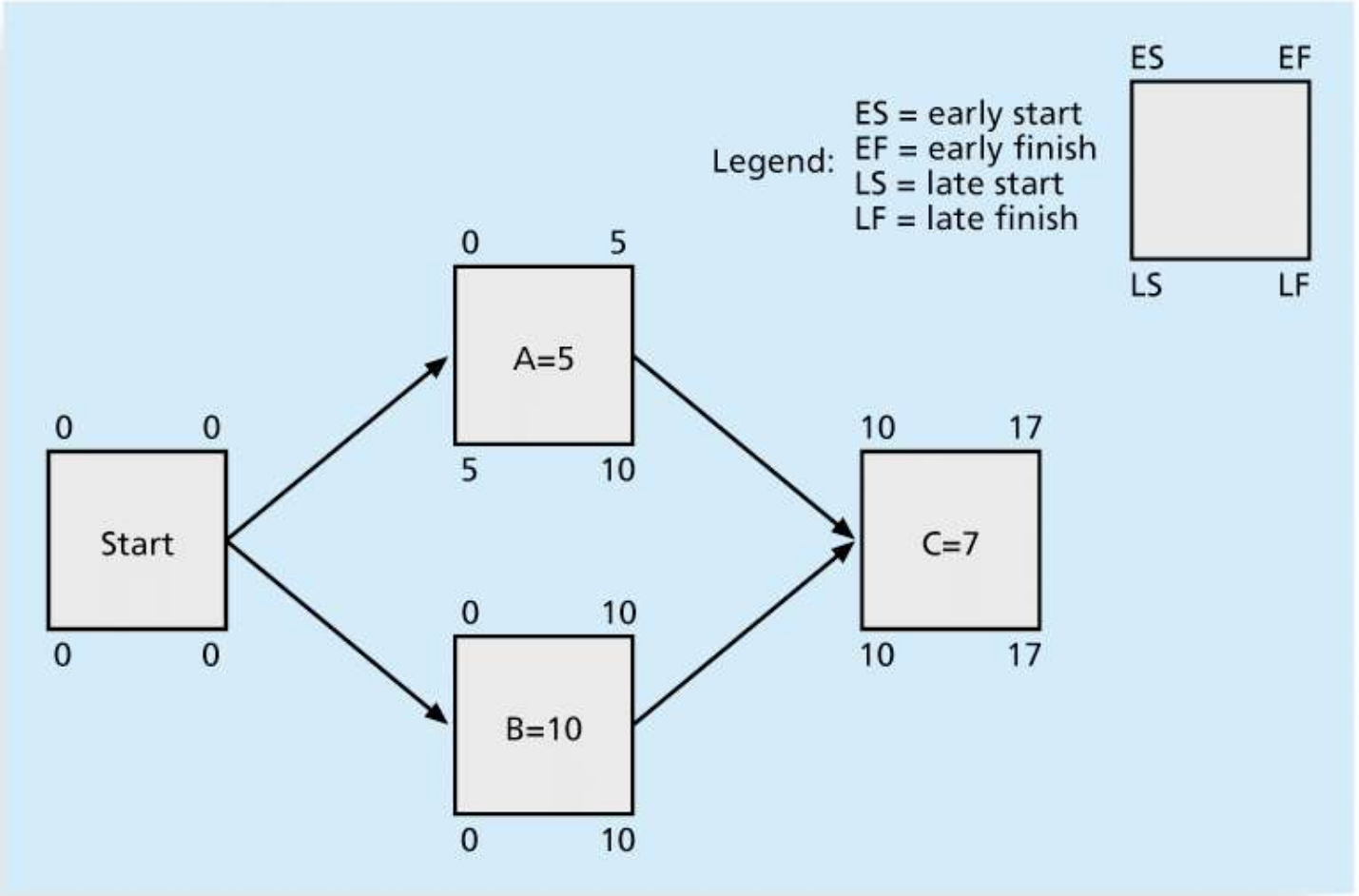
Path 1:	A-D-H-J	Length = 1+4+6+3 = 14 days
Path 2:	B-E-H-J	Length = 2+5+6+3 = 16 days
Path 3:	B-F-J	Length = 2+4+3 = 9 days
Path 4:	C-G-I-J	Length = 3+6+2+3 = 14 days

Since the critical path is the longest path through the network diagram, Path 2, B-E-H-J, is the critical path for Project X.

Using Critical Path Analysis to Make Schedule Trade-offs

- **Free slack** or **free float** is the amount of time an activity can be delayed without delaying the early start of any immediately following activities
- **Total slack** or **total float** is the amount of time an activity may be delayed from its early start without delaying the planned project finish date
- A **forward pass** through the network diagram determines the early start and finish dates
- A **backward pass** determines the late start and finish dates

Calculating Early and Late Start and Finish Dates



Free and Total Float or Slack for Project X

Task Name	Start	Finish	Late Start	Late Finish	Free Slack	Total Slack
A	8/3/15	8/3/15	8/5/15	8/5/15	0d	2d
B	8/3/15	8/4/15	8/3/15	8/4/15	0d	0d
C	8/3/15	8/5/15	8/5/15	8/7/15	0d	2d
D	8/4/15	8/7/15	8/6/15	8/11/15	2d	2d
E	8/5/15	8/11/15	8/5/15	8/11/15	0d	0d
F	8/5/15	8/10/15	8/14/15	8/17/15	7d	7d
G	8/6/15	8/13/15	8/10/15	8/17/15	0d	2d
H	8/12/15	8/19/15	8/12/15	8/19/15	0d	0d
I	8/14/15	8/17/15	8/18/15	8/19/15	2d	2d
J	8/20/15	8/24/15	8/20/15	8/24/15	0d	0d

© Cengage Learning 2016

Using the Critical Path to Shorten a Project Schedule

- Three main techniques for shortening schedules
  - Shortening durations of critical activities/tasks by adding more resources or changing their scope
  - **Crashing** activities by obtaining the greatest amount of schedule compression for the least incremental cost
  - **Fast tracking** activities by doing them in parallel or overlapping them

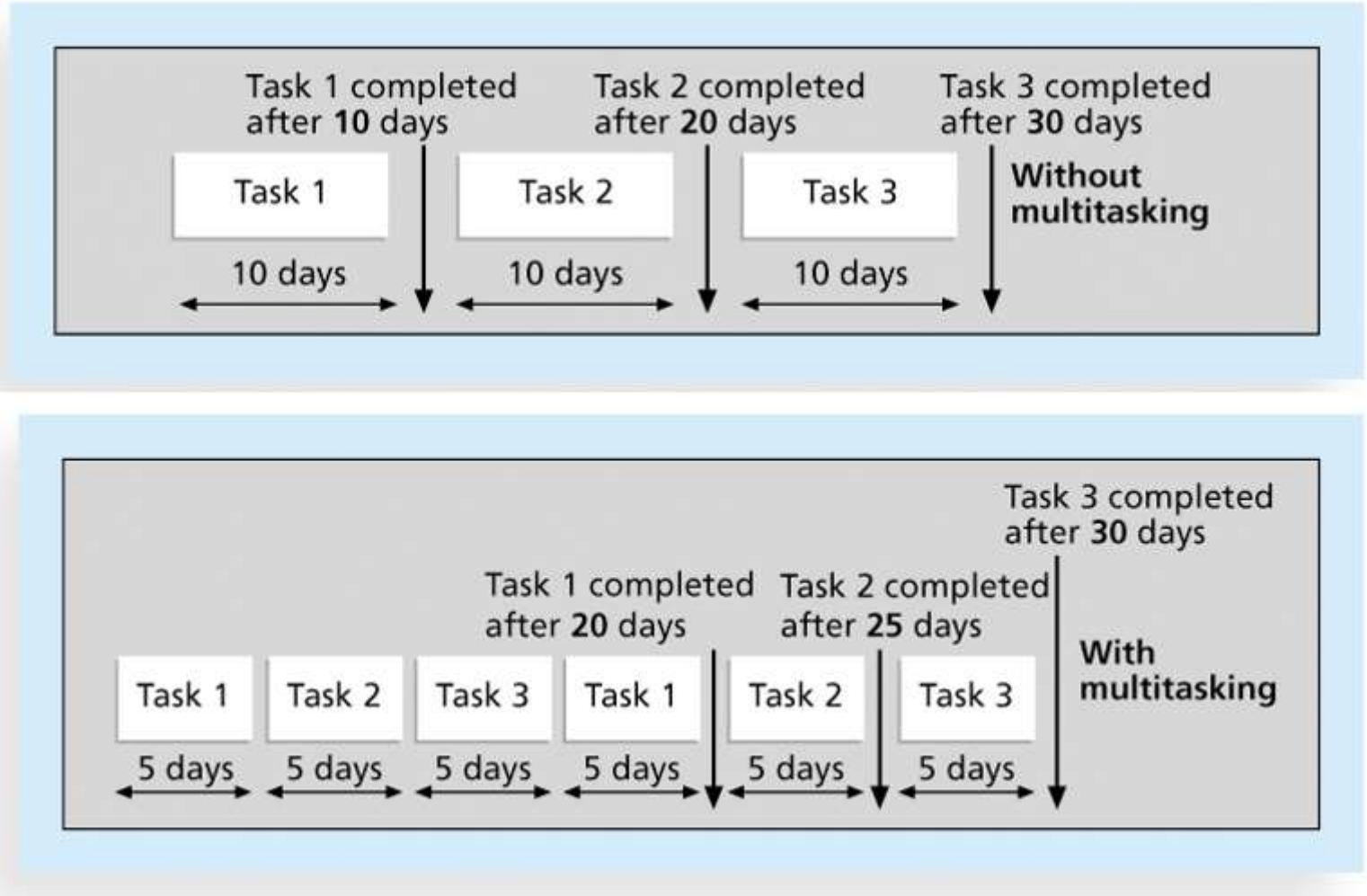
Importance of Updating Critical Path Data

- It is important to update project schedule information to meet time goals for a project
- The critical path may change as you enter actual start and finish dates
- If you know the project completion date will slip, negotiate with the project sponsor

Critical Chain Scheduling

- **Critical chain scheduling**
  - a method of scheduling that considers limited resources when creating a project schedule and includes buffers to protect the project completion date
- Uses the **Theory of Constraints (TOC)**
  - a management philosophy developed by Eliyahu M. Goldratt and introduced in his book *The Goal*.
- Attempts to minimize **multitasking**
  - when a resource works on more than one task at a time

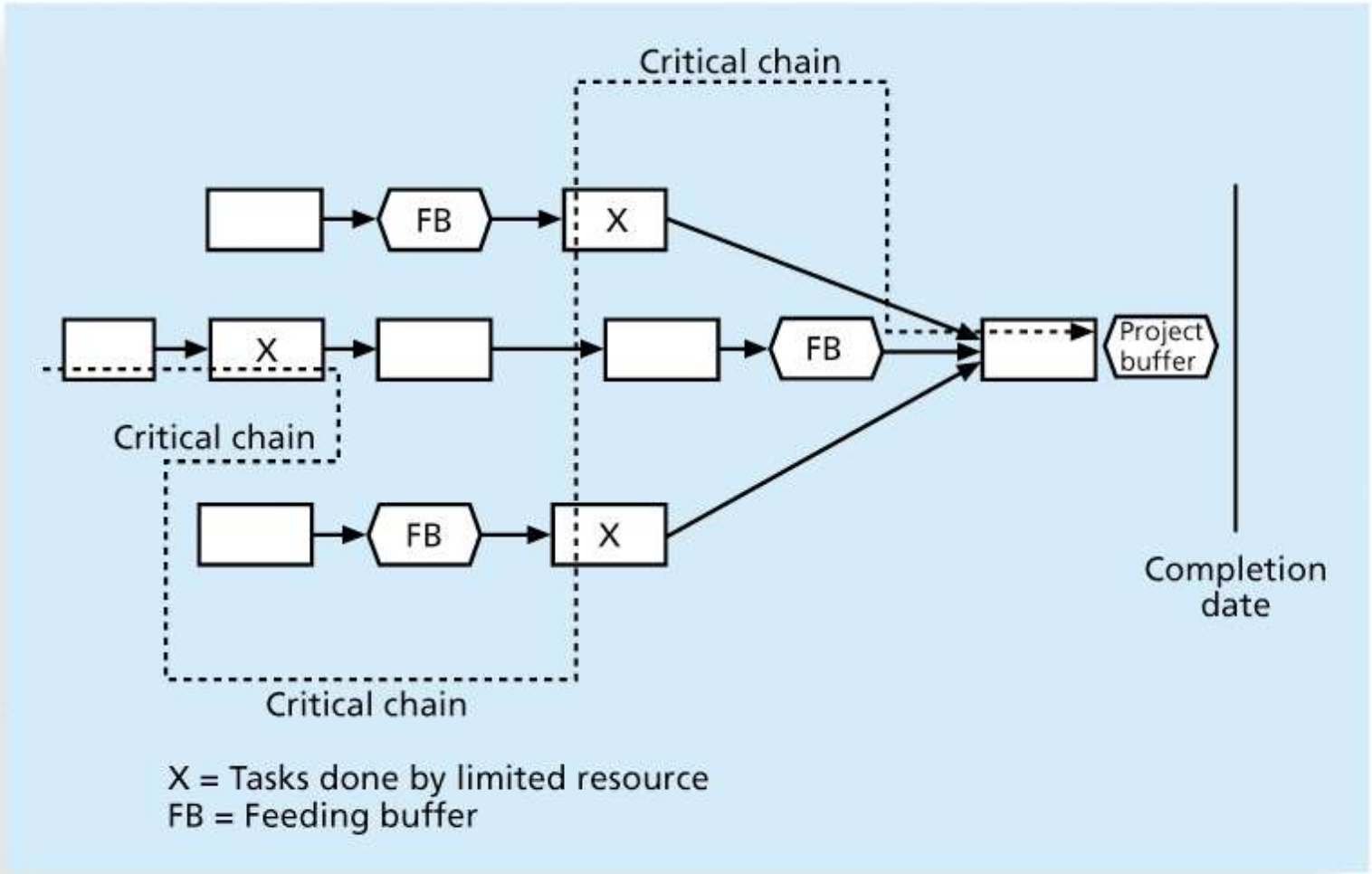
Multitasking Example



Buffers and Critical Chain

- A **buffer** is additional time to complete a task
- **Murphy's Law** states that if something can go wrong, it will
- **Parkinson's Law** states that work expands to fill the time allowed
- In traditional estimates, people often add a buffer to each task and use it if it's needed or not
- Critical chain scheduling removes buffers from individual tasks and instead creates
  - a **project buffer** or additional time added before the project's due date
  - **feeding buffers** or additional time added before tasks on the critical path

Example of Critical Chain Scheduling



Program Evaluation and Review Technique (PERT)

- **PERT** is a network analysis technique used to estimate project duration when there is a high degree of uncertainty about the individual activity duration estimates
- PERT uses **probabilistic time estimates**
  - duration estimates based on using optimistic, most likely, and pessimistic estimates of activity durations, or a three-point estimate

PERT Formula and Example

- PERT weighted average =

$$\frac{\text{optimistic time} + 4\text{X most likely time} + \text{pessimistic time}}{6}$$

- Example:

PERT weighted average =

$$\frac{8 \text{ workdays} + 4 \text{ X } 10 \text{ workdays} + 24 \text{ workdays}}{6} = \textbf{12 days}$$

where optimistic time= 8 days

most likely time = **10 days**, and

pessimistic time = 24 days

Therefore, you'd use **12 days** on the network diagram instead of 10 when using PERT for the above example



Software Project Management

Contents

- Putnam/SLIM Model,
- Estimating by Analogy,
- Cost Budgeting and Control

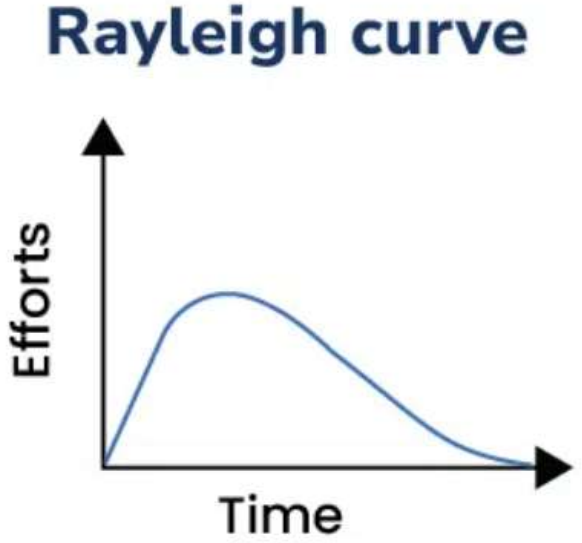
Putnam / SLIM Model

Putnam / SLIM Model

- **Putnam Model**, also known as the **SLIM Model** (Software Life Cycle Management), is a **software cost estimation model** developed by **Lawrence H. Putnam** in the late 1970s.
- It is based on the **Rayleigh-Norden curve** and emphasizes **time-effort tradeoffs** in software development.
- Predicts effort, schedule, and cost using historical project data and an empirically derived equation.
- It models the **staffing pattern over time** and is **non-linear**, accounting for **diminishing returns** when adding more effort to shorten development time.

Rayleigh Curve

- A bell-shaped curve that models how manpower is used over time in a software project.
- Reflects **slow ramp-up**, **peak staffing**, and **tapering off** as the project nears completion.



Putnam Equation

The main formula is:

$$\frac{S}{C_k} = (E)^{\frac{1}{3}} \cdot (T_d)^{\frac{4}{3}}$$

Where:

- S: Product size in **delivered source instructions** (DSI)
- E: Effort in **person-years**
- T_d: Development time in **years**
- C_k: Technology constant or **productivity parameter**, depends on tools, team, and environment

  Rearranged to solve for effort:

$$E = \left(\frac{S}{C_k \cdot T_d^{4/3}} \right)^3$$

  Or to solve for time:

$$T_d = \left(\frac{S}{C_k \cdot E^{1/3}} \right)^{3/4}$$

Assumptions of the SLIM Model

- Project follows a **waterfall model**
- Manpower builds up and reduces smoothly (Rayleigh curve).
- Applicable mainly for **large-scale** software projects
- Productivity (C□) remains constant during the project

Example

- Given:
  - Size $S = 200,000$ lines of code (200 KDSI)
  - Productivity constant $C_k = 2,000$
  - Desired development time $T_d = 2$ years

Step 1: Calculate Effort

$$E = \left(\frac{200,000}{2000 \times 2^{4/3}} \right)^3$$

First, calculate denominator:

$$2^{4/3} \approx 2.5198$$

$$2000 \times 2.5198 \approx 5039.6$$

Now compute:

$$\frac{200,000}{5039.6} \approx 39.7$$

$$E = (39.7)^3 \approx 62567 \text{ person-months}$$

Converted to person-years:

$$\approx 5213.9 \text{ person-years}$$

Estimating by Analogy

Estimating by Analogy

- **Comparative estimation technique** where the effort, cost, or duration of a **new software project** is estimated based on **actual data from similar past projects**.
- **Steps Involved in Estimating by Analogy**
  - **Identify past similar projects** (from your organization or known repositories)
  - **Extract key metrics** (e.g., size, complexity, effort, duration, cost)
  - **Compare and adjust** for differences (team size, technology stack, risk level)
  - **Apply the adjusted data** to estimate the new project
  - **Validate** with subject matter experts or historical data

Example

- Estimate by analogy, the **effort** and **cost** for a **new e-commerce mobile app** with ~100 user stories.
- **Similar Past Project:** Consider a previously developed **retail inventory app** with- 90 user stories, Took 6 months, Required 1200 person-hours, Cost ₹12 lakhs
- **Step 1: Find Similarity**
  - The retail app and e-commerce app are similar in:
  - Architecture (mobile + cloud backend)
  - Technology (React Native + Firebase)
  - Team experience
- **Step 2: Scale the Estimate**
  - New project = 100 stories → 11% more than 90 stories
  - **Estimated effort** = 1200 × (100 / 90) = 1333 person-hours
    Estimated cost = ₹12 lakhs × (100 / 90) = ₹13.33 lakhs
    Estimated duration = 6 months × (100 / 90) ≈ 6.6 months

Applications of Estimating by Analogy

Estimation Aspect	Example
Effort	Compare similar modules from past
Cost	Use past app development cost as base
Schedule	Timeline from similar project types
Size	Function points or user stories of previous systems

Advantages

- Simple & Intuitive
- Reuses Knowledge
- Adaptable
- Useful for Agile

Cost Budgeting and Control

Cost Budgeting and Control

- **Cost budgeting** is the process of **allocating the overall cost estimate** to individual work items or components of a software project to establish a **cost baseline**.
- **Objective:**
  - Define how much **funding** is needed and **when**
  - Establish a **cost baseline** for performance measurement
  - Enable **tracking of actual costs vs. planned costs**

Cost Control

- **Cost control** is the ongoing process of:
  - **Monitoring project costs**
  - **Measuring cost performance**
  - **Ensuring project stays within the approved budget**
  - **Taking corrective actions** when there are cost overruns

Phases in Cost Budgeting and Control

A. Cost Budgeting Steps:

- **Determine project WBS** (Work Breakdown Structure)
- **Estimate costs** for each work package
- **Aggregate costs** to form the total project budget
- **Define cost baseline** with time-phased expenditure

B. Cost Control Steps:

- **Track actual spending**
- **Compare with baseline**
- **Calculate cost variances (CV) and cost performance index (CPI)**
- **Forecast future performance**
- **Implement corrective actions**

Key Terms & Formulas

Metric	Formula	Meaning
Planned Value (PV)	Budgeted Cost of Work Scheduled	How much was planned to be spent
Earned Value (EV)	Budgeted Cost of Work Performed	Value of work actually done
Actual Cost (AC)	Actual Cost of Work Performed	What you actually spent
Cost Variance (CV)	EV - AC	Negative CV = over budget
Cost Performance Index (CPI)	EV / AC	CPI < 1 = cost overrun
Estimate at Completion (EAC)	BAC / CPI	Total estimated project cost

Example

Project: Development of an ERP Module

- Total Budget (BAC) = ₹10,00,000
- Planned Work for 2 Months (PV) = ₹4,00,000
- Completed Work Value (EV) = ₹3,50,000
- Actual Cost so far (AC) = ₹4,50,000

- **Step 1: Calculate CV and CPI**

$$\text{CV} = EV - AC = ₹3,50,000 - ₹4,50,000 = -₹1,00,000 \quad \text{(Over Budget)}$$

- $$\text{CPI} = EV/AC = 3,50,000/4,50,000 = 0.78 \quad \text{(Less than 1} \rightarrow \text{Inefficient)}$$
- **Estimate At completion(EAC)**

$$\text{EAC} = BAC/CPI = 10,00,000/0.78 \approx ₹12,82,000$$

Revision

- CoComo model
- **What it does:** Estimates software development effort and cost based on lines of code.
- **Think of it like:** Estimating how long it takes to write an essay based on word count
- **Types:**
- **Basic COCOMO:** Simple calculation using just lines of code
- **Intermediate COCOMO:** Adds complexity factors (team experience, tools, etc.)
- **Detailed COCOMO:** Most accurate, considers each phase separately

- Effort = a × (KLOC)^b
- Time = c × (Effort)^d

- **2. Function Point Analysis**
- **What it does:** Estimates project size based on functionality from user's perspective
- **Think of it like:** Estimating restaurant work by counting dishes served, not ingredients used
- **3. Putnam Model (SLIM)**
- **What it does:** Uses historical data patterns to estimate software projects
- **Think of it like:** Predicting marathon time based on training patterns of similar runners

- **PERT (Program Evaluation and Review Technique)**
- **What it does:** Plans and schedules complex projects with uncertain task durations
- **Think of it like:** Planning a wedding where some things might take longer than expected
- **Key Features:**
- **Handles uncertainty** with three time estimates per task
- **Finds critical path** (longest sequence of dependent tasks)
- **Probabilistic** approach

- **5. CPM (Critical Path Method)**
- **What it does:** Schedules projects by identifying the sequence of tasks that determines minimum project duration
- **Think of it like:** Finding the bottleneck route in city traffic that determines your total travel time

A library management system has the following components:

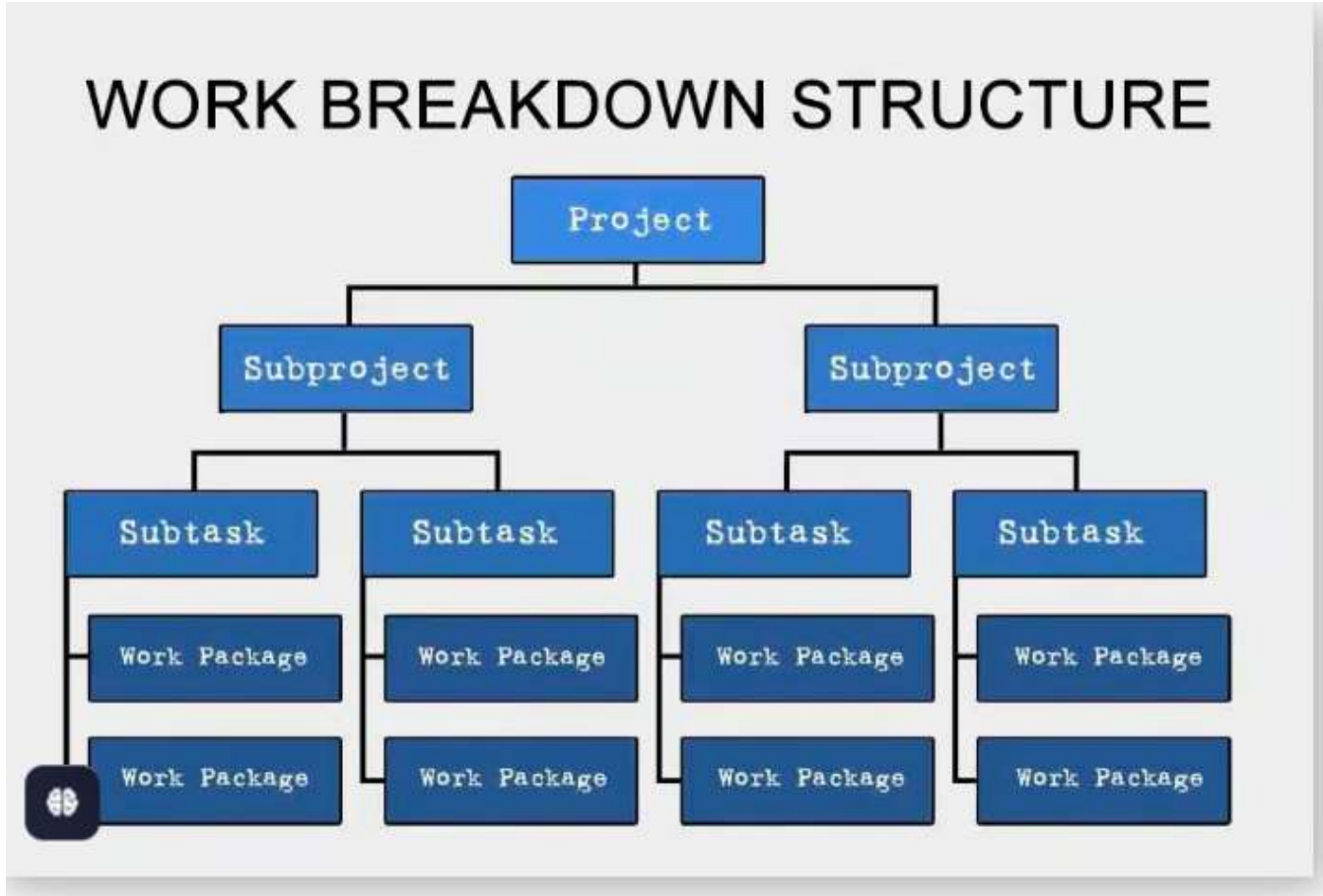
- External Inputs (EI): Add Book (Simple), Issue Book (Average), Return Book (Average)
- External Outputs (EO): Generate Reports (Complex), Print Receipt (Simple)
- Internal Logical Files (ILF): Book Master (Average), Member Master (Simple)
- External Interface Files (EIF): Publisher Database (Simple)
- External Inquiries (EQ): Search Books (Average)
- Calculate the Unadjusted Function Points (UFP) using standard complexity weights.
- **Given Weights:** EI: Simple=3, Average=4, Complex=6; EO: Simple=4, Average=5, Complex=7; ILF: Simple=7, Average=10, Complex=15; EIF: Simple=5, Average=7, Complex=10; EQ: Simple=3, Average=4, Complex=6.

Software Project Management

- Topic-1
- Earned Value Management (EVM) and Portfolio Management
- Topic-2
- Quality Planning, Quality Assurance, Quality Control
- Quality Control Tools

Topic-1

- Earned Value Management (EVM)
- Portfolio Management


WORK BREAKDOWN STRUCTURE
Project
Subproject
Subproject
Subtask
Subtask
Subtask
Subtask
Work Package
Work Package
Work Package
Work Package
Work Package
Work Package
Work Package
Work Package

Earned Value Management

- ***Earned Value Management*** (EVM) is a systematic approach to the integration and measurement of cost, schedule, and technical (scope) accomplishments on a project or task.

- ***Earned Value*** (EV) directly links all three primary project success Metrics (cost, schedule, and performance).

Earned Value Management

By using Earned Value and implementing an Earned Value Management System (EVMS), the following questions can be answered objectively:

- Where have we been?
- Where are we now?
- Where are we going?

Traditional management Vs Earned Value Management



Traditional management provides you with…



How much money and time a particular job is likely to require prior to starting and once stated, how much money was spent at any given time.



While **Earned Value Management provides you with…**



How much money and time a particular job is likely to require prior to starting and once stated, how much money was spent at any given time.



Earned Value Terms

- Planned value (PV)
- Earned value (EV)
- Actual cost of work performed (AC)
- Schedule variance (SV) and Schedule performance index (SPI)
- Cost variance (CV) and Cost performance index (CPI)
- Budgeted cost at completion (BAC)
- Estimate at completion (EAC)

- **Planned Value (PV)** - The authorized budget assigned to scheduled work. This is what you *planned* to spend by a given point in time.

- **Earned Value (EV)** - The measure of work actually performed, expressed in terms of the budget authorized for that work. This is the value you've actually *earned* through completed work.

- **Actual Cost (AC)** - The realized cost incurred for the work performed. This is what you've actually *spent*.

Steps in Earned Value Management

1. *Clearly define each activity* including its resource needs and budget.
2. *Create usage schedules* for activities and resources.
3. *Develop a time-phased budget* (PV).
4. *Total the actual costs* of doing each task (AC).
5. *Calculate* both the budget variance (CV) and schedule variance (SV).

Earned Value Example

<u>Schedule Variances</u>

Planned Value (PV) = 103

Earned Value (EV) = 44

Schedule Performance Index = .43 = 44/103 = EV/PV

Estimated Time to Completion = (1/.43)x7=16.3 months

<u>Cost Variances</u>

Cumulative Actual Cost of Work Performed (AC) = 78

Cost Performance Index = .56 = 44/78 = EV/AC

Estimated Cost to Completion = $210,714 = (1/.56)x$118,000

An Earned Value Management System (EVMS) framework

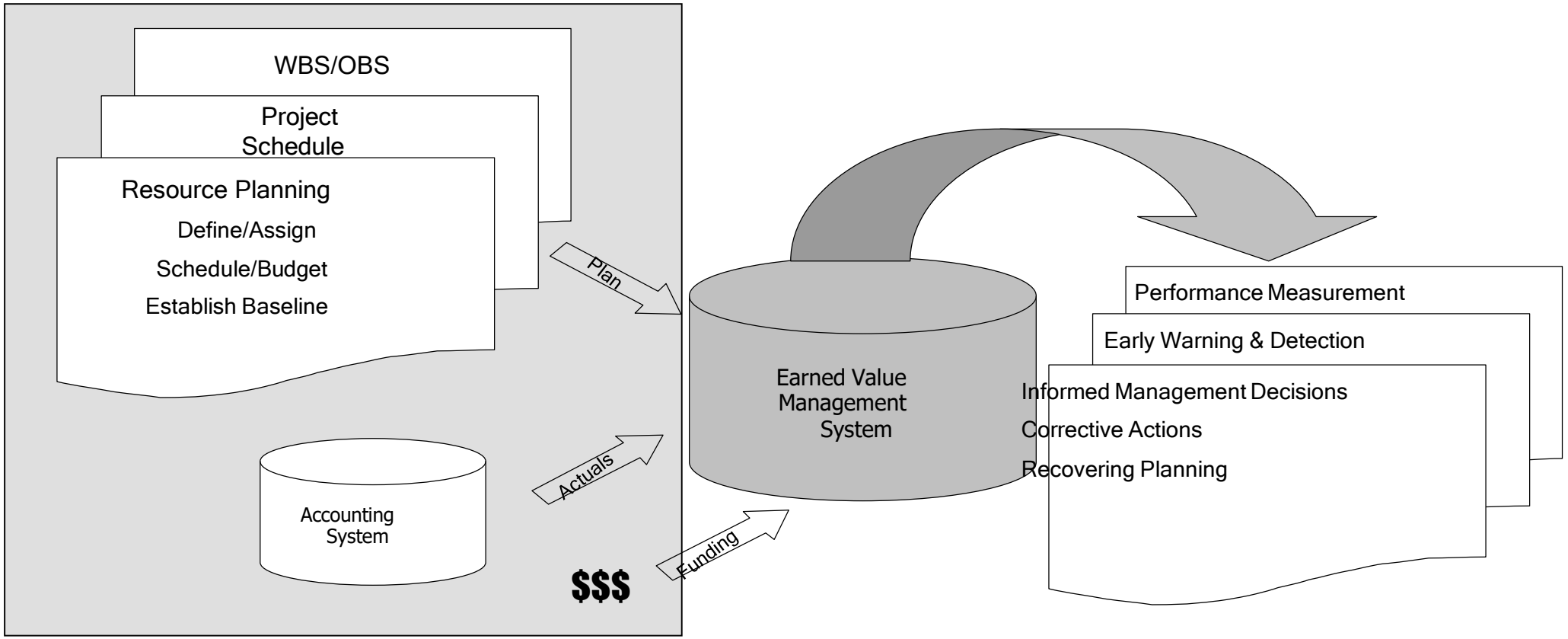
The EVMS framework can be divided into three phases:

1. Inputs - what is needed to implement Earned Value

2. Earned Value Methods – formulas, metrics and performance measurements used

3. Outputs – reporting requirements (structure, time-phases, details)

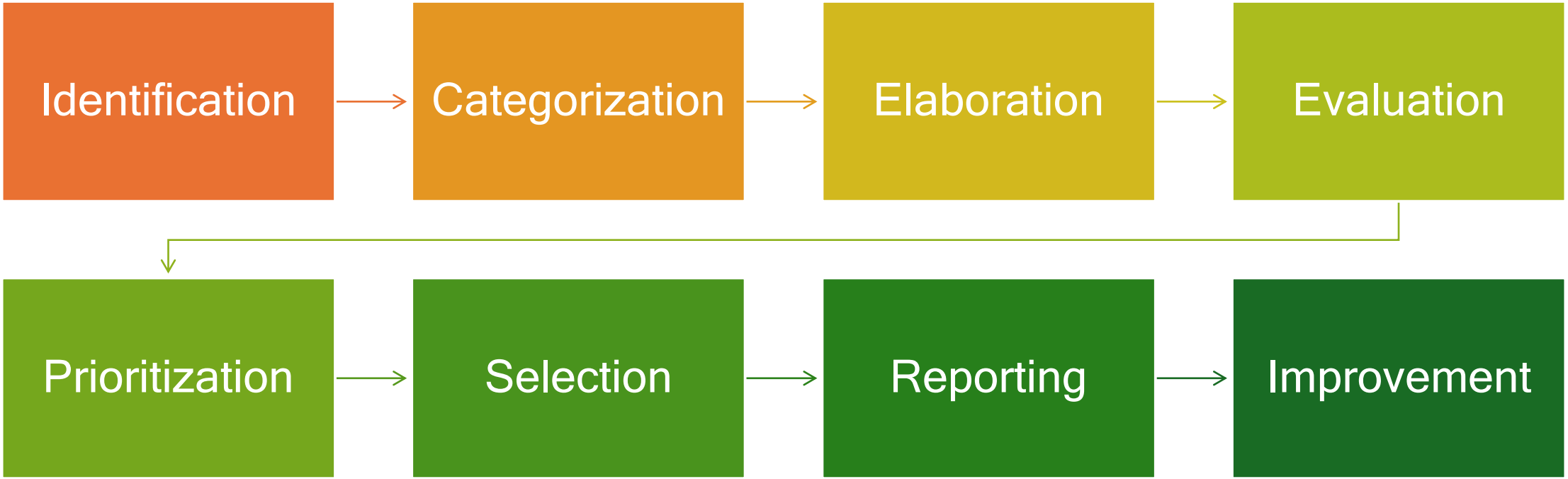
An Earned Value Management System (EVMS) framework



WHAT IS PORTFOLIO MANAGEMENT?

Gathering	Gathering candidate ideas for investment from various sources (internal and external)
Considering	Considering candidate and approved projects based upon an analytic decision-making framework
Choosing	Choosing highly ranked projects that together meet constraints for budget and resources.
Reporting	Reporting on changes, progress, and results in the portfolio
Reconsidering	Reconsidering the portfolio on an ongoing basis
Practicing	Practicing continuous improvement of Portfolio Management processes.

EIGHT KEY PROCESSES ARE INVOLVED IN CHOOSING THE RIGHT PORTFOLIO



Topic-2

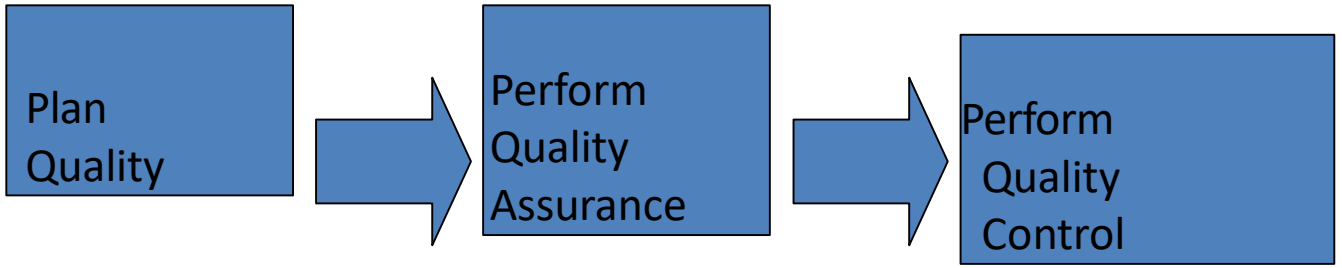
- Quality Planning, Quality Assurance and Quality Control
- Quality Control Tools

Definitions of Quality

- Quality is meeting or exceeding customer's expectations
- Quality is the totality of features and characteristics of a product or service that bears on its ability to satisfy stated or implied needs.
- Quality is sum of the attributes that can describe a project or product.
- Quality refers to meeting specifications and performance standards.
- Managing the quality of the software process and products is important.

How Do We Manage Quality?

- Three processes
  - Plan Quality
  - Perform Quality Assurance
  - Perform Quality Control



Quality management activities

- **Quality assurance**
  - Establish organisational procedures and standards for quality
- **Quality planning**
  - Select applicable procedures and standards for a particular project and modify these as required
- **Quality control**
  - Ensure that procedures and standards are followed by the software development team
- Quality management should be separate from project management to ensure independence

Quality Assurance & Quality Control



Quality assurance aims to assure that quality work and quality deliverables will be built in before work is done



Quality control aims to determine that quality work and quality deliverables did occur after work was done



Quality control is conducted by inspection and testing Of materials, workmanship and product

Quality Assurance Vs Control

- Quality Assurance is process oriented and focuses on defect prevention.
- QA is a set of activities for ensuring quality in the processes by which products are developed.

- Whereas

- Quality Control is product oriented and focuses on defect identification.
- QC is a set of activities for ensuring quality in products. The activities focus on identifying defects in the actual products produced.

Focus- Assurance Vs Control

- QA aims to prevent defects with a focus on the process used to make the product. It is a proactive quality process.
- QC aims to identify (and correct) defects in the finished product. Quality control, therefore, is a reactive process.

Software Quality Assurance

- **Software quality assurance** (**SQA**) consists of a means of monitoring the software engineering processes and methods used to ensure quality.

- The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI.

SQA

- SQA encompasses the entire software development process, which includes processes.
  - Requirements definition,
  - Software design,
  - Coding,
  - Source code control,
  - Code reviews,
  - Change management,
  - Configuration management,
  - Testing,
  - Release management, and
  - Product integration.

Quality planning

- A quality plan sets out the desired product qualities and how these are assessed and define the most significant quality attributes.
- It should define the quality assessment process.
- It should set out which organisational standards should be applied and, if necessary, define new standards.

Quality plan structure

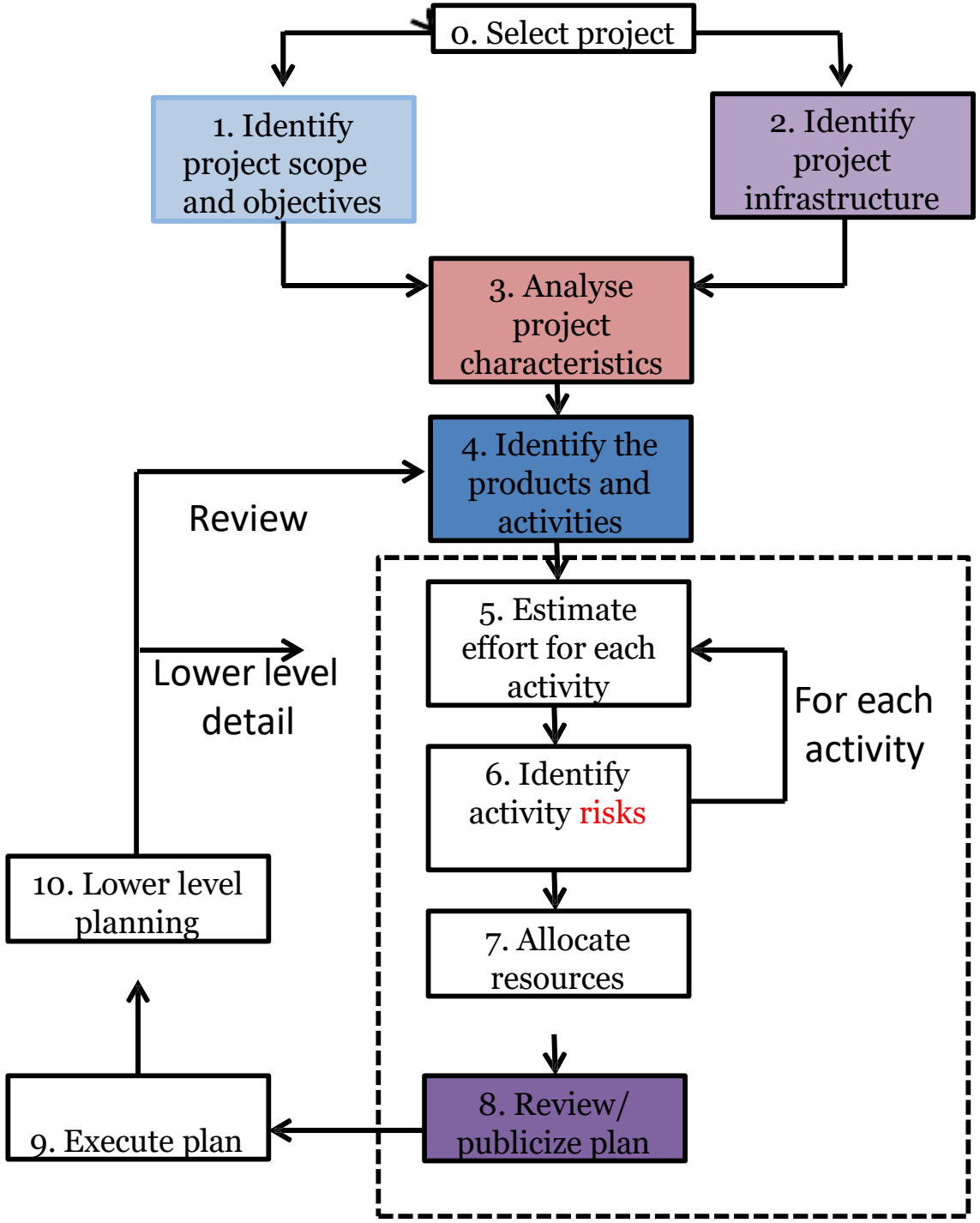
- Product introduction
- Product plans
- Process descriptions
- Quality goals
- Risks and risk management
- Quality plans should be short, succinct documents
  - If they are too long, no-one will read them

Quality plans

- quality standards and procedures should be documented in an organization's *quality manual*
- for each separate project, the quality needs should be assessed
- select the level of quality assurance needed for the project and document in a *quality plan*

Typical contents of a quality plan

- scope of plan
- references to other documents
- quality management, including organization, tasks, and responsibilities
- documentation to be produced
- standards, practices and conventions
- reviews and audits
- scope of plan
- references to other documents
- quality management, including organization, tasks, and responsibilities
- documentation to be produced
- standards, practices and conventions
- reviews and audits

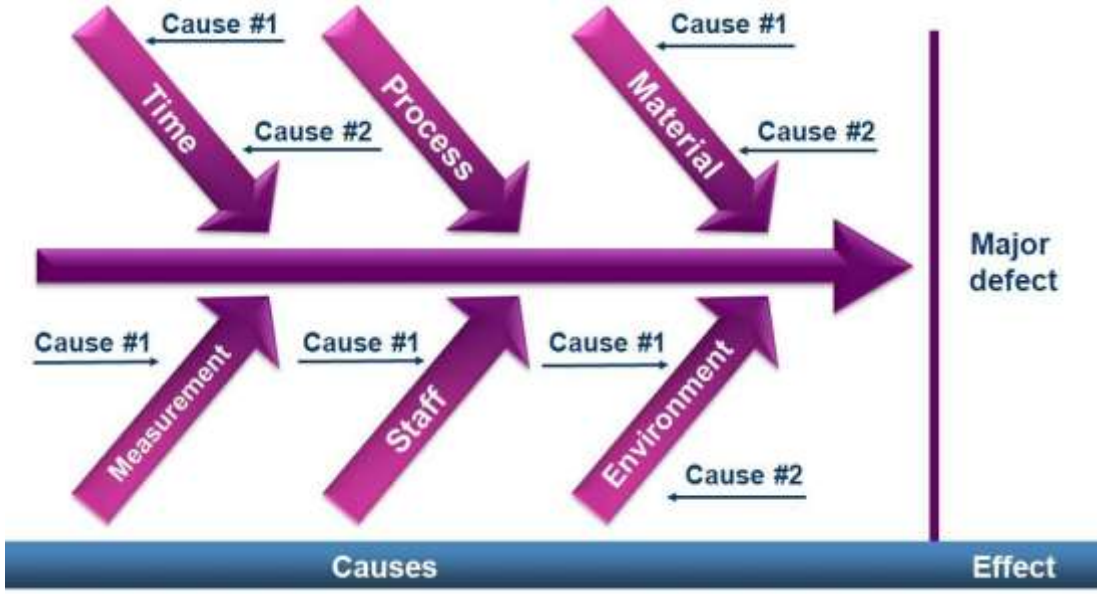

0. Select project
1. Identify project scope and objectives
2. Identify project infrastructure
3. Analyse project characteristics
4. Identify the products and activities
Review
5. Estimate effort for each activity
Lower level detail
For each activity
6. Identify activity risks
7. Allocate resources
10. Lower level planning
8. Review/ publicize plan
9. Execute plan

7 Basic Quality Control Tools (7QC)

- Cause & Effect Diagram(Fishbone/Ishikawa))
- Flowchart
- Histogram
- Pareto Chart
- Check Sheet
- Scatter Diagram
- Control Chart

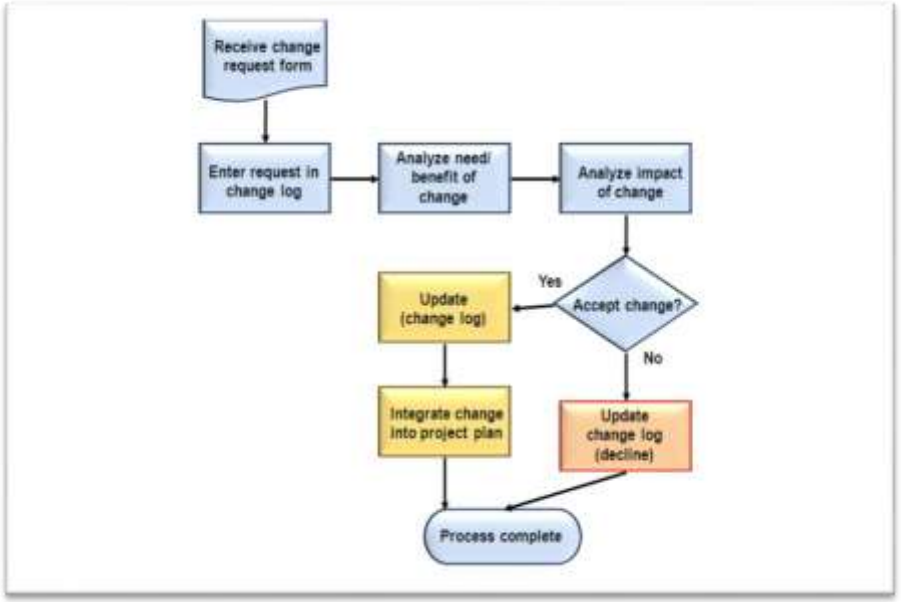
Cause & Effect Diagram

- To identify cause and effect relationship



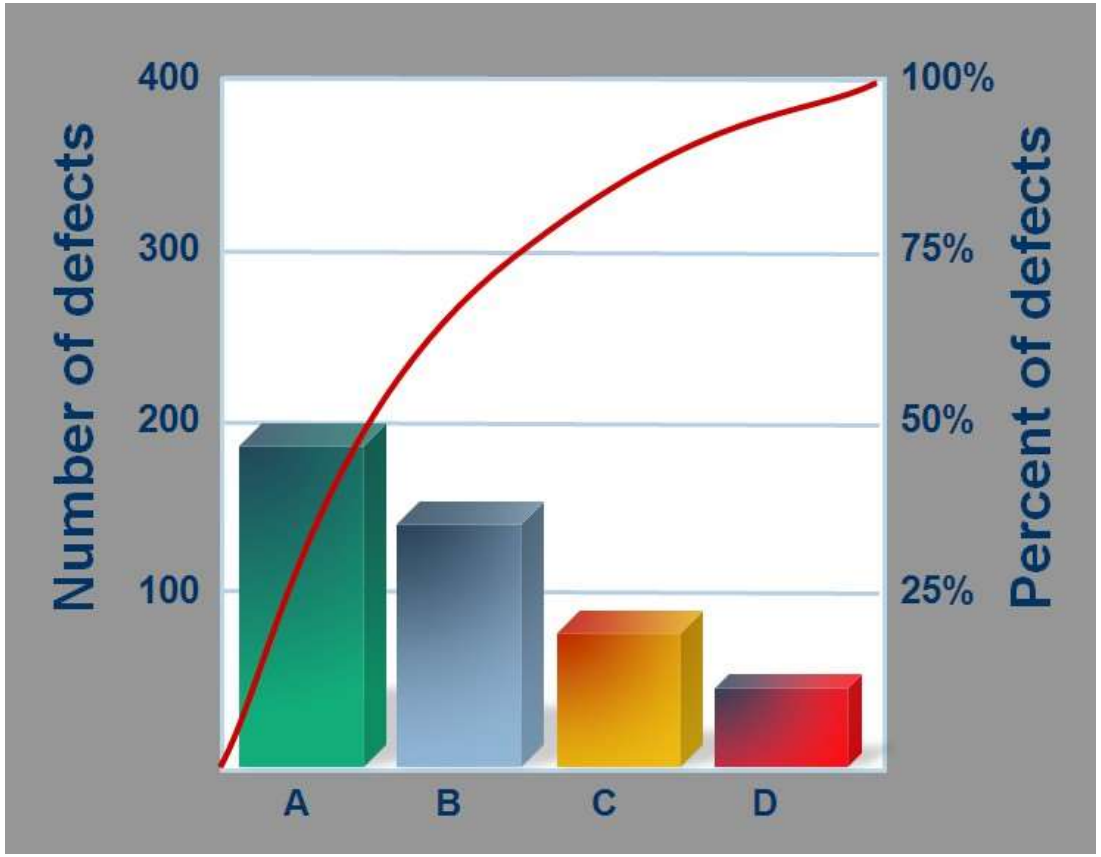
Flow Charts

- Basic processing performed when collecting data.
- Flowcharts show the logical steps in a process and how various elements within a system are related.
- They can be used to determine and analyse potential problems in quality planning and quality control.



Pareto diagram

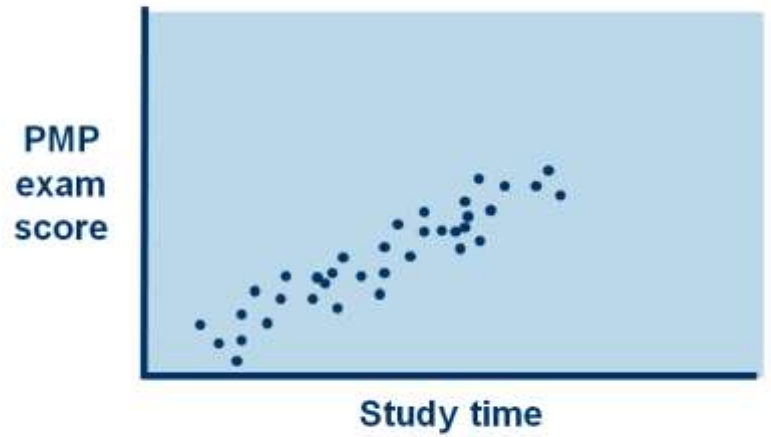
- To identify current status and issues.
- Based on Pareto's principle, frequently referred to as the 80/20 rule or 80/20 principle. Which means that eighty percent of the cost of defects are caused by twenty percent of the problems.



- 20% of customers generate 80% of revenue
- 20% of bugs cause 80% of software crashes
- 20% of employees produce 80% of output
- 20% of products account for 80% of sales

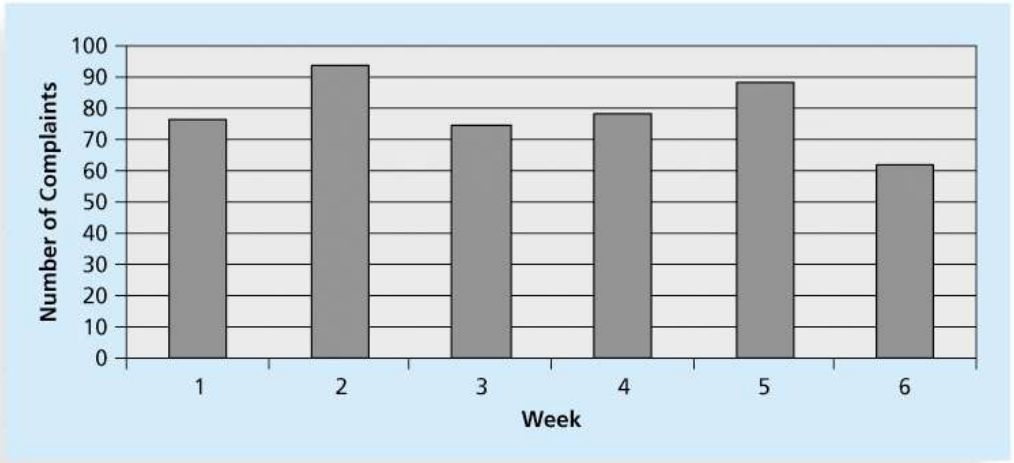
Scatter Diagram

- To identify relationship between two things.



Histogram

- To see the distribution of data.
- A **histogram** is a bar graph of a distribution of variables.



Check Sheets

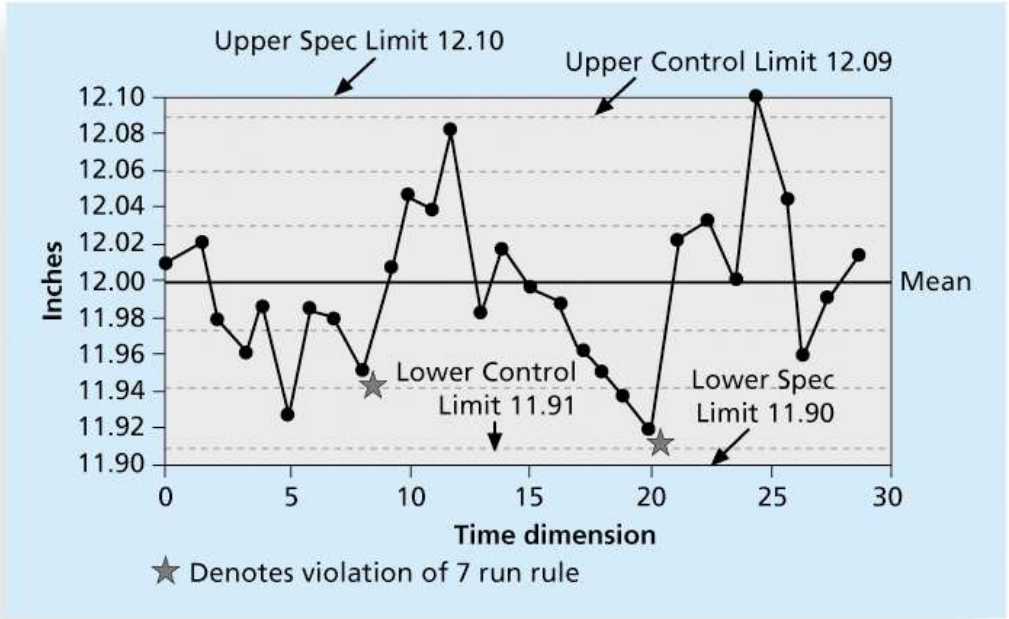
- To record data collections

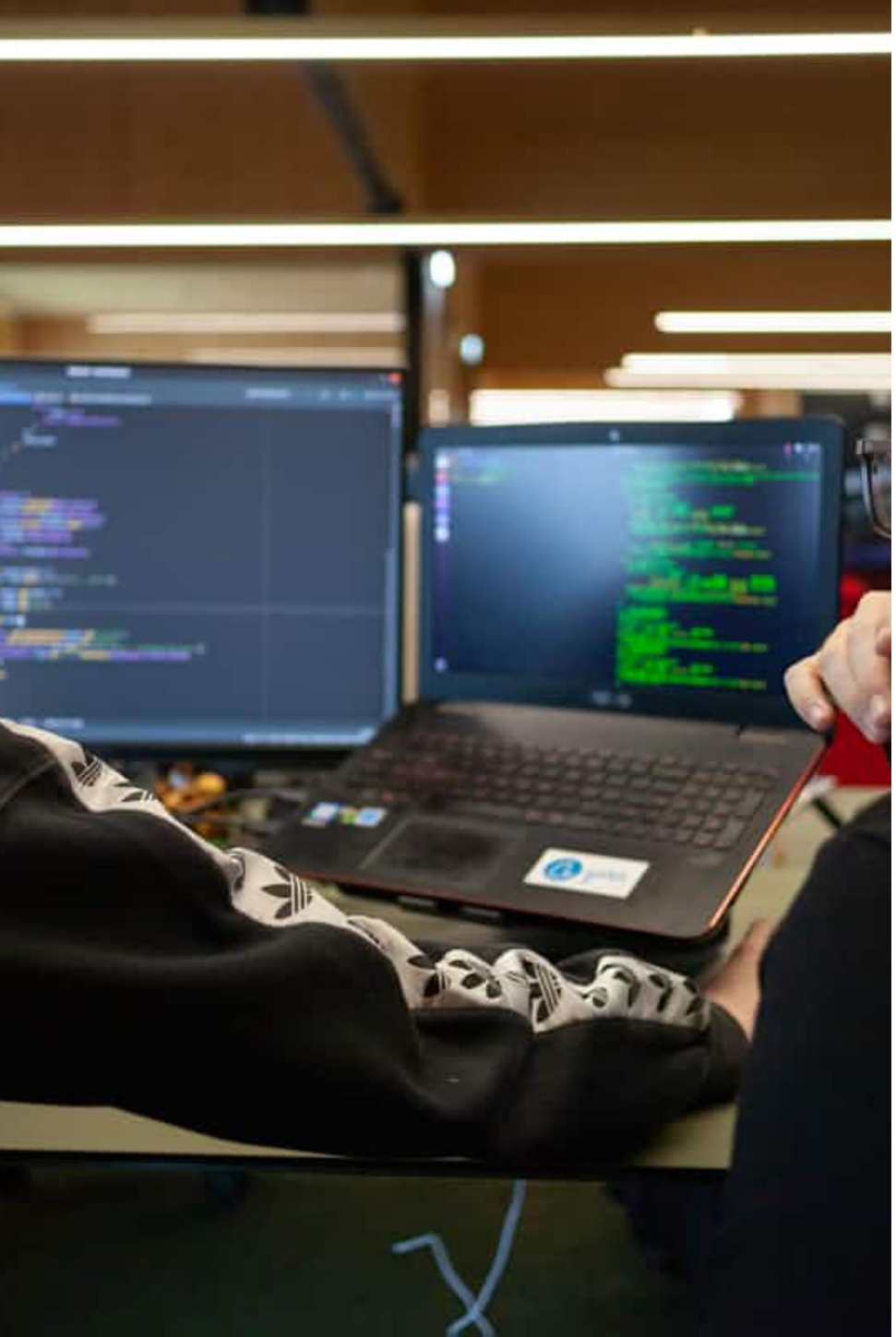
System Complaints

Source	Day							
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total
Email	𝍸	𝍸 |	𝍸	𝍸	𝍸		𝍸	12
Text	𝍸 𝍸 |	𝍸 |	𝍸 𝍸	𝍸	𝍸 |	𝍸	𝍸	29
Phone call	|	𝍸	|	|	𝍸	|		8
Total	11	10	8	6	7	3	4	49

Control Chart Graph

- To find out abnormalities and finding the current status.





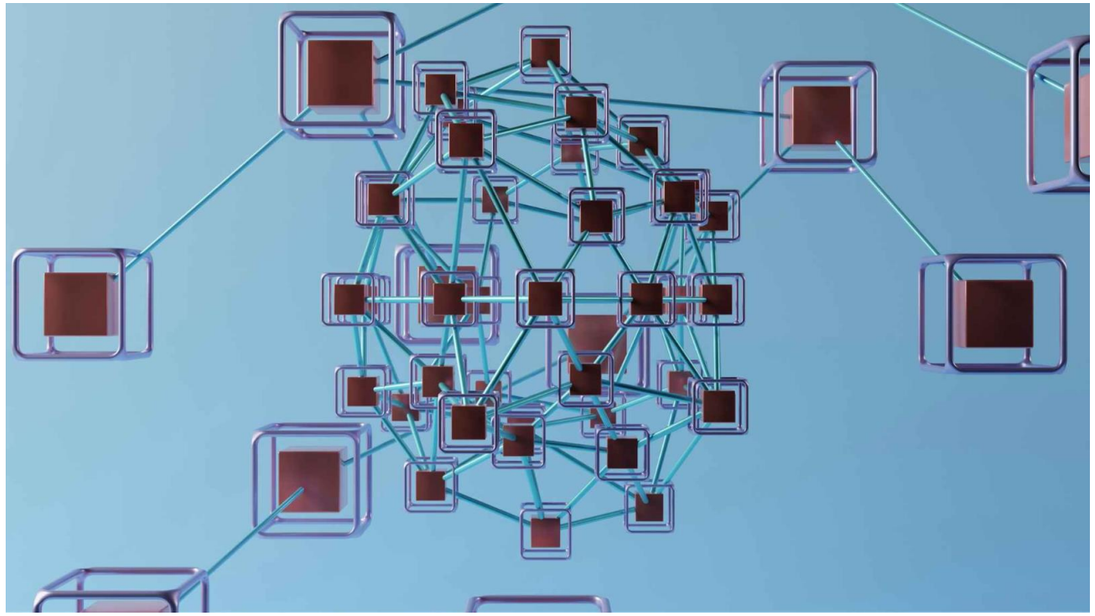
Software Quality Management

What is Software Quality?

Beyond "Bug-Free"

Software quality encompasses much more than the absence of defects. It's a comprehensive measure of how well software meets both specified requirements and user expectations.

Quality software delivers value consistently while maintaining performance, security, and user satisfaction across all interactions.



Core Quality Attributes

Reliability

Consistent performance under specified conditions without failure

Efficiency

Optimal use of system resources and fast response times

Usability

Intuitive interface and positive user experience

Maintainability

Easy to modify, debug, and enhance over time

Security

Protection against unauthorized access and data breaches

Why Quality Matters

User Satisfaction

Happy users drive product success. Quality software creates positive experiences that build loyalty and drive adoption rates.

Cost Reduction

Fewer bugs mean less rework and reduced maintenance costs. Investing in quality upfront saves significant resources long-term.

Reputation

Quality builds trust and credibility in the market. A strong reputation for reliability becomes a competitive advantage.

Risk Mitigation

Prevents critical system failures that could result in data loss, security breaches, or business disruption.



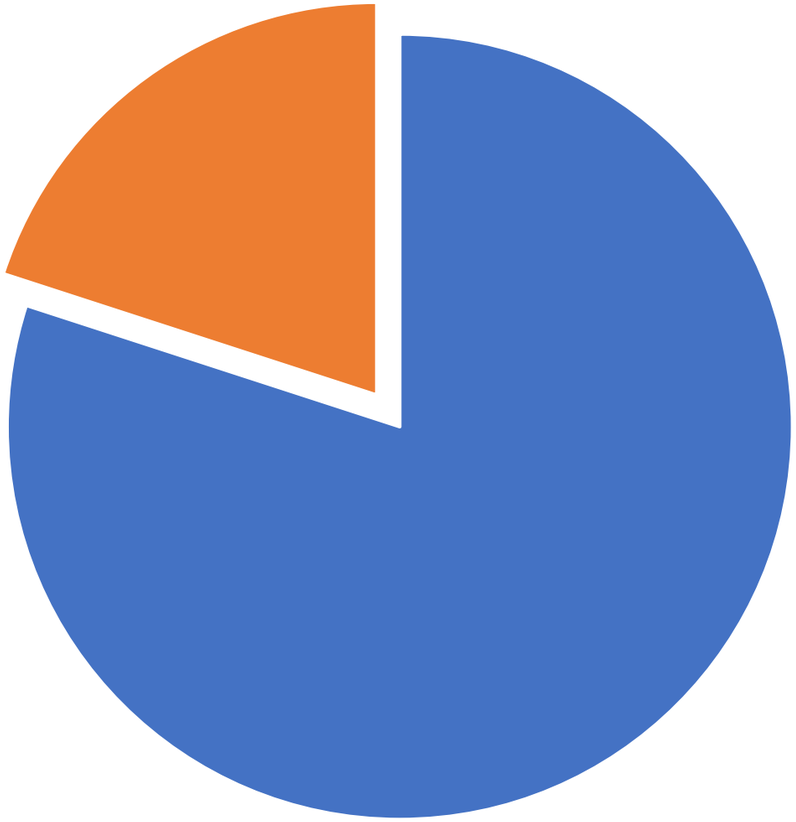
The Pareto Principle: 80/20 Rule

Core Principle

*For many events, roughly **80% of effects** come from **20% of causes**.*

Discovered by Italian economist Vilfredo Pareto, this principle is also known as the "Law of the Vital Few."

In software quality, identifying and addressing the critical few causes can dramatically improve overall system reliability.



Pareto Analysis in Software Quality

Core Concept

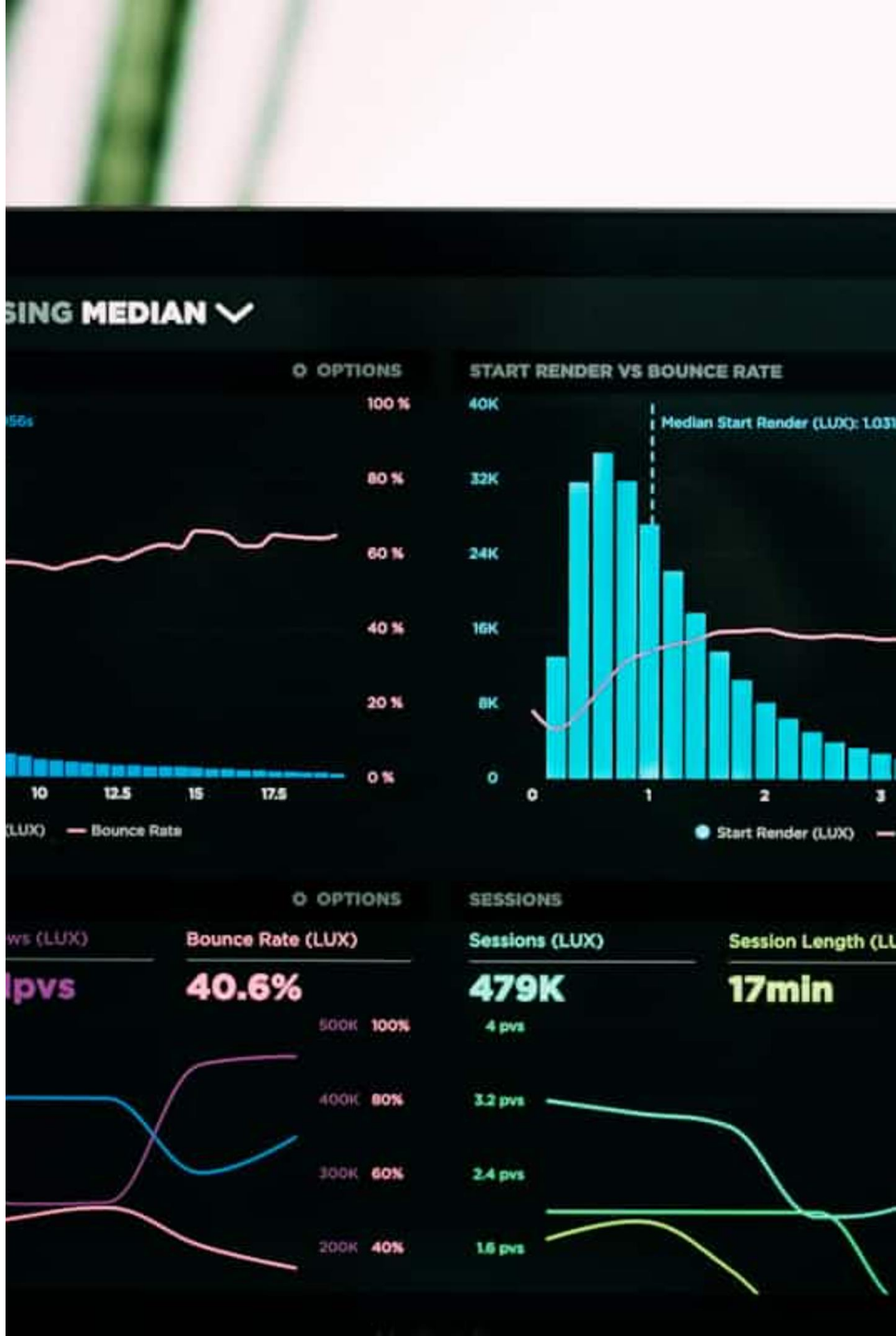
A small number of defect types cause the majority of problems in software systems.

Strategic Goal

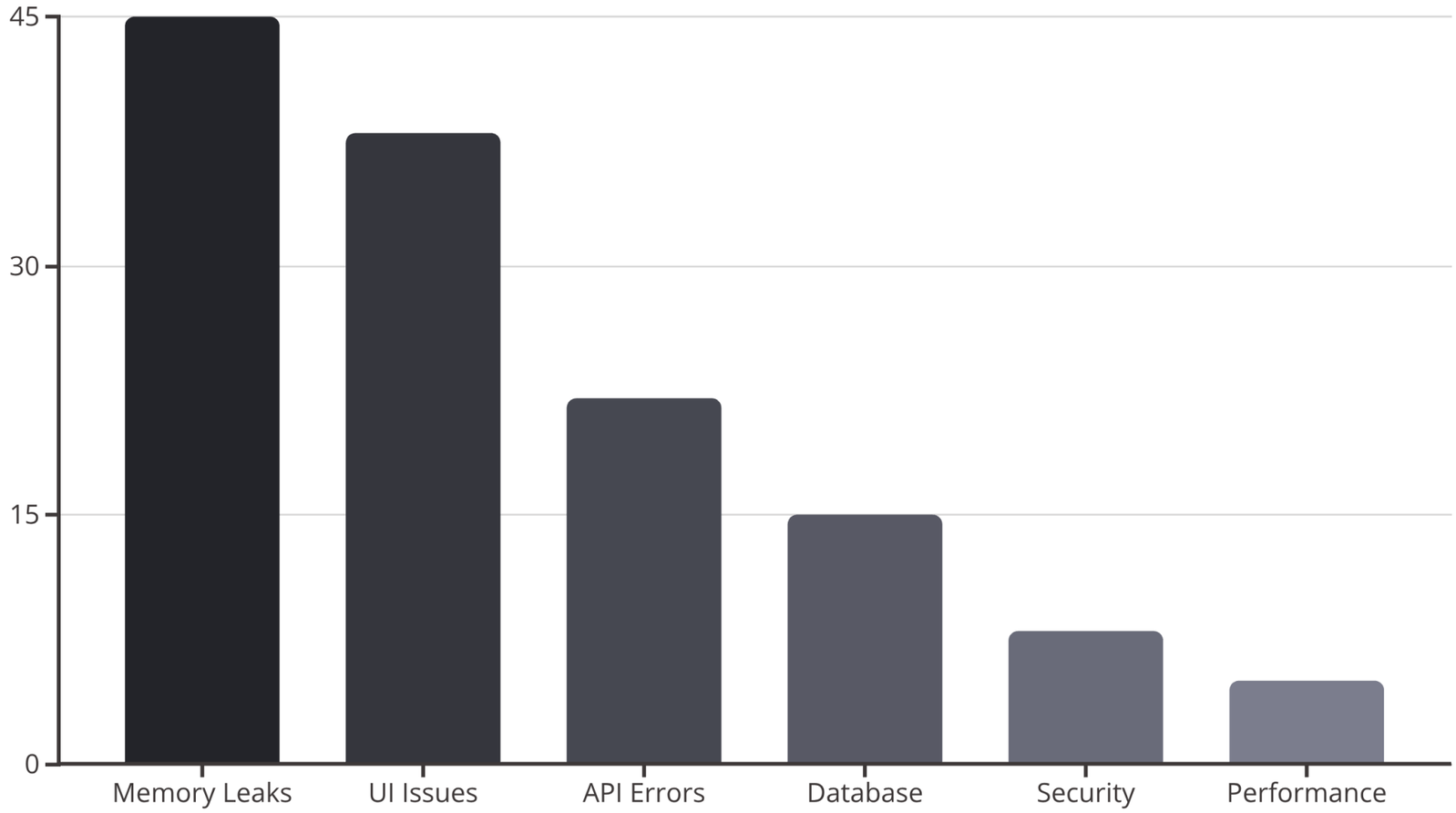
Identify the "vital few" root causes to prioritize quality improvement efforts effectively.

Real-World Scenarios

- *80% of bugs typically found in 20% of code modules*
- *80% of system crashes caused by 20% of functionalities*
- *80% of customer complaints relate to 20% of features*
- *80% of security vulnerabilities stem from 20% of attack vectors*



Pareto Chart: Defect Analysis Example



This Pareto chart reveals that memory leaks and UI issues account for over 60% of total defects. Focusing quality efforts on these two areas would yield the highest impact on overall system stability.

Phases

Initiation Phase (Identify Key Weight)

- Focus: In the initiation phase, use the 80/20 rule to identify which stakeholders, project requirements, or potential risks carry the most weight.
- Prioritization Example: If only 20% of stakeholders influence 80% of the project outcomes, early efforts must focus on engaging those key players.
- Question to Ask: Which stakeholder or risk has the biggest potential to either make or break the project?

Planning Phase (Concentrate Resources)

- Focus: During planning, the principle helps prioritize tasks and allocate resources efficiently.
- Task Prioritization: Analyze tasks to reveal the 20% of tasks that drive most of the project's value.
- Action: Concentrate planning, risk management activities, adequate resources, budgeting, and contingency plans specifically on those critical, most impactful elements.

Execution Phase (Resolve High-Impact Issues)

- Focus: During execution, the principle guides where to invest energy to maintain progress.
- • Issue Resolution: Identify the 20% of issues reported by your team that are causing 80% of the delays.
- Action: Address these high-impact issues promptly to minimize disruptions and keep the project on track. If a few key activities are consistently yielding most of the progress, maintain momentum by continually investing energy there.

Monitoring & Controlling (Track Key Health Metrics)

- Focus: Applying the Pareto Principle simplifies performance tracking by avoiding distraction from less significant data.
- Metrics: Concentrate on the top 20% of metrics—such as critical milestones or major risk indicators—that provide the most information about project health.
- Benefit: This selective focus aids in the early detection of problems, allowing for swift corrective actions before minor issues grow into major ones.

Closure Phase (Effective Review and Learning)

- Focus: The principle enables a more effective post-project review.
- Learning: Pinpoint the 20% of challenges that caused 80% of the hurdles.
- Benefit: This refined feedback provides clear insights into areas for improvement, helping to refine planning for future projects and making the entire project management process more efficient over time.

The Juran Trilogy

A Universal Approach

Developed by Dr. Joseph M. Juran, the trilogy provides a systematic framework for managing quality across any organization.

This breakthrough sequence views quality management as three interconnected, continuous processes that build upon each other.

Quality Planning: Foundation Phase

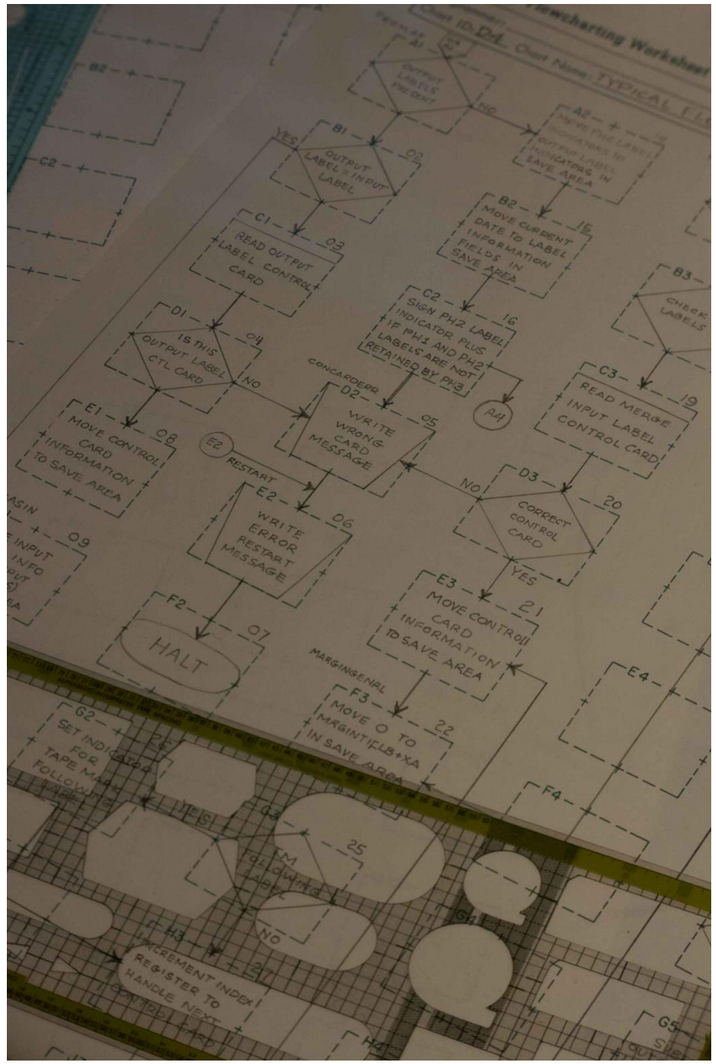
Proactive Approach

Design processes capable of meeting quality goals from the start.

Key Activities:

- *Identify customer and user needs*
- *Translate needs into specific product requirements*
- *Design products and processes to meet those requirements*
- *Establish quality metrics and success criteria*

> ***Example:*** *Defining clear performance metrics (response time < 1 second) and designing system architecture to achieve these goals before writing any code.*



Quality Control & Improvement

Quality Control (Reactive)

Monitor process performance and correct deviations in real-time.

- *Measure actual quality performance*
- *Compare against established goals*
- *Take corrective action on differences*

Running automated tests, code reviews, tracking defect density

Quality Improvement (Strategic)

Achieve breakthrough levels of performance through systematic improvement.

- *Establish improvement infrastructure*
- *Identify specific improvement projects*
- *Analyze root causes and develop solutions*

Post-release analysis of recurring bug patterns, then redesigning processes to prevent future occurrences



Six Sigma: The Statistical Perfectionist

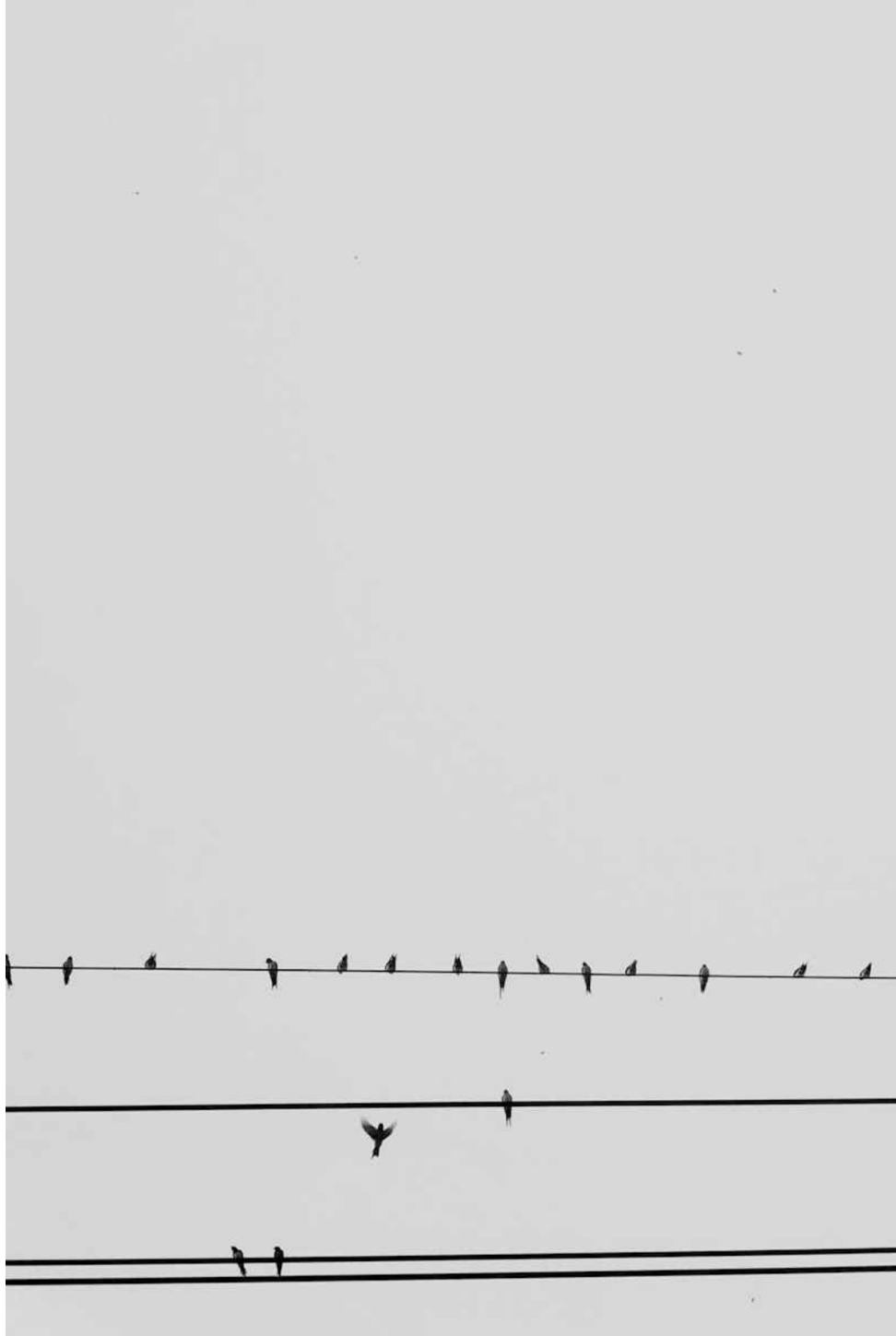
Six Sigma is a disciplined, statistical-based approach for eliminating defects in any process. The methodology targets a statistical goal of no more than **3.4 defects per million opportunities**.

The term 'sigma' (σ) represents standard deviation. A six-sigma process means the nearest specification limit is six standard deviations from the process mean.

Statistical Goal

Goal = 6σ ⟹ 3.4 DPMO

This represents near-perfect quality through rigorous statistical control and continuous measurement.



DMAIC: The Five-Phase Framework

01

Define

Define the problem, project goals, and customer deliverables with precision and clarity.

02

Measure

Measure current process performance to establish a reliable baseline for improvement.

03

Analyze

Analyze data systematically to determine root causes of defects and variations.

04

Improve

Improve the process by eliminating identified root causes through targeted solutions.

05

Control

Control future performance to ensure improvements are sustained over time.

Six Sigma in Action
E-commerce

Problem

5% checkout failure rate impacting revenue and customer satisfaction

Analysis

80% of failures traced to specific browser-payment gateway combination

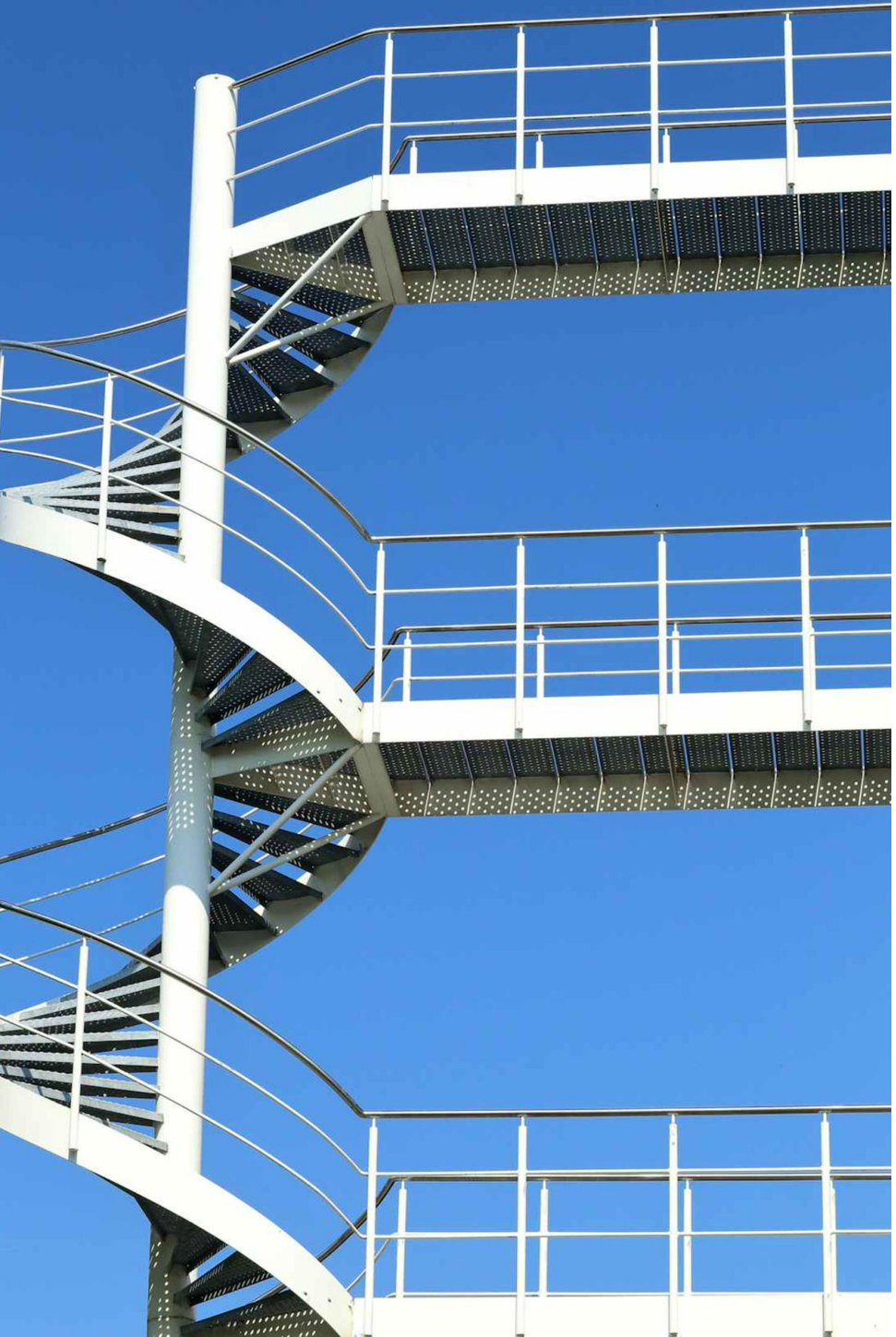
Solution

Targeted module rewrite and enhanced testing suite deployment

Result

Failure rate reduced to <0.1% with continuous monitoring dashboard





Capability Maturity Model (CMM)

CMM provides a structured roadmap for improving an organization's software development processes.

Developed by Carnegie Mellon's Software Engineering Institute, it classifies organizations into five distinct maturity levels.

Unlike Six Sigma's focus on specific processes, **CMM addresses entire organizational capability**, creating systematic improvement across all development activities.

The Five Maturity Levels

Level 1: Initial (Chaotic)

Unpredictable, reactive processes. Success depends on individual heroics and luck rather than systematic approaches.

Level 2: Repeatable (Managed)

Basic project management processes established. Success from similar projects can be systematically repeated.

Level 3: Defined (Standardized)

Documented, standardized processes integrated across the organization. All projects use approved, tailored versions.

Level 4: Quantitatively Managed

Quantitative quality goals set for products and processes. Detailed measures collected and controlled systematically.

Level 5: Optimizing

Organization focused on continuous improvement. Quantitative feedback drives identification and implementation of enhancements.

ISO 9001: International Quality Standard

Core Framework

ISO 9001 specifies requirements for a Quality Management System (QMS), focusing on consistent, documented processes rather than prescriptive methodologies.

The key mantra: **"Say what you do, do what you say, and prove it."**

Fundamental Principles

- Customer focus and requirements
- Process-driven approach
- Evidence-based decision making
- Continual organizational improvement





CMM vs. ISO 9001: Different Approaches

CMM Approach

Prescriptive roadmap specifying exact software engineering and management practices required at each maturity level.

ISO 9001 Approach

Framework-based system requiring documented processes, consistent execution, and verifiable records without dictating specific methods.

Complementary Quality Strategies

Juran Trilogy

Quality planning, control, and improvement framework

Pareto Analysis

Identify vital few problems causing most issues



Six Sigma

Data-driven defect elimination methodology

ISO 9001

International quality management system standard

CMM

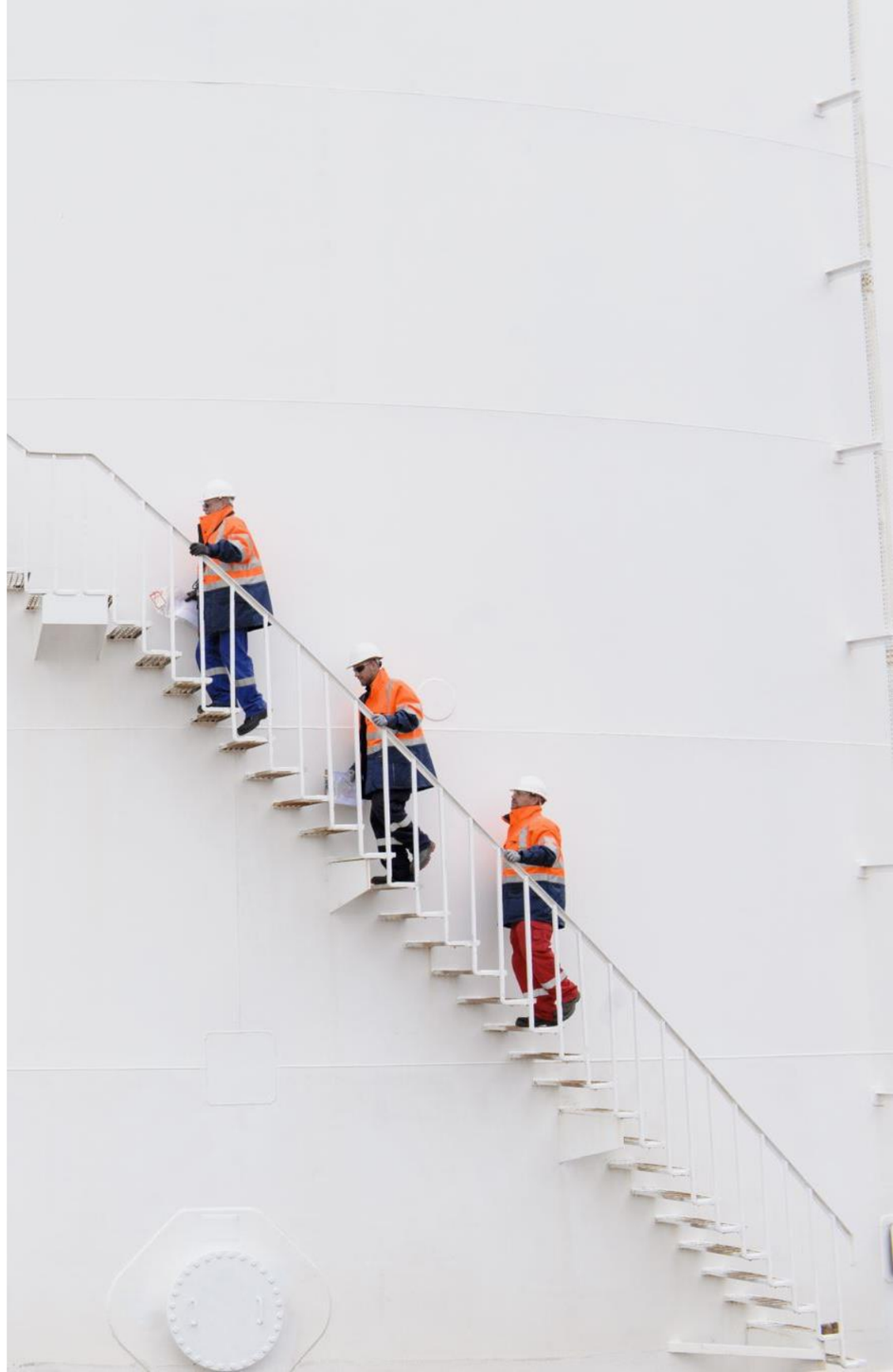
Organizational maturity development roadmap

These methodologies work together synergistically. Use Pareto to identify problems, run Six Sigma projects for solutions, within CMM-mature organizations following ISO 9001 standards.

Quality is Not an Accident

Quality results from **intelligent effort**, **structured processes**, and a **culture of continuous improvement**. These methodologies provide the framework for achieving excellence in software engineering organizations.

Success requires commitment to data-driven decision making, systematic process improvement, and organizational maturity development.



Software Quality

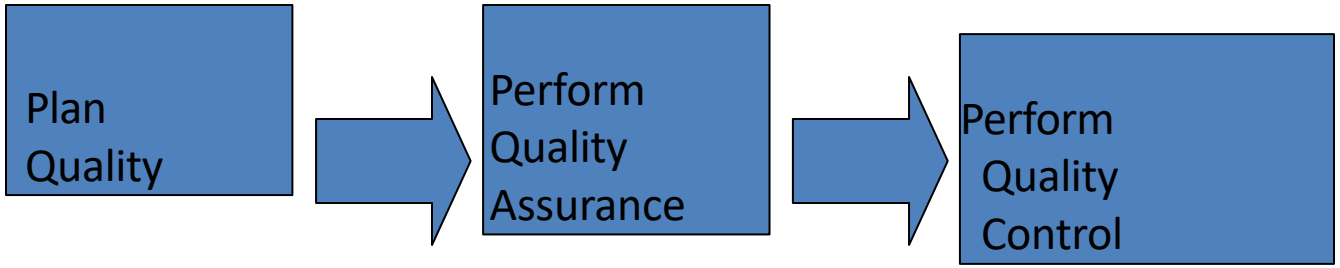
- Quality Planning, Quality Assurance and Quality Control
- Quality Control Tools

Definitions of Quality

- Quality is meeting or exceeding customer's expectations
- Quality is the totality of features and characteristics of a product or service that bears on its ability to satisfy stated or implied needs.
- Quality is sum of the attributes that can describe a project or product.
- Quality refers to meeting specifications and performance standards.
- Managing the quality of the software process and products is important.

How Do We Manage Quality?

- Three processes
  - Plan Quality
  - Perform Quality Assurance
  - Perform Quality Control



Quality management activities

- **Quality assurance**
  - Establish organisational procedures and standards for quality
- **Quality planning**
  - Select applicable procedures and standards for a particular project and modify these as required
- **Quality control**
  - Ensure that procedures and standards are followed by the software development team
- Quality management should be separate from project management to ensure independence

Quality Assurance & Quality Control



Quality assurance aims to assure that quality work and quality deliverables will be built in before work is done



Quality control aims to determine that quality work and quality deliverables did occur after work was done



Quality control is conducted by inspection and testing Of materials, workmanship and product

Quality Assurance Vs Control

- Quality Assurance is process oriented and focuses on defect prevention.
- QA is a set of activities for ensuring quality in the processes by which products are developed.

- Whereas

- Quality Control is product oriented and focuses on defect identification.
- QC is a set of activities for ensuring quality in products. The activities focus on identifying defects in the actual products produced.

Focus- Assurance Vs Control

- QA aims to prevent defects with a focus on the process used to make the product. It is a proactive quality process.
- QC aims to identify (and correct) defects in the finished product. Quality control, therefore, is a reactive process.

Software Quality Assurance

- **Software quality assurance** (**SQA**) consists of a means of monitoring the software engineering processes and methods used to ensure quality.

- The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI.

SQA

- SQA encompasses the entire software development process, which includes processes.
  - Requirements definition,
  - Software design,
  - Coding,
  - Source code control,
  - Code reviews,
  - Change management,
  - Configuration management,
  - Testing,
  - Release management, and
  - Product integration.

Quality planning

- A quality plan sets out the desired product qualities and how these are assessed and define the most significant quality attributes.
- It should define the quality assessment process.
- It should set out which organisational standards should be applied and, if necessary, define new standards.

Quality plan structure

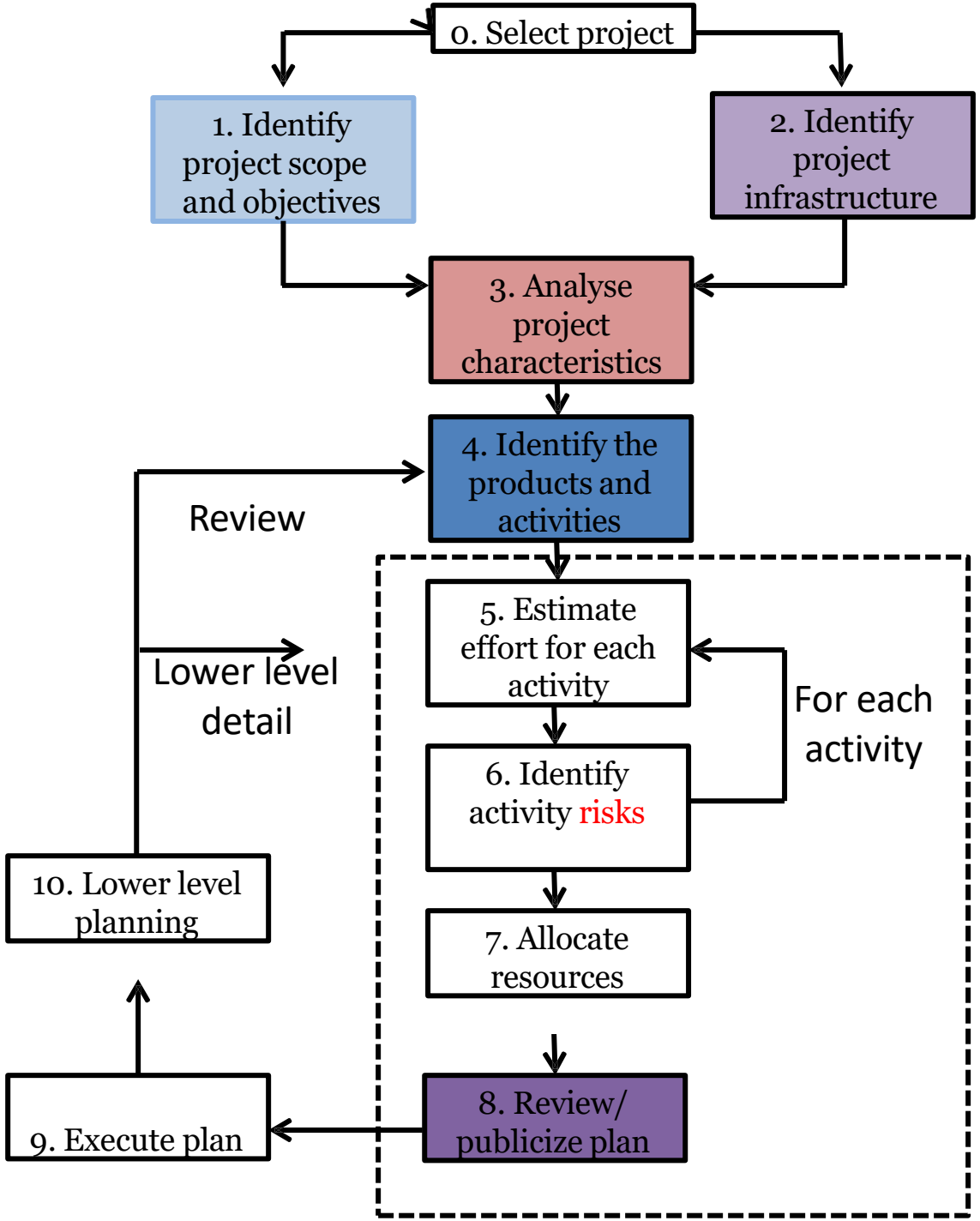
- Product introduction
- Product plans
- Process descriptions
- Quality goals
- Risks and risk management
- Quality plans should be short, succinct documents
  - If they are too long, no-one will read them

Quality plans

- quality standards and procedures should be documented in an organization's *quality manual*
- for each separate project, the quality needs should be assessed
- select the level of quality assurance needed for the project and document in a *quality plan*

Typical contents of a quality plan

- scope of plan
- references to other documents
- quality management, including organization, tasks, and responsibilities
- documentation to be produced
- standards, practices and conventions
- reviews and audits
- scope of plan
- references to other documents
- quality management, including organization, tasks, and responsibilities
- documentation to be produced
- standards, practices and conventions
- reviews and audits

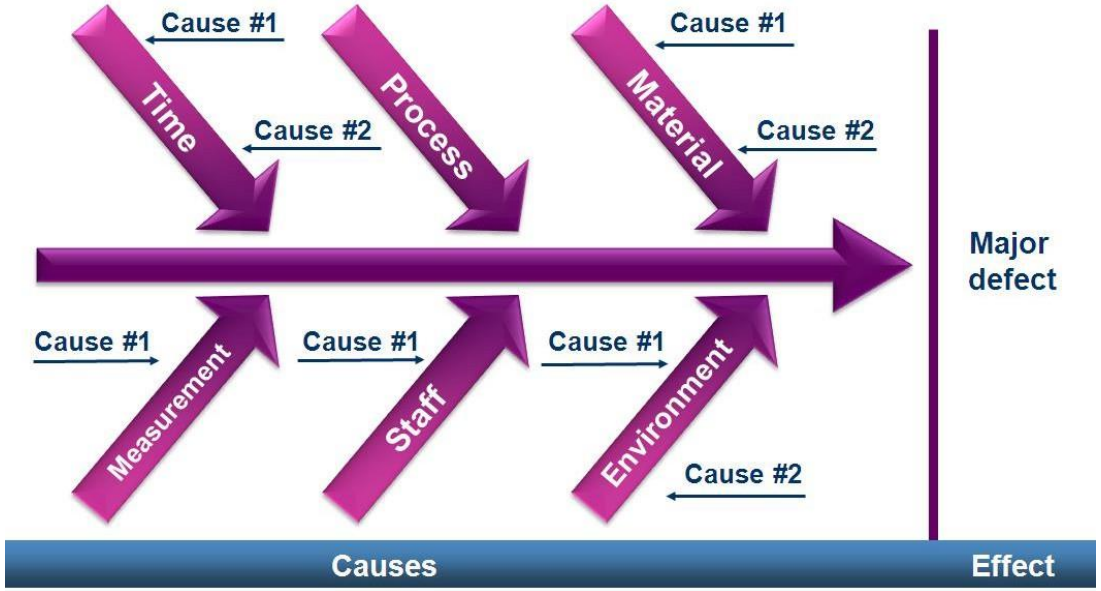

0. Select project
1. Identify project scope and objectives
2. Identify project infrastructure
3. Analyse project characteristics
4. Identify the products and activities
Review
5. Estimate effort for each activity
Lower level detail
For each activity
6. Identify activity risks
7. Allocate resources
10. Lower level planning
8. Review/ publicize plan
9. Execute plan

7 Basic Quality Control Tools (7QC)

- Cause & Effect Diagram
- Flowchart
- Histogram
- Pareto Chart
- Check Sheet
- Scatter Diagram
- Control Chart

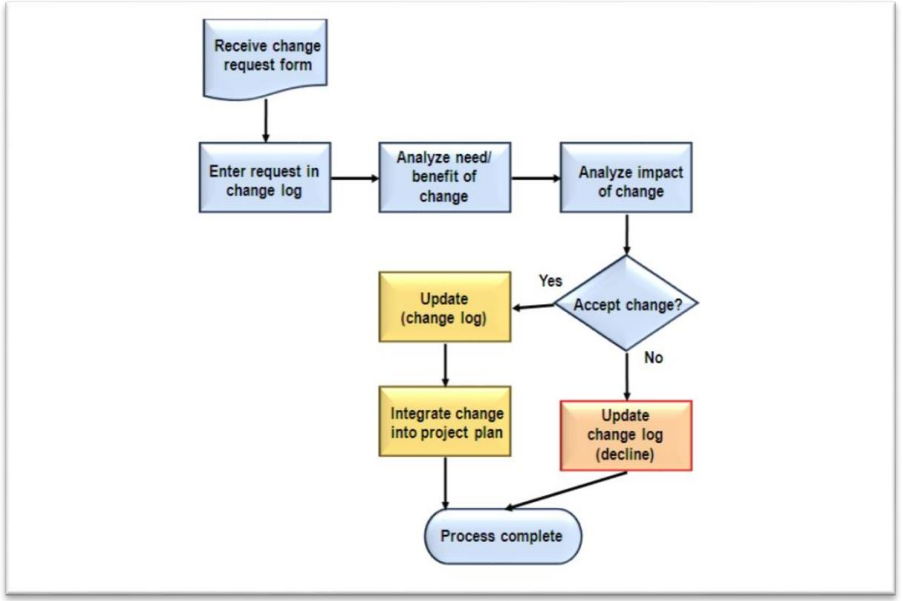
Cause & Effect Diagram

- To identify cause and effect relationship



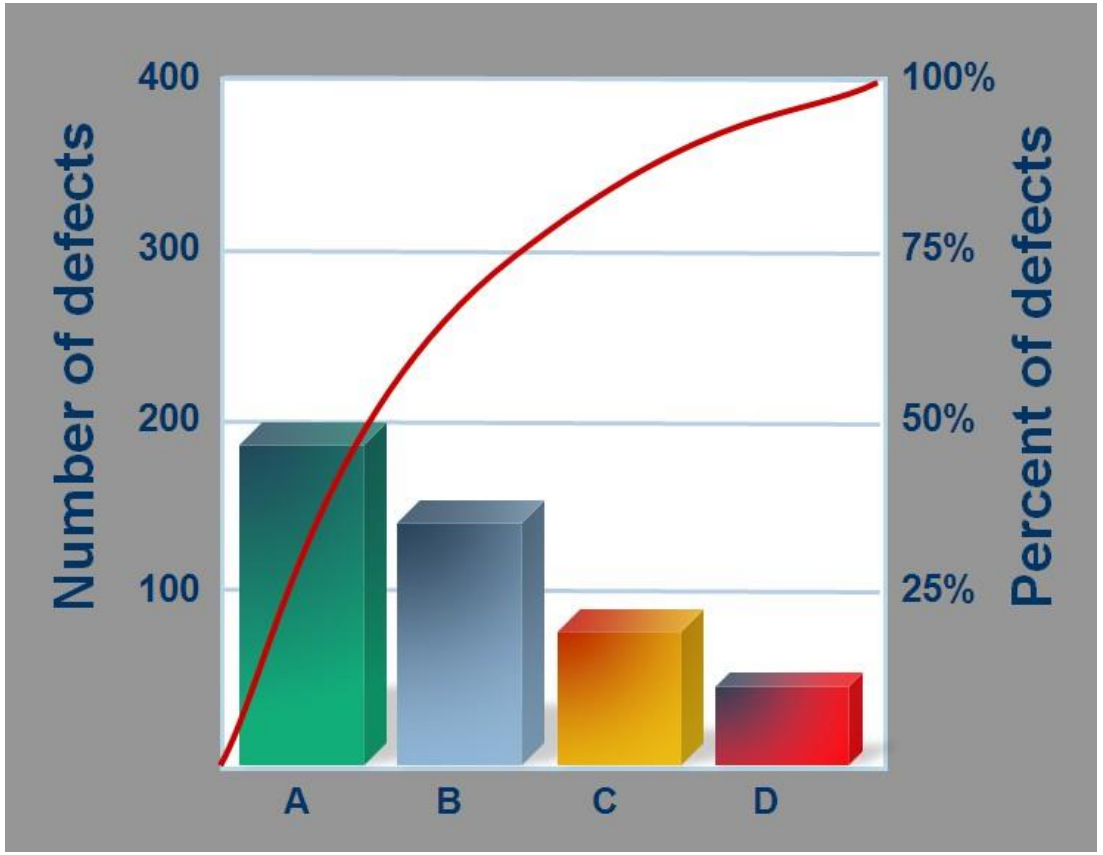
Flow Charts

- Basic processing performed when collecting data.
- Flowcharts show the logical steps in a process and how various elements within a system are related.
- They can be used to determine and analyse potential problems in quality planning and quality control.



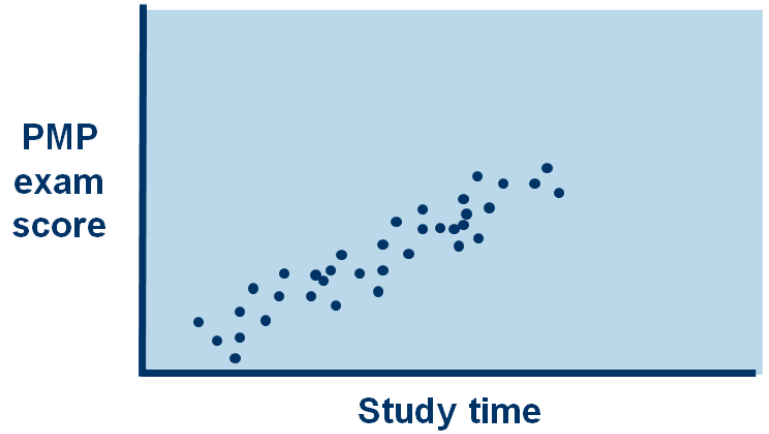
Pareto diagram

- To identify current status and issues.
- Based on Pareto's principle, frequently referred to as the 80/20 rule or 80/20 principle. Which means that eighty percent of the cost of defects are caused by twenty percent of the problems.



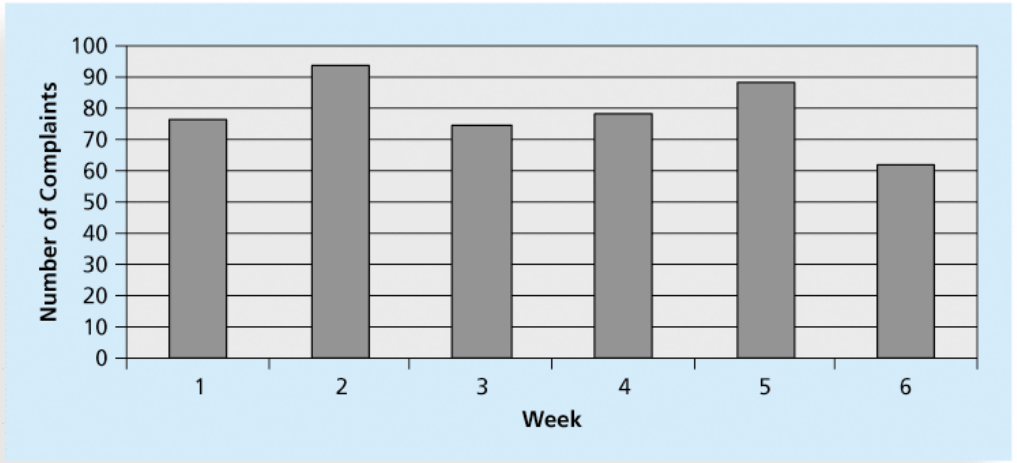
Scatter Diagram

- To identify relationship between two things.



Histogram

- To see the distribution of data.
- A **histogram** is a bar graph of a distribution of variables.



Check Sheets

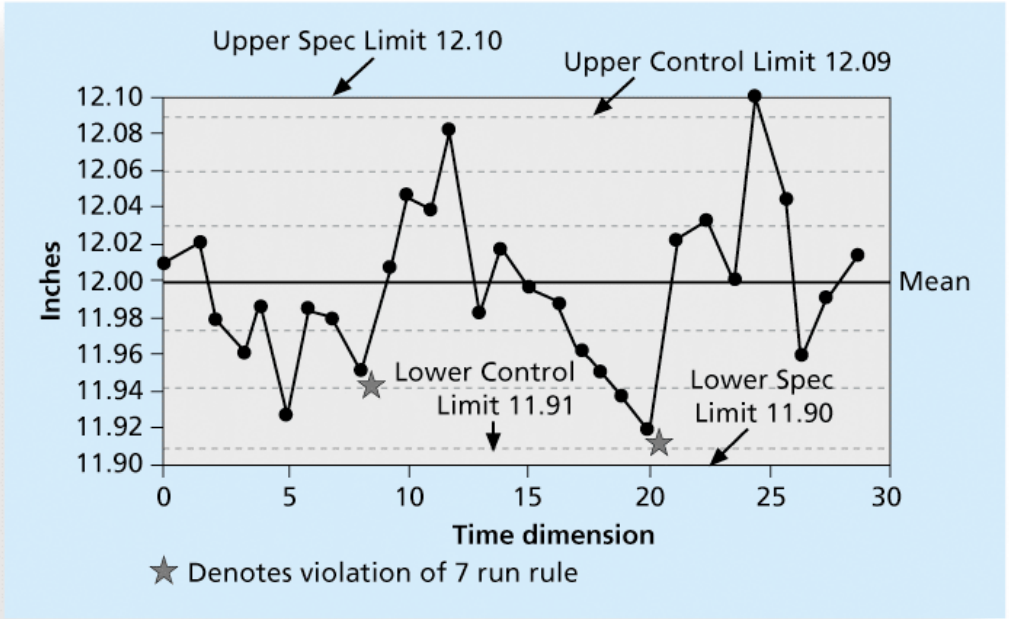
- To record data collections

System Complaints

Source	Day							
	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total
Email	𝍸	𝍸	𝍸	𝍸	𝍸		𝍸	12
Text	𝍸 𝍸	𝍸	𝍸	𝍸	𝍸	𝍸	𝍸	29
Phone call	𝍸	𝍸	𝍸	𝍸	𝍸	𝍸		8
Total	11	10	8	6	7	3	4	49

Control Chart Graph

- To find out abnormalities and finding the current status.



Earned Value Management (EVM)
Portfolio Management

Earned Value Management

- ***Earned Value Management*** (EVM) is a systematic approach to the integration and measurement of cost, schedule, and technical (scope) accomplishments on a project or task.

- ***Earned Value*** (EV) directly links all three primary project success Metrics (cost, schedule, and performance).

Earned Value Management

By using Earned Value and implementing an Earned Value Management System (EVMS), the following questions can be answered objectively:

- Where have we been?
- Where are we now?
- Where are we going?

Traditional management Vs Earned Value Management



Traditional management provides you with...



How much money and time a particular job is likely to require prior to starting and once stated, how much money was spent at any given time.



While **Earned Value Management provides you with...**



How much money and time a particular job is likely to require prior to starting and once stated, how much money was spent at any given time.



Earned Value Terms

- Planned value (PV)
- Earned value (EV)
- Actual cost of work performed (AC)
- Schedule variance (SV) and Schedule performance index (SPI)
- Cost variance (CV) and Cost performance index (CPI)
- Budgeted cost at completion (BAC)
- Estimate at completion (EAC)

Steps in Earned Value Management

1. ***Clearly define each activity*** including its resource needs and budget.
2. ***Create usage schedules*** for activities and resources.
3. ***Develop a time-phased budget*** (PV).
4. ***Total the actual costs*** of doing each task (AC).
5. ***Calculate*** both the budget variance (CV) and schedule variance (SV).

Earned Value Example

<u>Schedule Variances</u>

Planned Value (PV) = 103

Earned Value (EV) = 44

Schedule Performance Index = .43 = 44/103 = EV/PV

Estimated Time to Completion = (1/.43)x7=16.3 months

<u>Cost Variances</u>

Cumulative Actual Cost of Work Performed (AC) = 78

Cost Performance Index = .56 = 44/78 = EV/AC

Estimated Cost to Completion = $210,714 = (1/.56)x$118,000

An Earned Value Management System (EVMS) framework

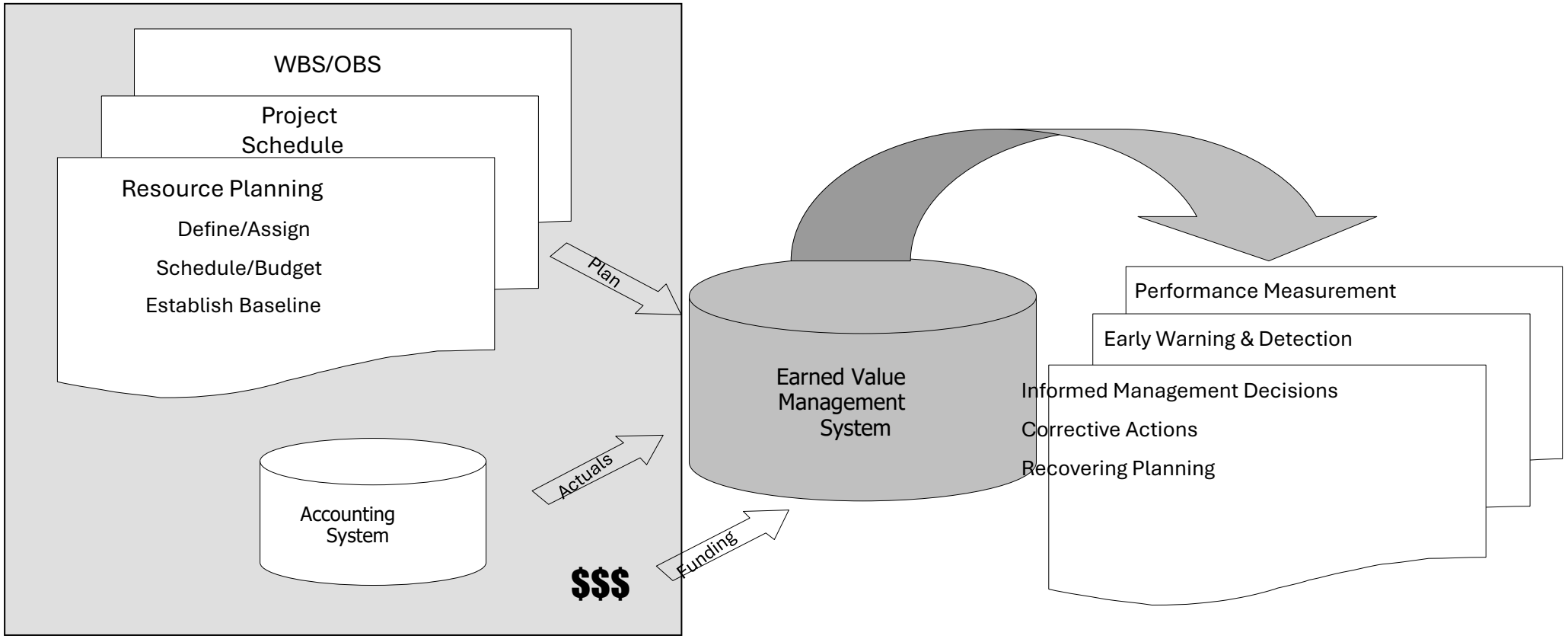
The EVMS framework can be divided into three phases:

1. Inputs - what is needed to implement Earned Value

2. Earned Value Methods – formulas, metrics and performance measurements used

3. Outputs – reporting requirements (structure, time-phases, details)

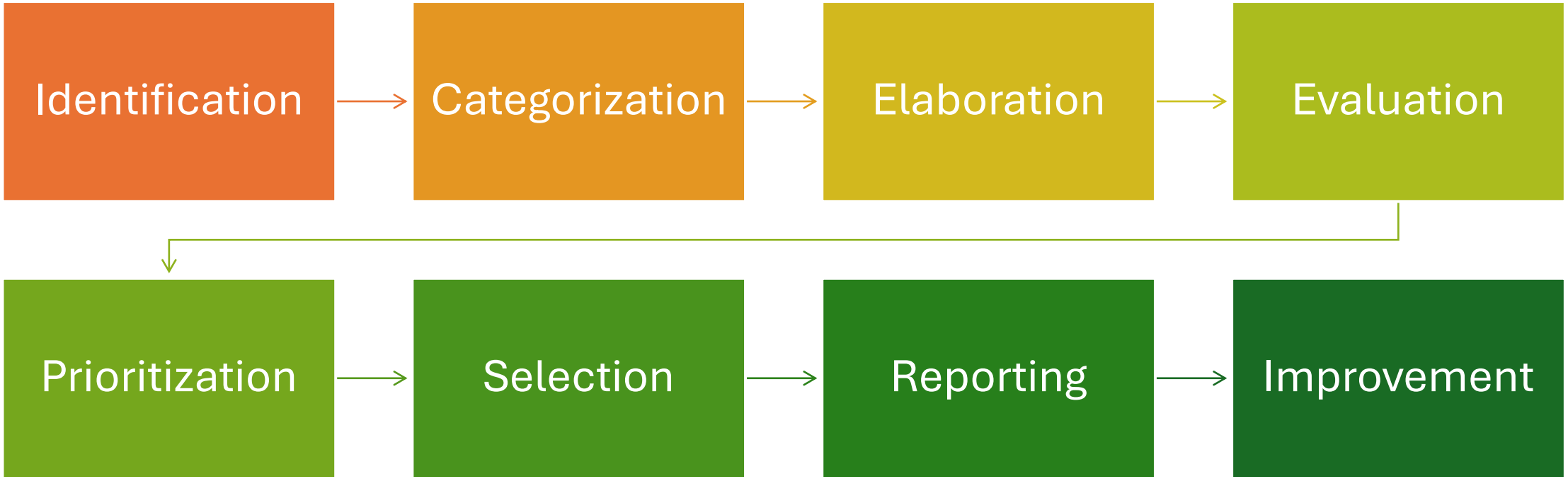
An Earned Value Management System (EVMS) framework



WHAT IS PORTFOLIO MANAGEMENT?

Gathering	Gathering candidate ideas for investment from various sources (internal and external)
Considering	Considering candidate and approved projects based upon an analytic decision-making framework
Choosing	Choosing highly ranked projects that together meet constraints for budget and resources.
Reporting	Reporting on changes, progress, and results in the portfolio
Reconsidering	Reconsidering the portfolio on an ongoing basis
Practicing	Practicing continuous improvement of Portfolio Management processes.

EIGHT KEY PROCESSES ARE INVOLVED IN CHOOSING THE RIGHT PORTFOLIO

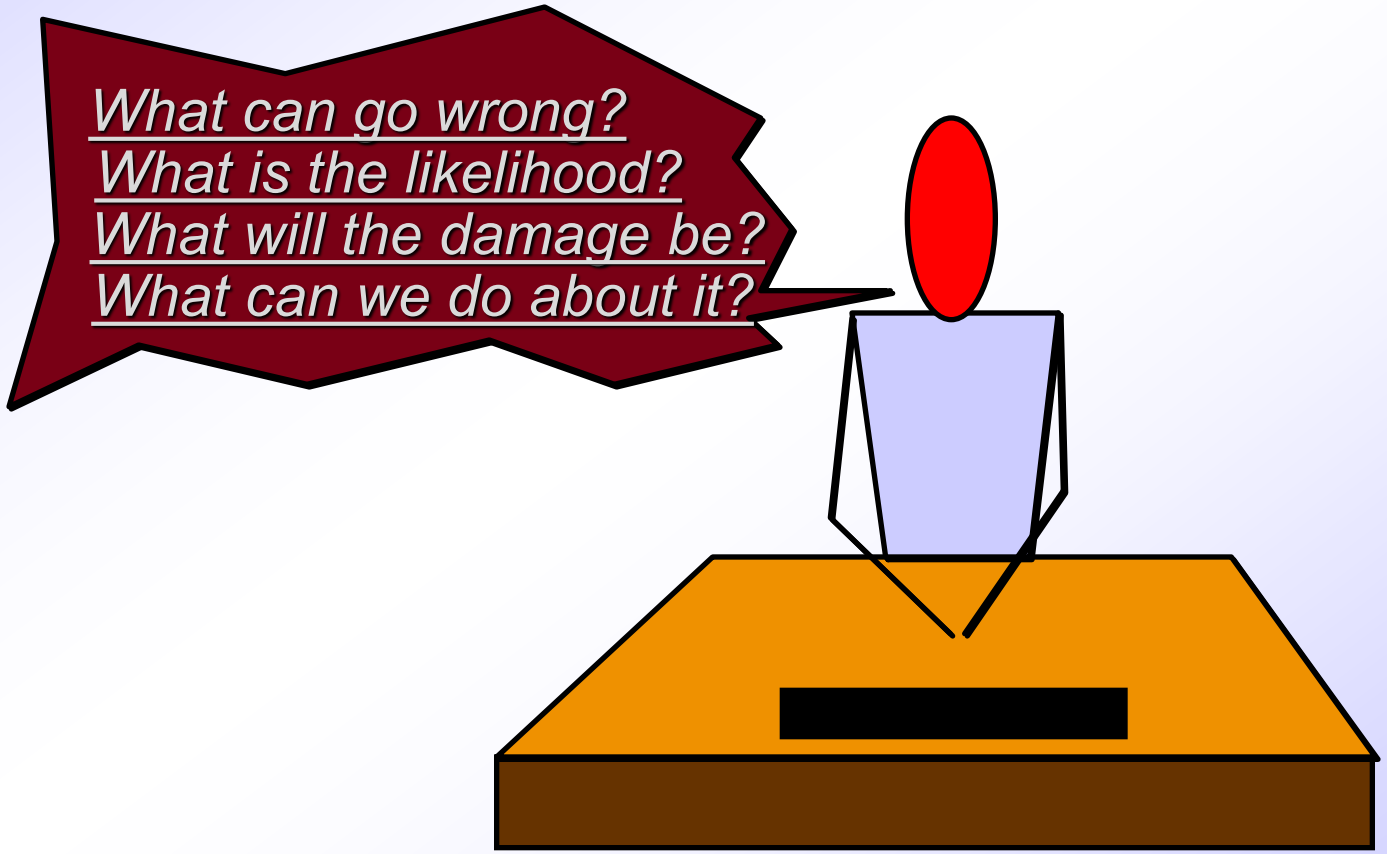


Risk Management and Plan

- Introduction
- Defining Risk
- Risk Management Plan
- Risk Identification
- Risk Analysis

(Source: Pressman, R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2005)

Project Risks



Risk Management Plan



Steps for Risk Management

Identifying possible risks

Analyze each risk to estimate the probability of each occurrence.

Rank the risks by probability and impact
- Impact may be negligible, marginal, critical, and catastrophic

Develop a contingency plan to manage those risks having high probability and high impact

Risk Management Process

- **Planning risk management** : Deciding how to approach and plan the risk management activities for the project.
- **Risk Identification**: Determining which risks are likely to affect a project and documenting the characteristics.
- **Performing qualitative risk analysis**: Prioritizing risks based on their probability and impact of occurrence

Cont…

- **Performing quantitative risk analysis**: Numerical estimation of the effects of risks on project objectives.
- **Planning risk responses**
- **Monitoring and Controlling Risk**:

Planning Risk Management

- The main output of this process is a **risk management plan**—a plan that documents the procedures for managing risk throughout a project.
- The project team should review project documents and understand the organization's and the sponsor's approaches to risk.

Risk Identification

Risk Identification/Characterization

There are many different types of risks which affects the software project:

1. Technology risks
2. Tools risks
3. Estimation risks
4. People risks
5. Requirement risks
6. Organizational risks

What Is Software Risk?

- **Software risk** refers to the possibility of loss in a software development project. It is generally caused due to lack of information, control or time
- Loss can be anything, increase in production cost, development of poor quality software, Project cancellation or failure to meet objectives.

Types of software risks

- A software risk can be of two types
  - (1) internal risks that are within the control of the project manager and
  - (2) external risks that are beyond the control of project manager.

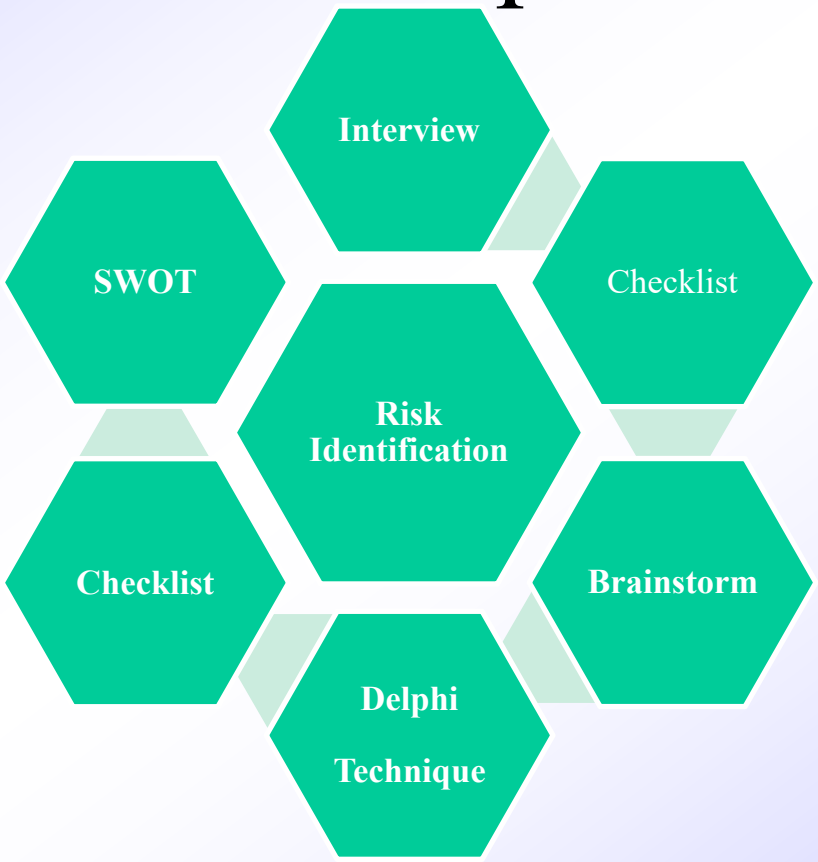
Risk Identification

- Risk identification is a systematic attempt to specify threats to the project plan.
- Identifying risks is the process of understanding what potential events might hurt or enhance a particular project.
- Another consideration is the likelihood of advanced discovery
- The project manager takes a first step toward avoiding them when possible and controlling them when necessary.

- Generic risks
  - Risks that are a potential threat to every software project
- Product-specific risks
  - Risks that can be identified only by those a with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built
  - This requires examination of the project plan and the statement of scope
  - "What special characteristics of this product may threaten our project plan?"

- Risk identification tools and techniques include:
  - Brainstorming
  - The Delphi Technique
  - Interviewing
  - SWOT analysis

Risk Analysis/Identification Techniques

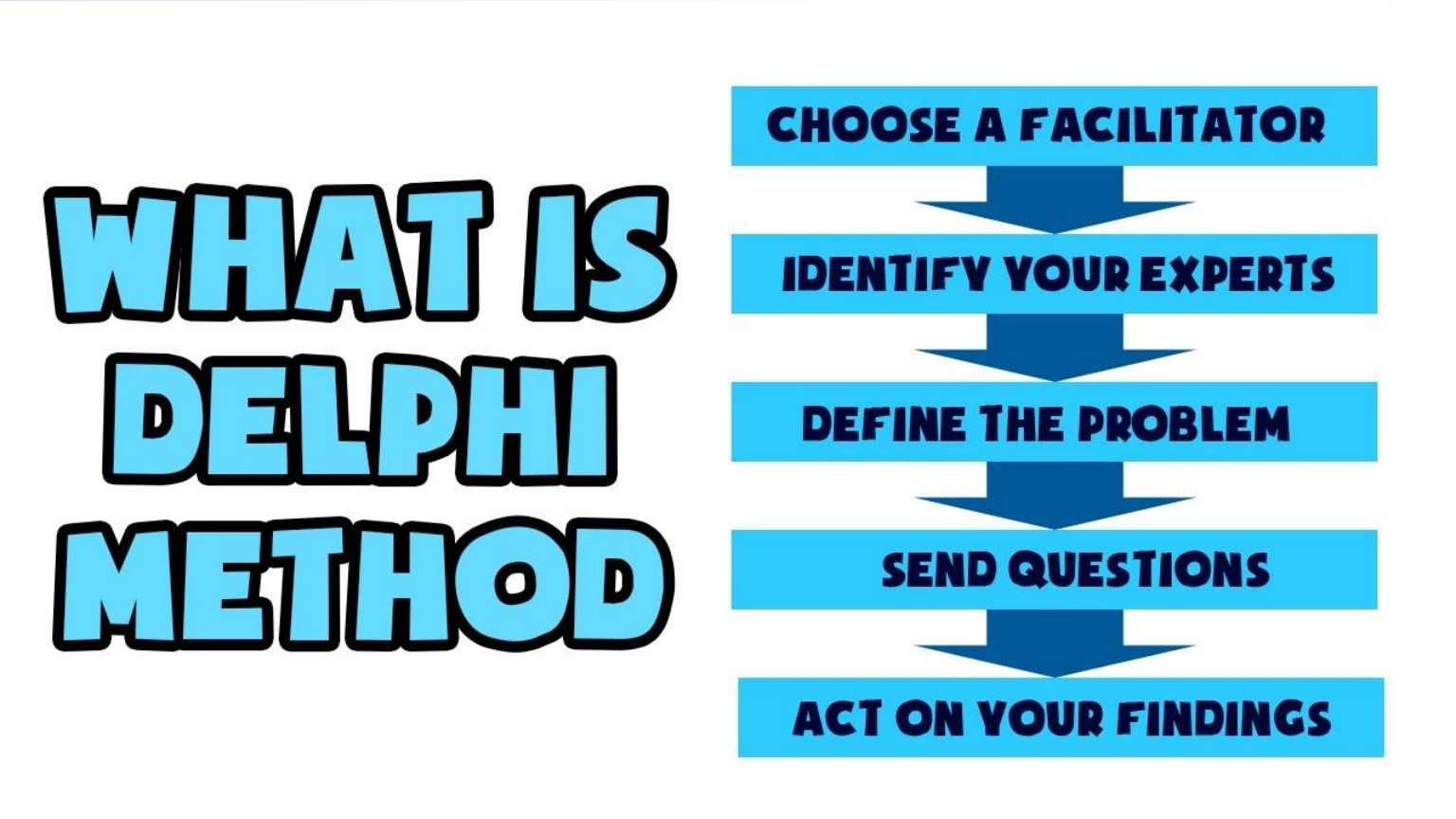


Risk Identification Techniques

Brainstorming

- Encourages free and open participation by all project team members
- Creates greater sense of ownership of project risks
- Builds team commitment to managing risks during the project
- Enables spontaneous identification of risks in a **"no wrong answer"** environment
- Allows team members to build on each other's ideas
- Supports creative and unique risk-gathering process
- Also used to determine best possible solutions to emerging problems/issues

Delphi Method





The **Delphi Technique** is used to derive a consensus among a panel of experts who make predictions about future developments.

Uses repeated rounds of questioning and written responses and avoids the biasing effects possible in oral methods, such as brainstorming

Applications:

- Effective when group of experts is available
- Useful when decisions depend on **qualitative criteria**
- Example: Project cost estimation

Interviewing

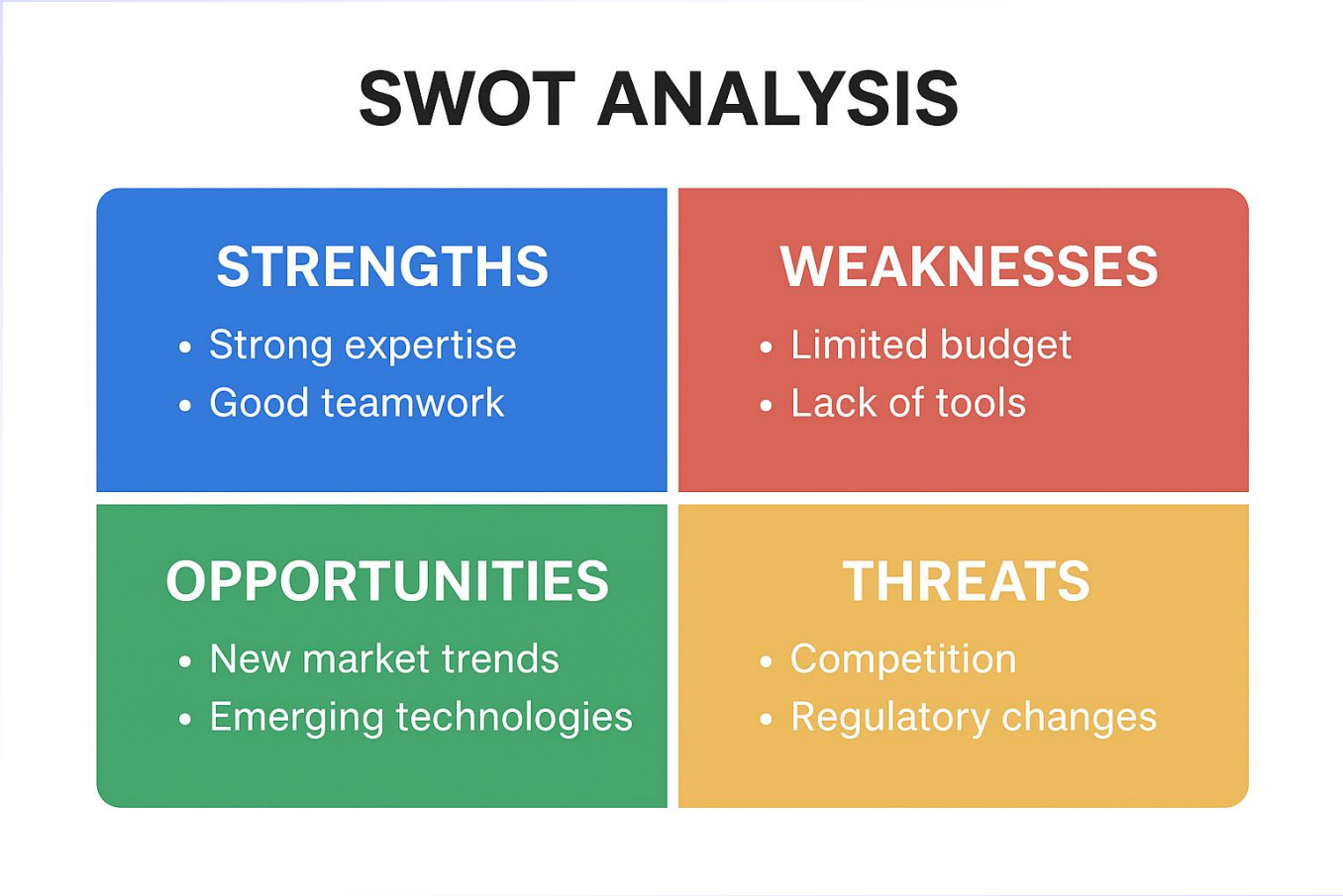
- **Interviewing** is a fact-finding technique for collecting information in face-to-face, phone, e-mail, or instant-messaging discussions
- Interviewing people with similar project experience is an important tool for identifying potential risks

SWOT Analysis - Strengths-Weaknesses-Opportunities-Threat (SWOT) helpful for identifying risks within greater organization context. It is generally used as planning tool for analyzing business.

Applications of SWOT Analysis

- Helps in **strategic planning** and decision-making
- Identifies **internal capabilities** vs. **external environment**
- Assists in **risk management** by anticipating challenges
- Useful for **project planning, business development, and problem-solving**

SWOT Matrix


SWOT ANALYSIS
STRENGTHS
• Strong expertise
• Good teamwork
WEAKNESSES
• Limited budget
• Lack of tools
OPPORTUNITIES
• New market trends
• Emerging technologies
THREATS
• Competition
• Regulatory changes

Qualitative Risk Analysis

Definition

- A technique to **prioritize risks** for further analysis or action by assessing their **probability** (likelihood) and **impact** (consequences).
- It uses **subjective, descriptive, and non-numerical data** (expert judgment, ranking, categories) instead of numbers.

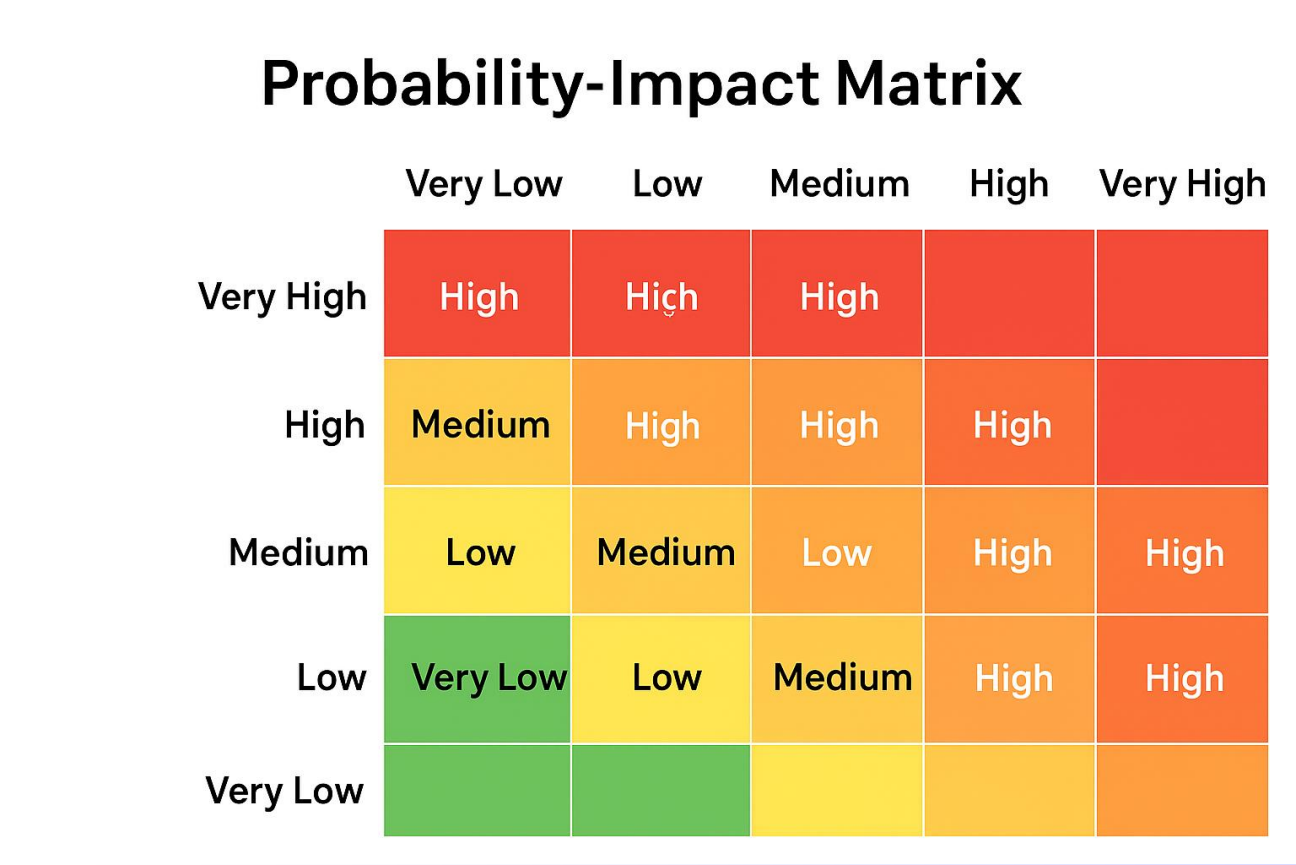
Steps of Qualitative Risk Analysis

- Identify risks from Risk Register
- Assess Probability
- Assess Impact
- Probability Risk Matrix
- Categorize Risk
- Update Risk Register

Risk Register

Risk ID	Risk Description	Category	Probability	Impact	Priority	Response Strategy	Owner	Status
R1	Delay in supplier delivery	Schedule	High	High	Critical	Mitigate (alternative supplier)	Project Manager	Open
R2	Requirement changes	Scope	Medium	High	High	Accept + Contingency	BA Lead	Open
R3	Team member turnover	People	Low	Medium	Medium	Transfer (contracting)	HR Manager	Monitoring
R4	Tool license expiry	Technology	Medium	Low	Low	Avoid (renew early)	Tech Lead	Closed

Risk-Probability Matrix



Quantitative Risk analysis

- Numerically analyzes the effect of identified risks
- Uses **quantitative techniques** (Monte Carlo simulation, decision trees, sensitivity analysis)
- Helps determine:
  - Probability of meeting project objectives
  - Range of possible outcomes (cost, schedule)
  - Risk exposure in monetary/time terms

Plan Risk Responses

- Develops strategies to **reduce threats** and **enhance opportunities**
- Common strategies:
  - **Avoid** – Eliminate risk cause
  - **Mitigate** – Reduce probability/impact
  - **Transfer** – Shift risk to third party (e.g., insurance, contracts)
  - **Accept** – Tolerate risk (with/without contingency plan)
- Updates **risk register** with response actions

- **Control Risks**
- Ongoing process of **monitoring risks** throughout the project
- Ensures **response plans are implemented**
- Tracks **residual risks & new risks**
- Uses **risk audits, variance analysis, technical performance measures**
- Updates risk register and project documents accordingly

Risk Mitigation, Monitoring, and Management

Summary

- Risk Management includes :-
  -Risk mitigation (i.e., avoidance)
  - Risk monitoring
  - Risk management and contingency planning
- Risk mitigation (avoidance) is the primary strategy and is achieved through a plan

Summary (continued)

- During risk monitoring, the project manager monitors factors that may provide an indication of whether a risk is becoming more or less likely.
- Risk management and contingency planning assume that mitigation efforts have failed and that the risk has become a reality.
- RMMM steps incur additional project cost
  - Large projects may have identified 30 – 40 risks
- Risk is not limited to the software project itself
  - Risks can occur after the software has been delivered to the user

Software Project Management

Procurement & Configuration Management

Today's Agenda (A 3-Hour Journey)

Part 1: Procurement Management

The "what, why, and how" of acquiring external software and services.

- The Contract SOW
- Planning Contracts (FP, CR, T&M)
- Requesting & Selecting Sellers
- Administering & Closing

Part 2: Configuration Management

The formal process of controlling your project's internal assets.

- Software Configuration Items (SCIs)
- Retaining Versions (Version Control)
- Change Control & Management
- The Change Control Board (CCB)

Part 1: Project Procurement

Acquiring the software and services you need to succeed.

The Foundation: Contract SOW



Scope of Work

Detailed description of the work, objectives, and tasks. (e.g., "Develop a new user authentication module.")



Deliverables & Milestones

Specifies deadlines, what will be delivered (code, docs), and the acceptance criteria for each.



Requirements

Technical specs, performance standards, and constraints. (e.g., "Must handle 10k concurrent users.")

Planning Contracts: Risk & Cost



Fixed-Price (FP)

SOW is clear. Price is set. Risk is on the *seller*. (e.g., "Build our website for $10,000.")



Cost-Reimbursable (CR)

SOW is unclear. Buyer pays all costs + a fee. Risk is on the *buyer*. (e.g., "R&D for a new algorithm.")



Time & Materials (T&M)

Hybrid model. Good for staff augmentation or support. (e.g., "Hire a senior developer for $90/hour.")

The Procurement Management Process

1. Plan

Create SOW, define contract type, make procurement plan.

2. Request

Issue RFP (Request for Proposal) or RFQ (Request for Quotation).

3. Select

Evaluate proposals, negotiate, and award the contract.

4. Administer

Manage relationship, monitor performance, make payments.

5. Close

Verify work, finalize payments, formally close contract.

Selecting Sellers: Source Selection



Vendor B is cheapest but scores poorly on technical skill and team experience. Vendor A is the best overall fit based on our weighted criteria.

Contract Admin & Closeout

Administering the Contract

This is *not* "set it and forget it." This is an active management role.

- Monitor seller performance against the SOW.
- Manage contract changes.
- Approve deliverables and make payments.
- Resolve disputes early.

Closing the Contract

A formal hand-off that prevents future problems.

- **Formal Acceptance:** Get written sign-off that the work is complete.
- **Final Payments:** Settle all invoices.
- **Lessons Learned:** Update records.
- **Legal Closure:** Formally notify the seller.

Part 2: Software Configuration Management (SCM)

Controlling your project's DNA—the code, documents, and assets.

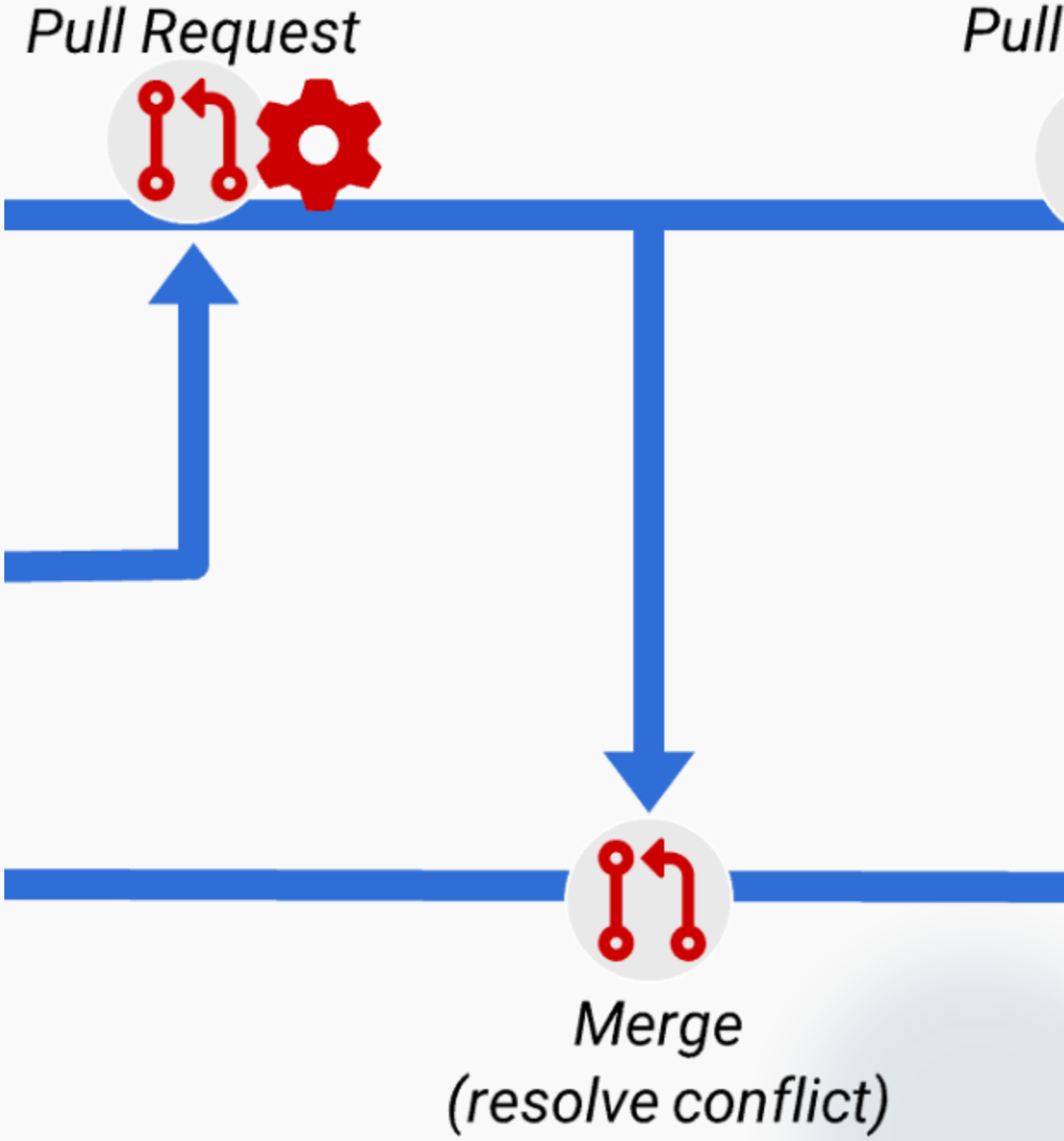
What is SCM?

SCM is a set of practices to manage, organize, and control changes to software project assets.

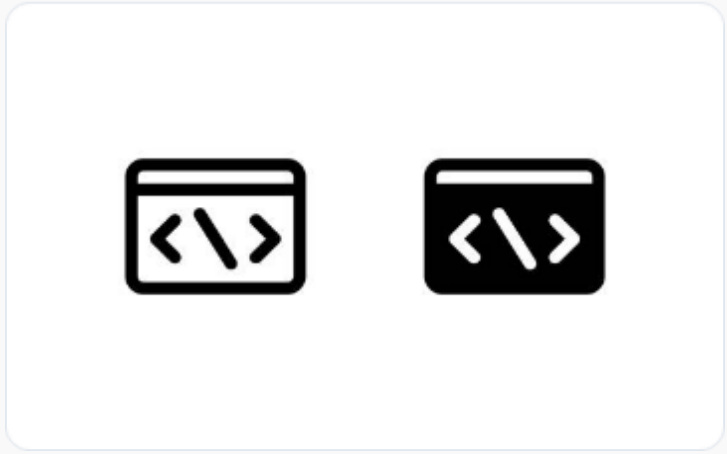
It is the **single source of truth**.

- It answers:
  - What is our "current" build?
  - Who changed this file, and why?
  - What's the difference between v1.1 and v1.2?

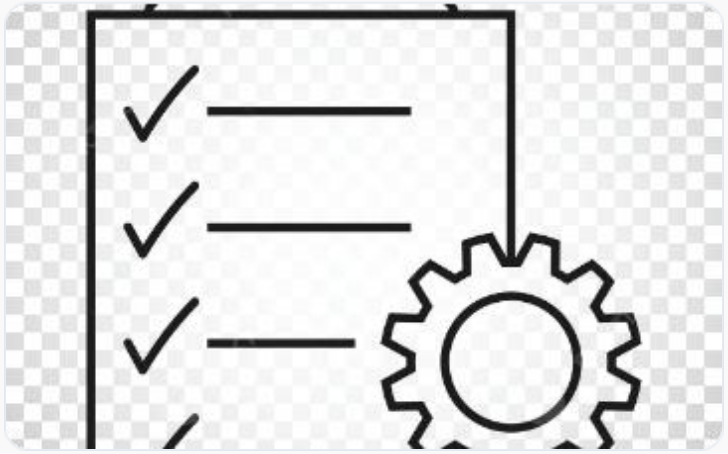
Without SCM, you have chaos.



Identifying SCIs (Software Configuration Items)



Source Code Files
(.java, .py, .js)



Project Documents
(Requirements, SOW, Design Docs)



Test Artifacts
(Test Plans, Test Cases)

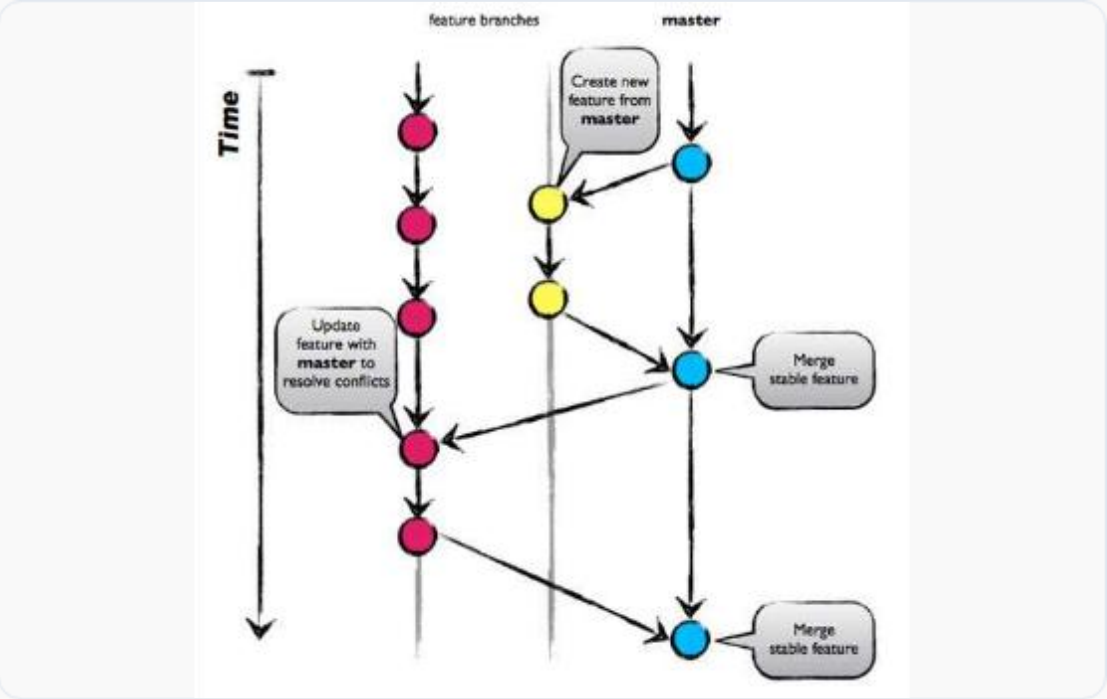
Retaining Versions (Version Control)

This is the *how* of SCM. We use Version Control Systems (VCS).

- **Centralized (e.g., SVN):** One central server. (Old-school)
- **Distributed (e.g., Git):** Everyone has a full copy. (Modern standard)

Key Concepts:

- commit: Save a snapshot of your changes.
- branch: Create a separate line of development (e.g., feature-login).
- merge: Combine your branch back into the main code.



The Need for Change Control

> The only constant in software is change… but uncontrolled change is chaos.
>
> — (Common Software Engineering Adage)

The Change Management Process

1. Submit

Stakeholder submits a formal Change Request (CR).

2. Analyze

Project team analyzes the CR for impact (cost, time, scope).

3. Approve/Reject

The Change Control Board (CCB) makes a decision.

4. Implement

If approved, the change is implemented, tested, and merged.

The Change Control Board (CCB)

- **Who is it?** The "gatekeepers" of the project baseline. Typically includes the Project Manager, Tech Lead, a Key Stakeholder (Product Owner), and the QA Lead.
- **What do they do?** They review all Change Requests (CRs), analyze their impact on scope, schedule, and cost, and then formally approve or reject them.
- **Why do they exist?** To protect the project from uncontrolled "scope creep" and ensure the stability and integrity of the project's baseline.

Key Takeaways

Procurement is Legal

Manage external resources with formal contracts and processes.

- The SOW is your legal foundation.
- Use the right contract type for the job.
- Administer and close contracts formally.

SCM is Control

Manage internal assets with formal version and change control.

- Establish a "single source of truth."
- Identify your SCIs (it's not just code).
- Use Git for versioning.
- Manage all changes through a formal CCB.

Questions?

Thank you for your attention.

Image Sources



https://www.red-gate.com/simple-talk/wp-content/uploads/2020/12/branching-diagram.png

Source: www.red-gate.com



https://media.istockphoto.com/id/1270030225/vector/command-line-icons-that-represent-programming-or-source-code.jpg?s=612x612&w=0&k=20&c=_xbm3RiDc1HEA4PYtVuSxtYchSTK9kwS35Bm4EKt9mU=

Source: www.istockphoto.com



https://png.pngtree.com/png-clipart/20230918/original/pngtree-flat-illustration-of-a-document-vector-icon-representing-project-management-depicted-png-image_12335203.png

Source: pngtree.com



https://cdn.produkto.io/designs/6686967515888d0842c45a23/Outline%20Circuit%20Icon.webp

Source: produkto.io



https://www.abtasty.com/wp-content/uploads/github-flow-branching-model.jpeg

Source: www.abtasty.com

Pareto Analysis

Introduction

- Pareto analysis is a formal technique useful where many possible courses of action are competing for attention
- Pareto analysis is a creative way of looking at causes of problems because it helps stimulate thinking and organize thoughts
- This technique helps to identify the top 20% of causes that needs to be addressed to resolve the 80% of the problems
- Works on principle of “the vital few and the trivial many”

History

- Named after Vilfredo Pareto, an Italian economist and sociologist who lived from 1848 to 1923

- In 1906, He created a mathematical formula to describe the unequal distribution of wealth in his country, observing that twenty percent of the people owned eighty percent of the wealth

Definition and Key concepts

- Pareto Analysis is a decision-making technique based on the Pareto Principle (80/20 rule), which states that 80% of problems are often caused by 20% of the causes. It helps prioritize efforts by identifying the most significant factors contributing to an issue

  Key Concepts

- Pareto Principle: Focus on the "vital few" rather than the "trivial many."
- Cause vs. Impact: Rank causes by their impact on the problem.
- Cumulative Contribution: Helps visualize how few causes lead to most of the effect

Pareto Analysis: Procedure

1. Gather data on the frequency of the causes

2. Rank the causes from the most to the least important, and calculate the cumulative percentage

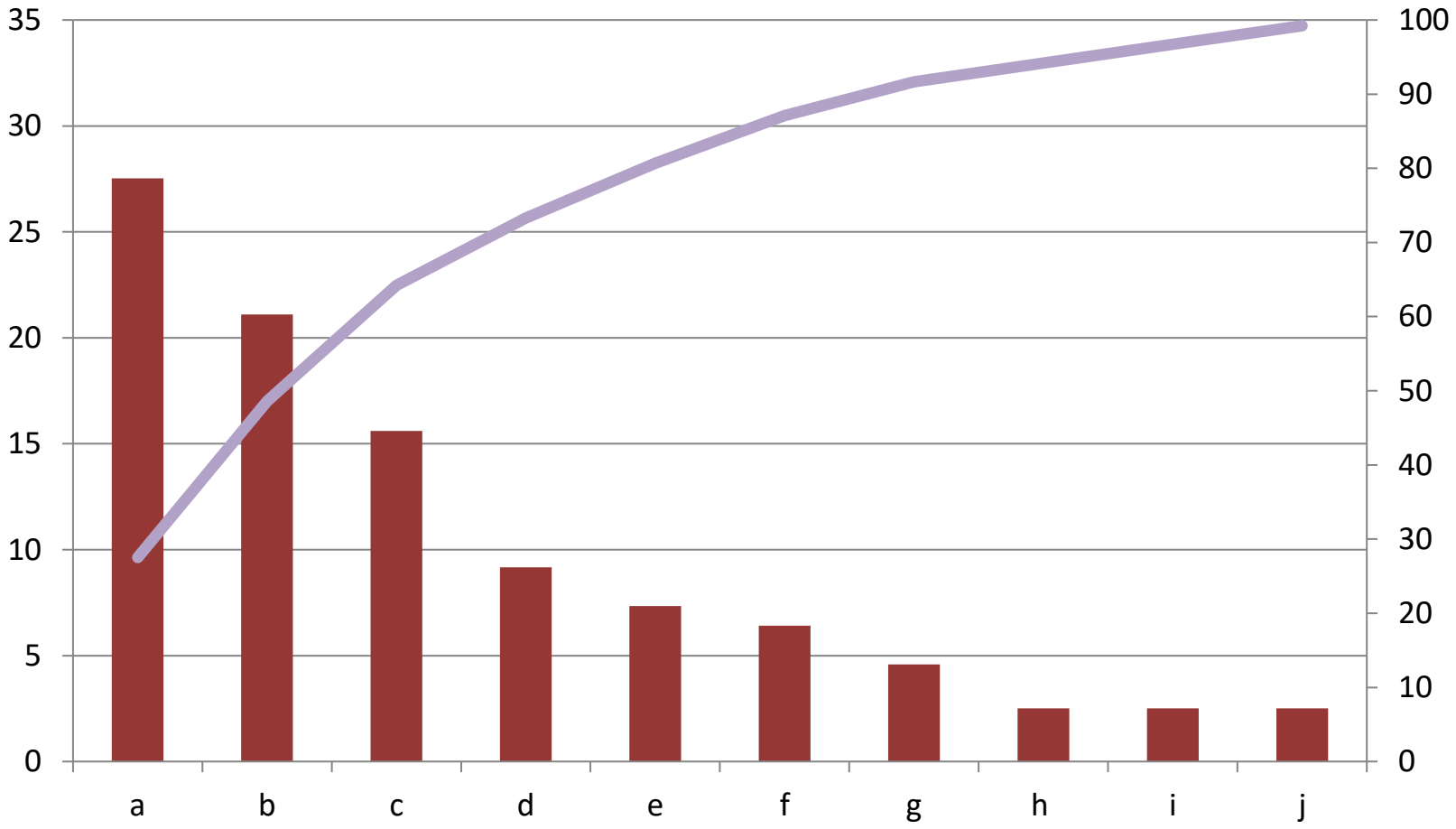
Causes	Percentage of Total	Cumulative Percent
A	20%	20%
B	18%	38%
C	15%	53%
D	11%	64%
E	10%	74%
F	6%	80%
G	6%	86%
H	6%	92%
I	5%	97%
J	3%	100%

Pareto Analysis: Procedure

3. Plot with the different causes on the x-axis, ordered from the most to least frequent, and the percentages on the y-axis, from 0 to 100%. Construct a bar graph based on the percentage of each cause

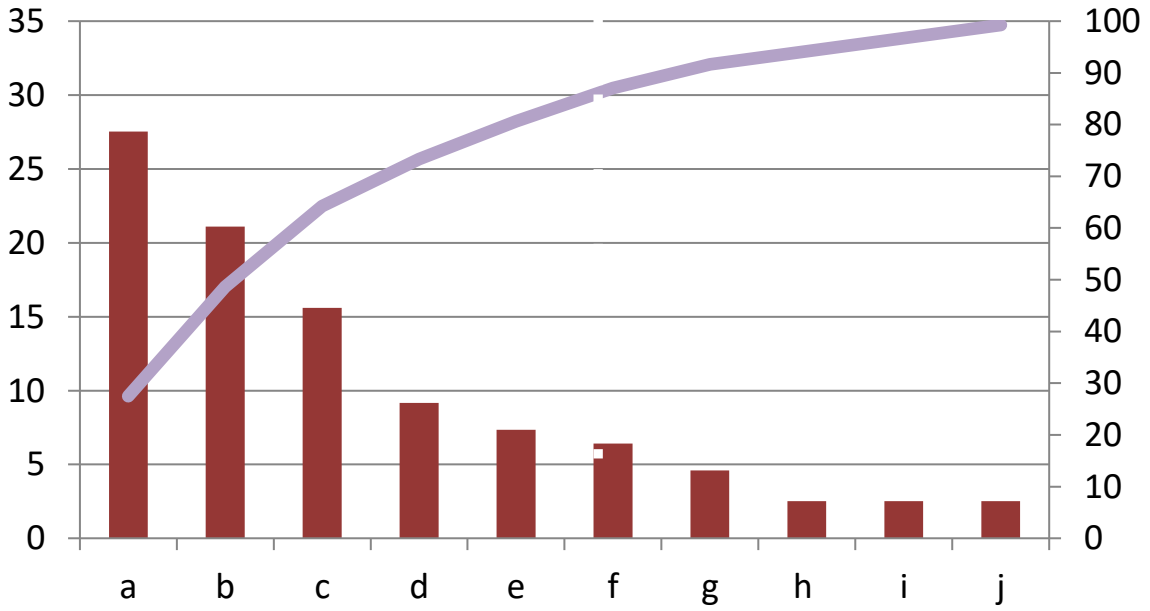
4. Construct a line graph of the cumulative percent

Pareto Analysis: Procedure



Pareto Analysis: Procedure

5. Draw a line from 80% on the y-axis (of cumulative percent) to the line graph that is parallel to the x-axis, and then drop the line down to the X axis. This line separates the important causes from the trivial ones



Pareto Analysis: Example

Mr. X is trying to determine why his clinic has been reaching such a small proportion of its eligible clients.

In carrying out the exercise, he forms a team. The team will first identify the primary reason why users are not using the services. They will then establish the causes for the problem and define a strategy and a plan of action for solving it.

The objective of the process will be to improve the quality of the services offered.

Pareto Analysis

Step 1. Frequency analysis

Possible Causes of Long Waiting Time	Percent of total
(A) Policies require excess information on users	1
(B) Policies require complicated procedures	1
(C) Too much paperwork	2
(D) Not enough funding	2
(E) Inadequate schedules	13
(F) Inadequate policies	2
(G) Clinic personnel have too many chores at home	2
(H) Clinic personnel have other jobs	2

Pareto Analysis

(H) Clinic personnel lack punctuality	6
(I) Clinic personnel have insufficient training	2
(J) Clinic personnel aren't motivated	1
(K) Clinic personnel are careless	1
(L) Clinic personnel don't follow the schedule	16
(M) Users forget ID cards	1
(N) Users don't keep appointments	2
(O) Delay in handing over lab results to doctors	14
(P) Outdated methods	12
(Q)Lack of automation	9
(R) Procedures take too long	11

Pareto Analysis

Step 2.Ranking causes

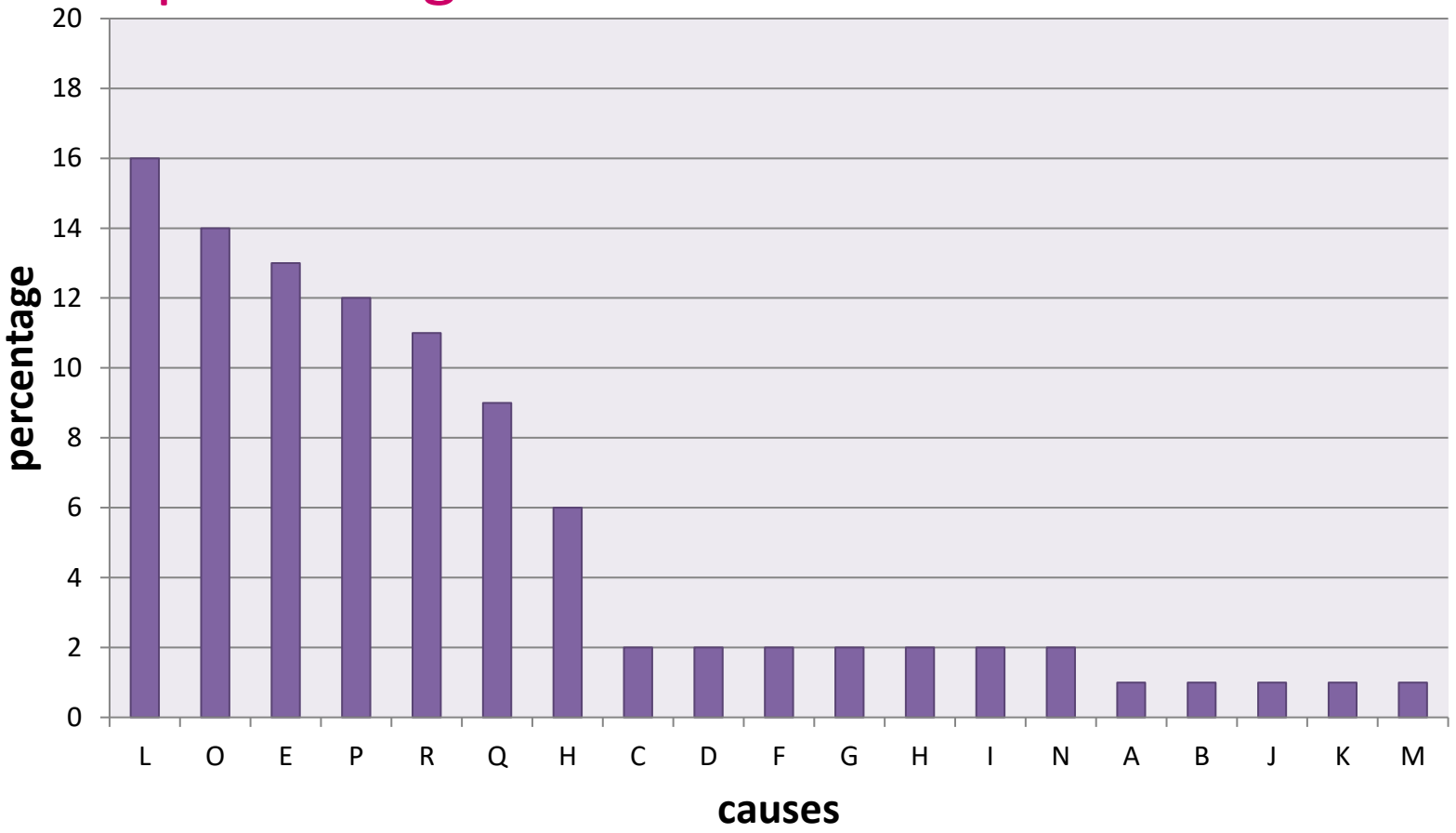
Cause	Percentage	Cumulative %
(L) Clinic personnel don't follow the schedule	16	16
(O) Delay in handing over lab results to doctors	14	30
(E) Inadequate schedules	13	43
(P) Outdated methods	12	55
(R) Procedures take too long	11	66
(Q) Lack of automation	9	75
(H) Clinic personnel lack punctuality	6	81
(C) Too much paperwork	2	83

Pareto Analysis

(D) Not enough funding	2	85
(F) Inadequate policies	2	87
(G) Clinic personnel have too many chores at home	2	89
(H) Clinic personnel have other jobs	2	91
(I) Clinic personnel have insufficient training	2	93
(N) Users don't keep appointments	2	95
(A) Policies require excess information on users	1	96
(B) Policies require complicated procedures	1	97
(J) Clinic personnel aren't motivated	1	98
(K) Clinic personnel are careless	1	99
(M) Users forget ID cards	1	100

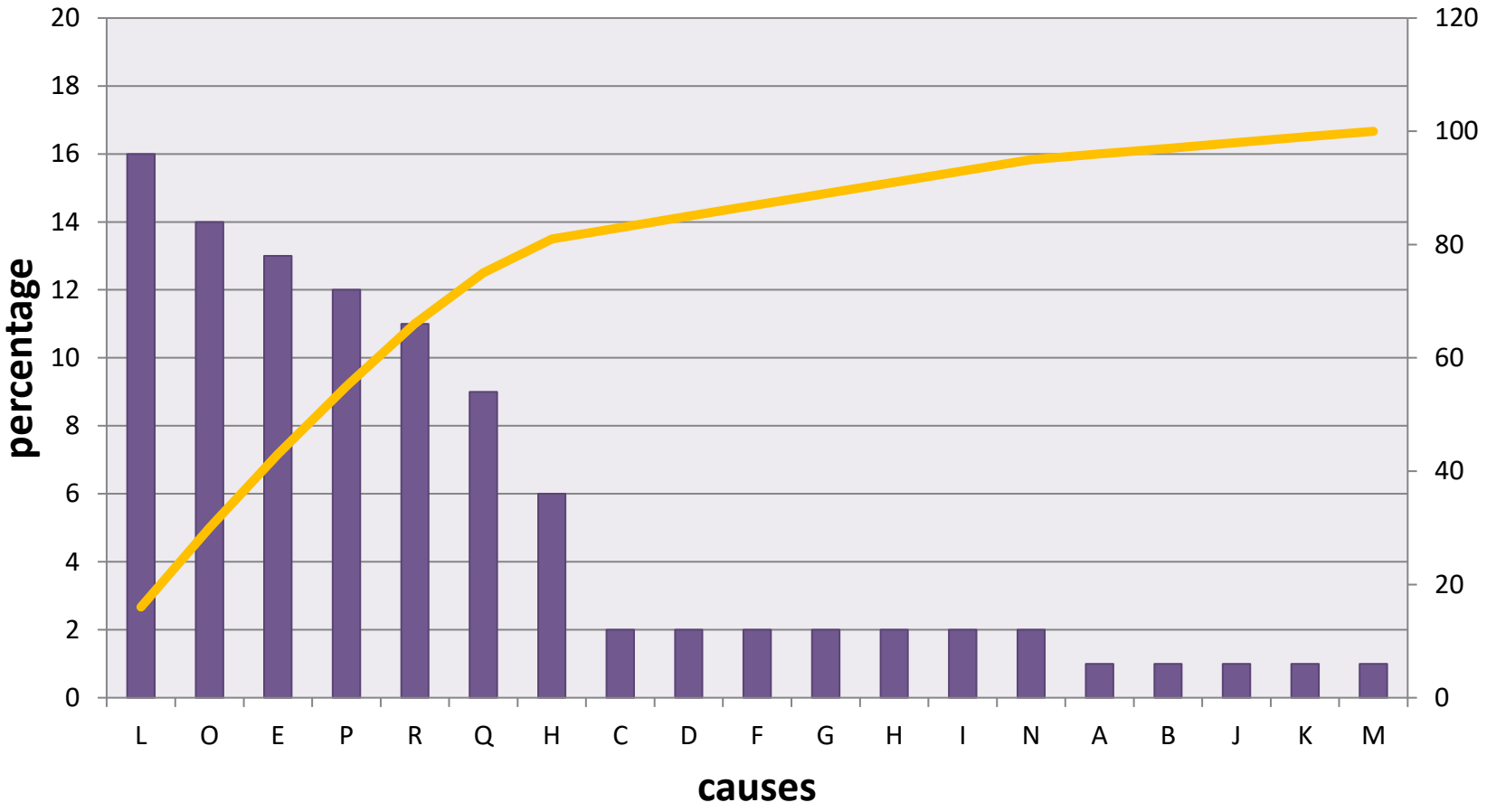
Pareto Analysis

Step 3. Construct a bar graph based on the percentage of each cause



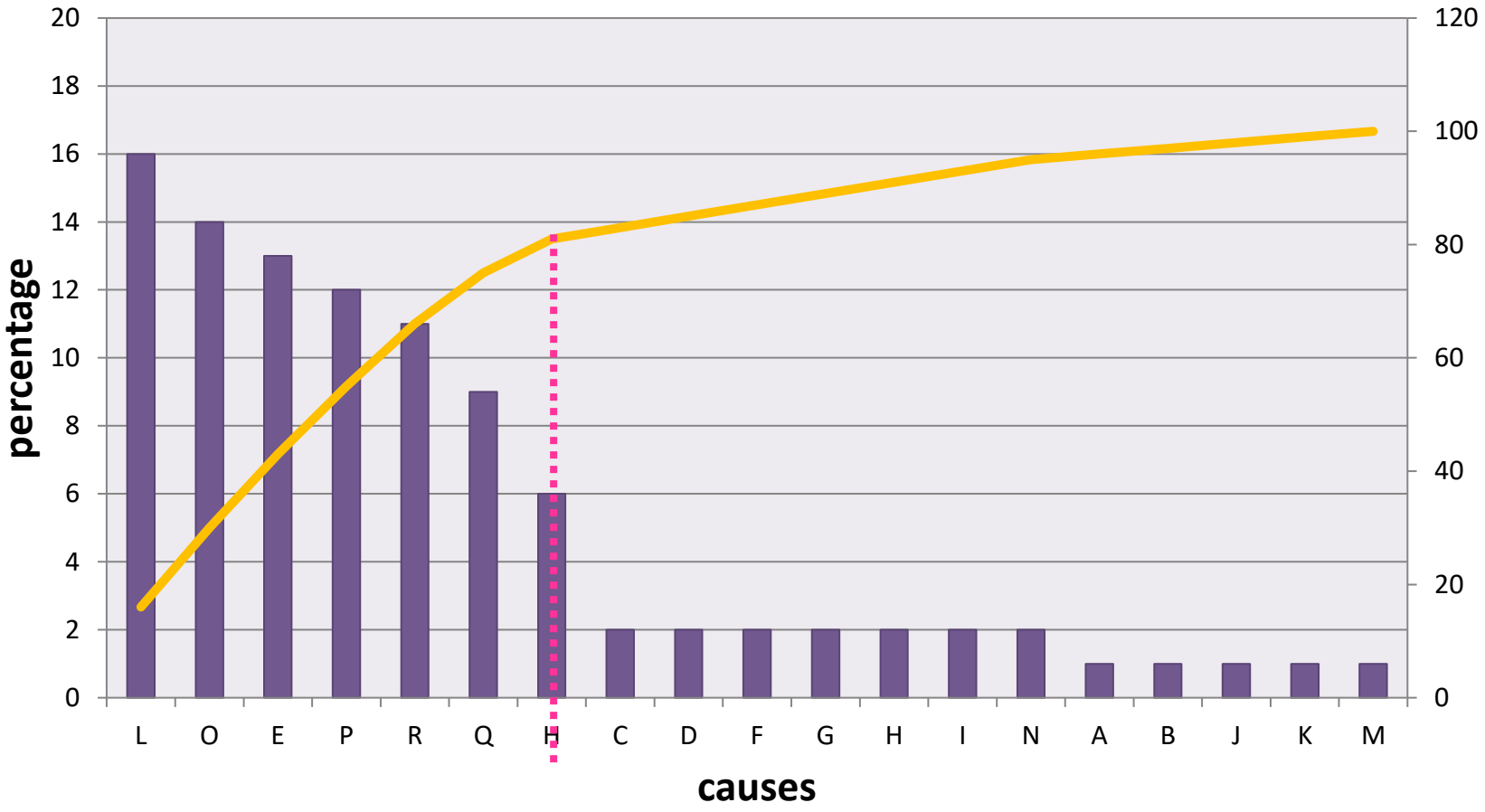
Pareto Analysis

Step 4. Construct a line graph based on cumulative %



Pareto Analysis

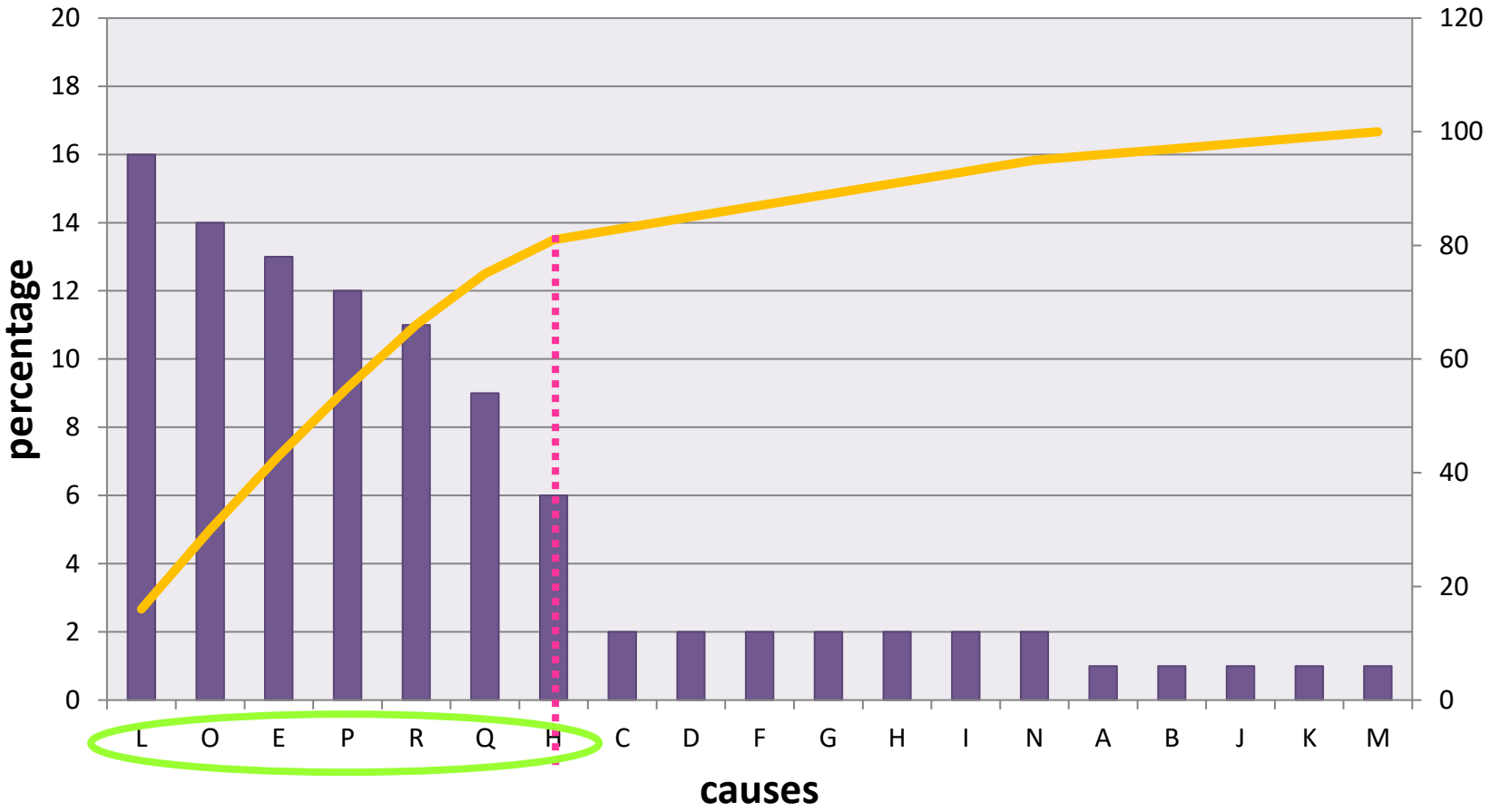
Step 5. Pareto Diagram



Pareto Analysis

Approximately 7 factors are responsible for 80% of the waiting time problem. The other 12 factors are responsible for only 20% of the said problem.

Interpretation of results



Benefits

- Solves efficiently a problem by the identification and the ranking, according to their importance, of the main causes of the faults
- Shows where to focus efforts
- Allows better use of limited resources
- Enhances problem-solving skills
- Improves decision making

Software Quality Assurance & Control

Software Project Management

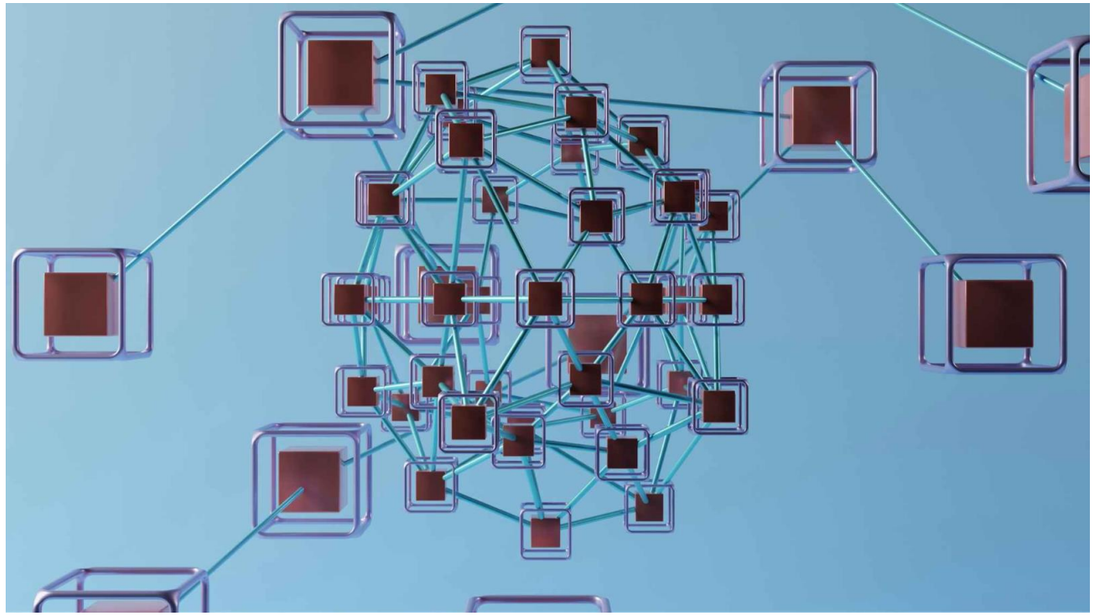


What is Software Quality?

Beyond "Bug-Free"

Software quality encompasses much more than the absence of defects. It's a comprehensive measure of how well software meets both specified requirements and user expectations.

Quality software delivers value consistently while maintaining performance, security, and user satisfaction across all interactions.



Core Quality Attributes

Reliability

Consistent performance under specified conditions without failure

Efficiency

Optimal use of system resources and fast response times

Usability

Intuitive interface and positive user experience

Maintainability

Easy to modify, debug, and enhance over time

Security

Protection against unauthorized access and data breaches

Why Quality Matters

User Satisfaction

Happy users drive product success. Quality software creates positive experiences that build loyalty and drive adoption rates.

Cost Reduction

Fewer bugs mean less rework and reduced maintenance costs. Investing in quality upfront saves significant resources long-term.

Reputation

Quality builds trust and credibility in the market. A strong reputation for reliability becomes a competitive advantage.

Risk Mitigation

Prevents critical system failures that could result in data loss, security breaches, or business disruption.



Why Quality Matters in Software Development

The Cyberpunk 2077 Disaster

One of gaming's most anticipated releases became a cautionary tale. Despite years of development and massive hype, the game launched with:

- *Game-breaking bugs and glitches*
- *Poor performance on consoles*
- *Massive reputation damage*
- *Millions in financial losses*

*This wasn't just a technical failure—it was a **quality failure** that could have been prevented.*



What Does "Working Software" Actually Mean?

Does it compile without errors?

Basic requirement, but far from sufficient for quality software

Does it pass your tests?

Good start, but what about edge cases and real-world scenarios?

Can it handle 1,000 simultaneous users?

Performance under load reveals hidden quality issues

What about unexpected user input?

Users always find ways to break software you never imagined

Quality Assurance
The Proactive Approach

The Chef Analogy

QA isn't about tasting the final dish. It's about ensuring you have:

- *A proven recipe*
- *Fresh ingredients*
- *Clean cooking tools*
- *Trained kitchen staff*

All before you start cooking!

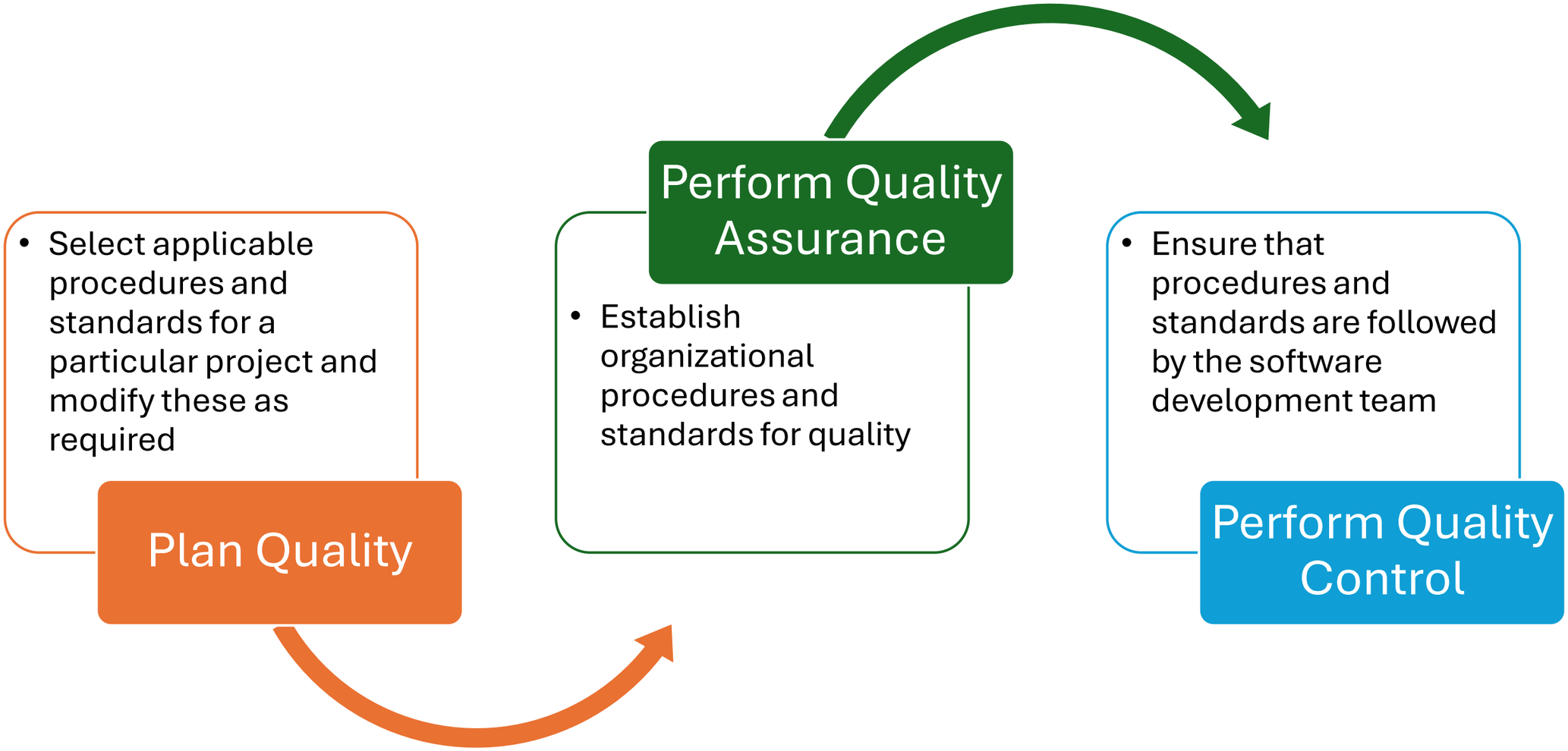
Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality.

The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI.



Quality Assurance ensures development processes are adequate to produce high-quality products.

How Do We Manage Quality?



Key QA Activities
Building the Foundation

Process Definition

- *Coding standards & conventions*
- *Documentation requirements*
- *Review procedures*

Tool Selection

- *Version control systems (Git)*
- *Bug tracking software (Jira)*
- *Testing frameworks*

Team Training

- *Process education*
- *Tool proficiency*
- *Best practices sharing*

Process Audits

- *Compliance checking*
- *Gap identification*
- *Continuous improvement*

The ultimate goal: Defect Prevention

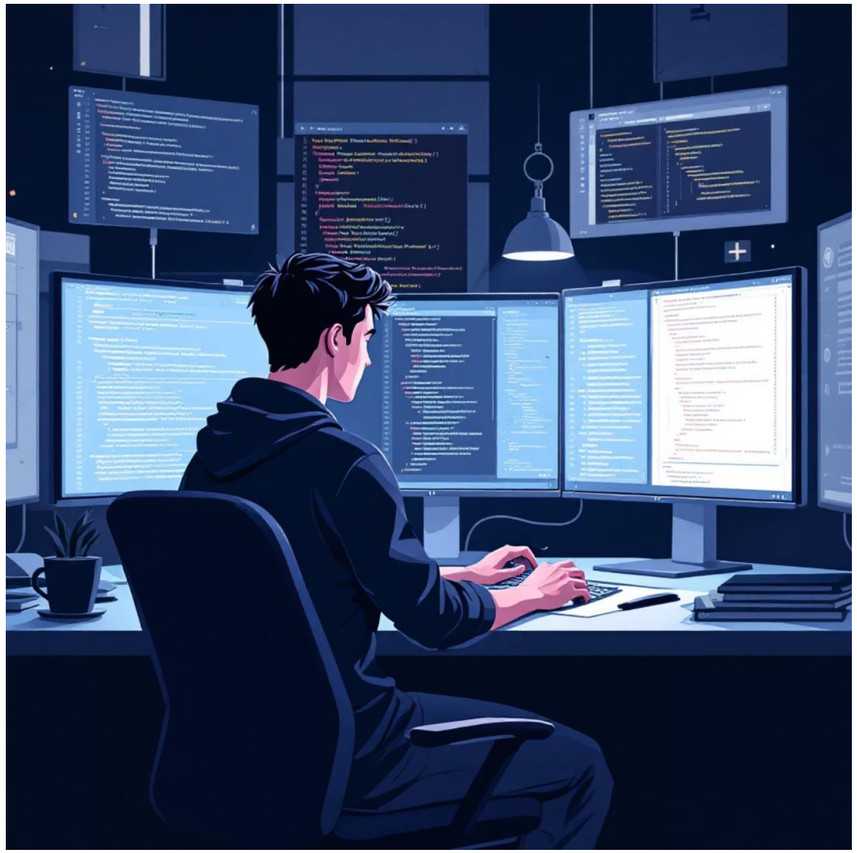


Software Quality Assurance

Requirements definition	Software design	Coding
Source code control	Code reviews	Change management
Configuration management	Testing	Release management
	Product integration	

Quality Control

The Reactive Approach



Now We Taste the Dish

While QA focuses on the process, QC is all about the product.

> ⚠ ***Definition:*** *Quality Control evaluates developed products to identify defects before customer release.*

QC asks the critical questions:

- *Does it meet requirements?*
- *Are there bugs or defects?*
- *Is performance acceptable?*
- *Will users be satisfied?*

QC Activities

Finding What's Wrong

1

Code Reviews & Walkthroughs

Peer examination of code logic, standards compliance, and design feasibility

2

Unit Testing

Testing individual functions and components in isolation

3

Integration Testing

Verifying how multiple components work together

4

System Testing

End-to-end testing of the complete application

5

User Acceptance Testing

Final validation by clients and end-users

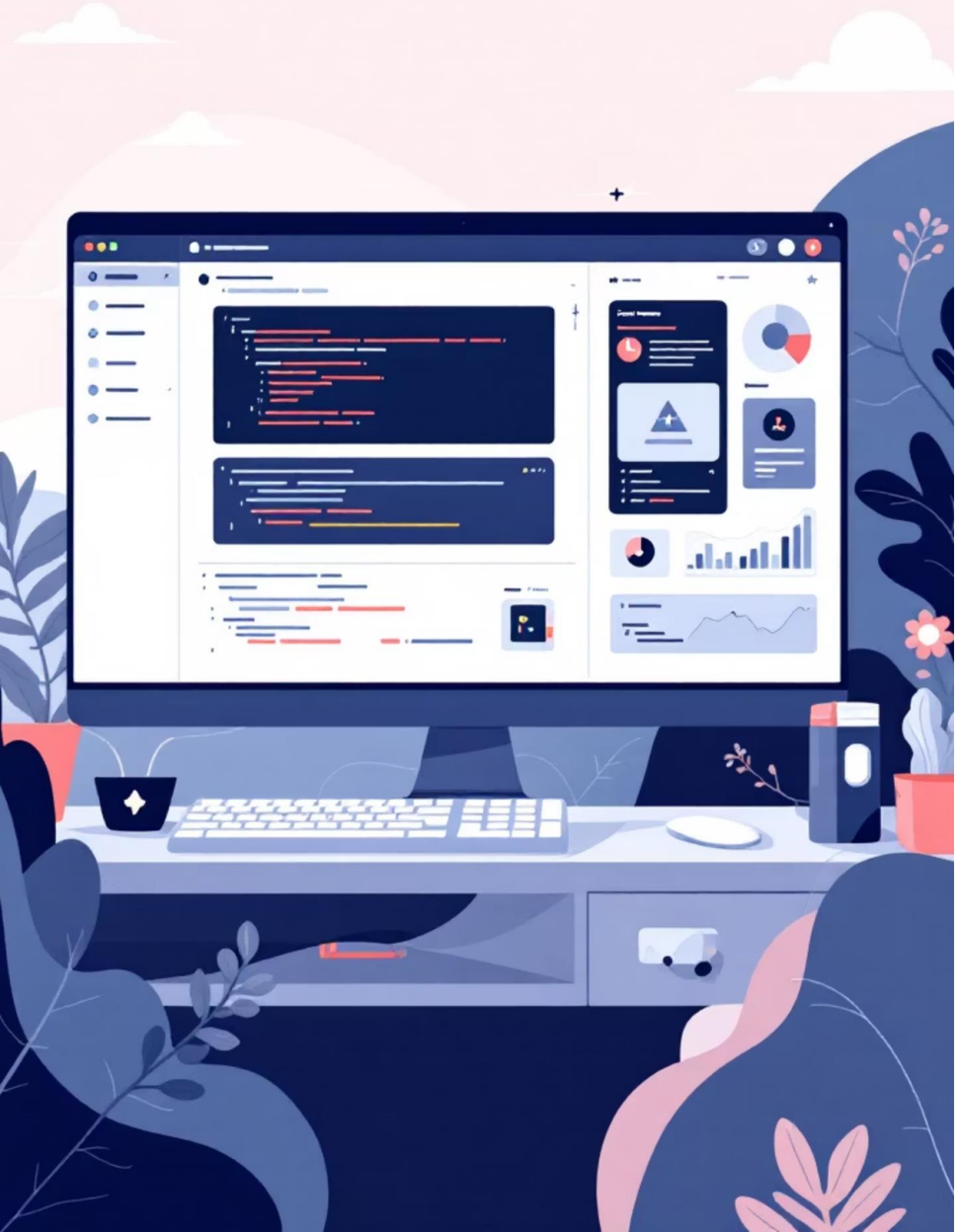


QA vs QC

Two Sides of Quality

Basis	*Quality Assurance (QA)*	*Quality Control (QC)*
Focus	*Process-oriented*	*Product-oriented*
Goal	*Prevent defects*	*Detect defects*
Nature	*Proactive*	*Reactive*
Scope	*Entire development lifecycle*	*Testing phase focus*
Responsibility	*Everyone on the team*	*Primarily testing team*
Analogy	*Checking recipe & ingredients*	*Tasting the final dish*

*Remember: These approaches are **complementary**, not competing!*



Your Quality Engineering Toolkit

Classic Quality Tools

- **Flowcharts**

  Visualize process flow and identify problem areas

- **Pareto Charts**

  80% of bugs come from 20% of modules—focus your efforts

- **Control Charts**

  Track quality metrics over time to spot process breakdowns

- **Cause-Effect Diagrams**

  Systematic root cause analysis for recurring issues

Modern Software Tools

- **Static Analysis**

  SonarQube, ESLint—find bugs without running code

- **Test Automation**

  Selenium, Jest—systematic testing at scale

- **Performance Testing**

  JMeter, Gatling—simulate real-world load

- **Bug Tracking**

  Jira, GitHub Issues—manage defect lifecycle

*Quality is **everyone's responsibility**—and what separates professionals from amateurs.*

Key Takeaways

Quality is Not an Afterthought: It must be planned and managed throughout the entire project lifecycle. It is everyone's responsibility.

QA Prevents, QC Detects: Quality Assurance is the proactive, process-focused effort to prevent defects. Quality Control is the reactive, product-focused effort to find defects.

You Need Both: A mature development team relies on a strong QA culture to build things correctly and rigorous QC practices to verify the result.

Use Tools Systematically: Don't just test randomly. Use established techniques and modern tools to make your quality control efforts efficient and effective.

Expert Judgement and Decision Trees



Judgement

“Expert judgment is the process of relying on specialized knowledge or experience from individuals or groups to make informed project decisions, especially under uncertainty.”

Expert Judgement Process

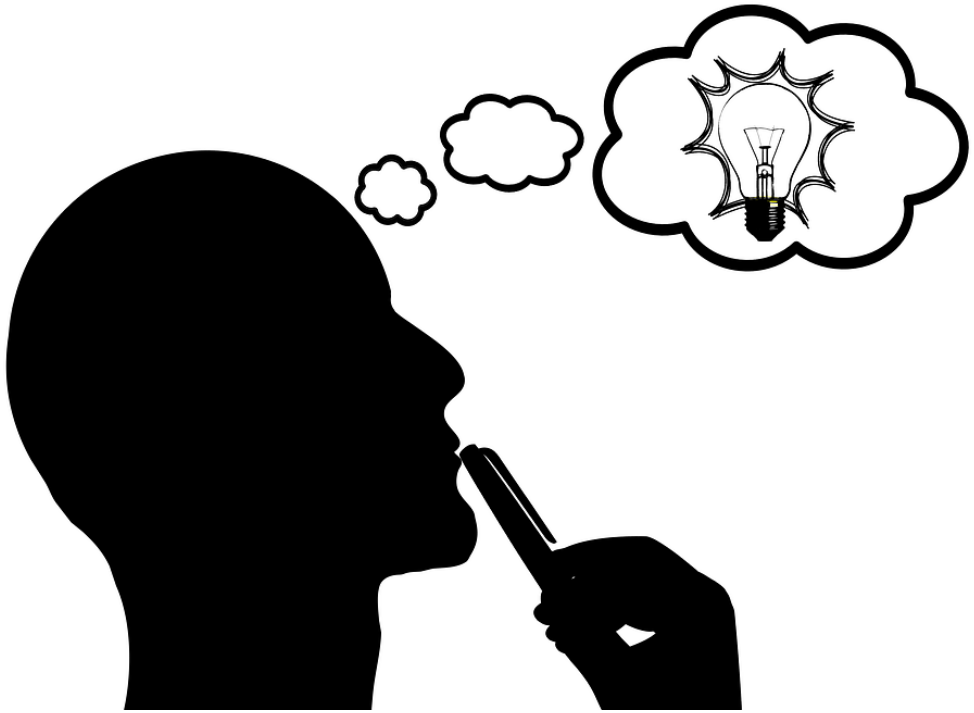
- Frame the problem.
- Plan the elicitation of expert opinions.
- Select the appropriate experts.
- Brief/train the experts so they can contribute effectively.
- Elicit their opinions/judgments.
- Analyze and combine the information to create your expert judgment.
- Document and communicate the results.

Expert Judgement in SPM

- Effort Estimation (e.g., Wideband Delphi, Planning Poker)
- Cost Estimation
- Risk Identification and Assessment
- Scope Estimation
- Technology Selection
- Vendor Evaluation
- Schedule Feasibility

Who Are the Experts?

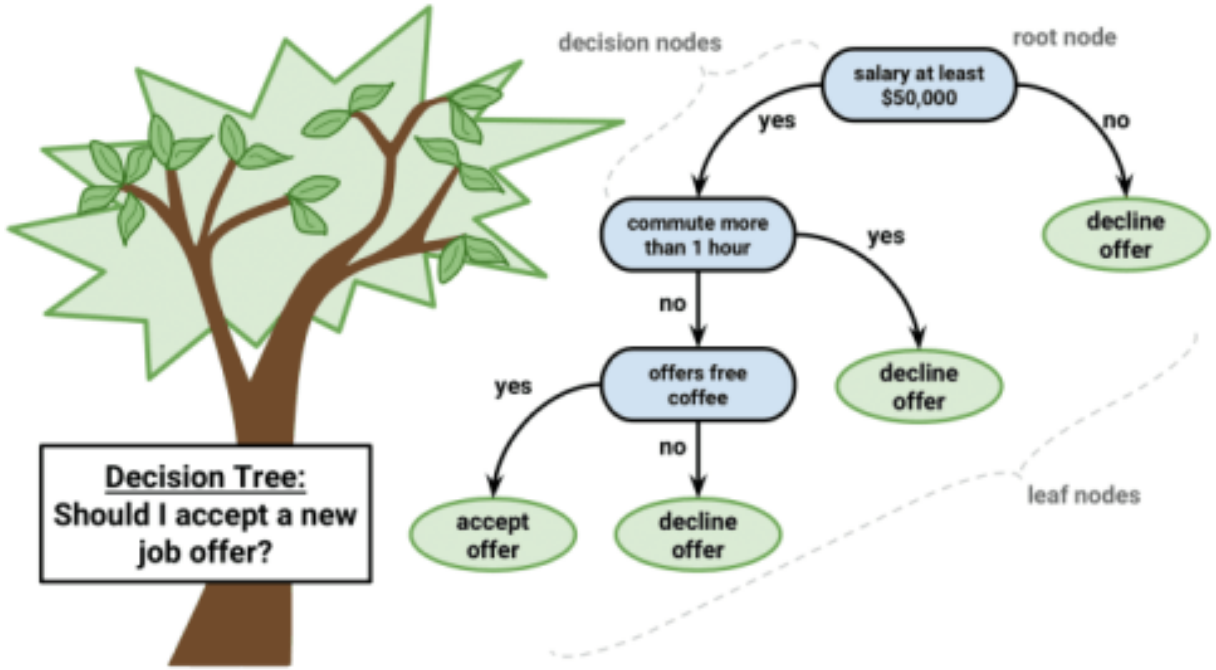
- Senior Developers
- Architects
- Project Managers
- QA Leads
- External Consultants
- Domain Experts



Techniques

- Interviews
- Brainstorming sessions
- Delphi Technique
- Workshops
- Team meetings

Decision Trees and Expected Monetary Value



“A decision tree is a graphical and mathematical tool used to analyze decisions with uncertain outcomes. It visually maps out decision options, chance events, their probabilities, and expected monetary values (EMV). Identifies the best path when multiple decisions or outcomes are possible.”

Components of a Decision Tree

- **Root Node:** The starting point of the decision tree, it represents the initial decision or question to be considered.

- **Branches:** Represents the different options or alternatives available at each decision or change point. Each branch can lead to another decision node, a chance node or an outcome(leaf node).

- **Internal Nodes:** Also called decision nodes, they represent the points in the tree where decisions are made.

- **Leaf Nodes:** These are the final nodes of the decision tree and represent the outcomes of the decisions made earlier in the tree.

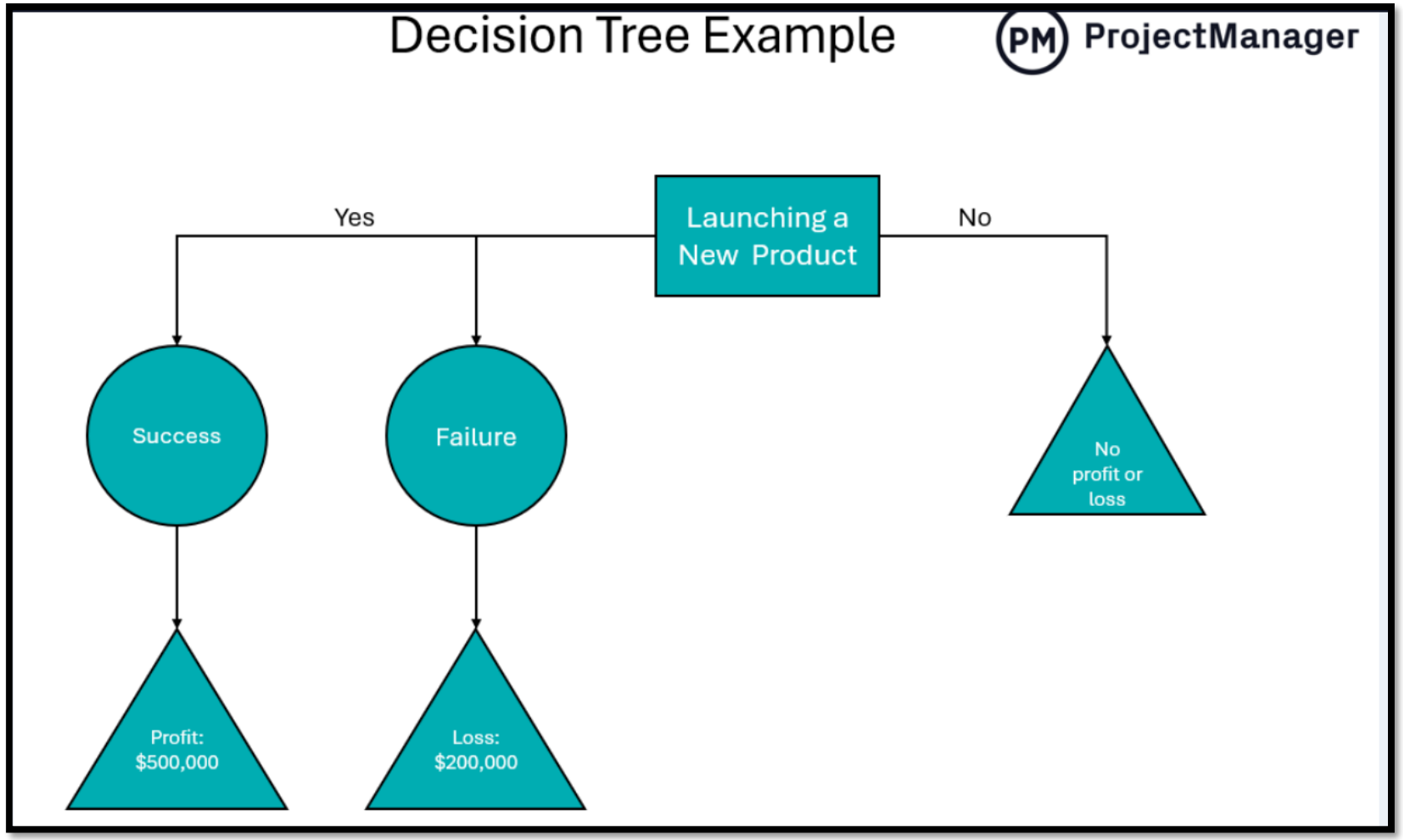
Decision Trees in SPM : Scenarios

- Decision tree analysis is especially useful for project managers who must make multiple decisions throughout the life cycle of a project. A few examples include when allocating resources, budgeting and scheduling projects and dealing with the risk inherent in any project.
- Risk Analysis
- Root Cause Analysis

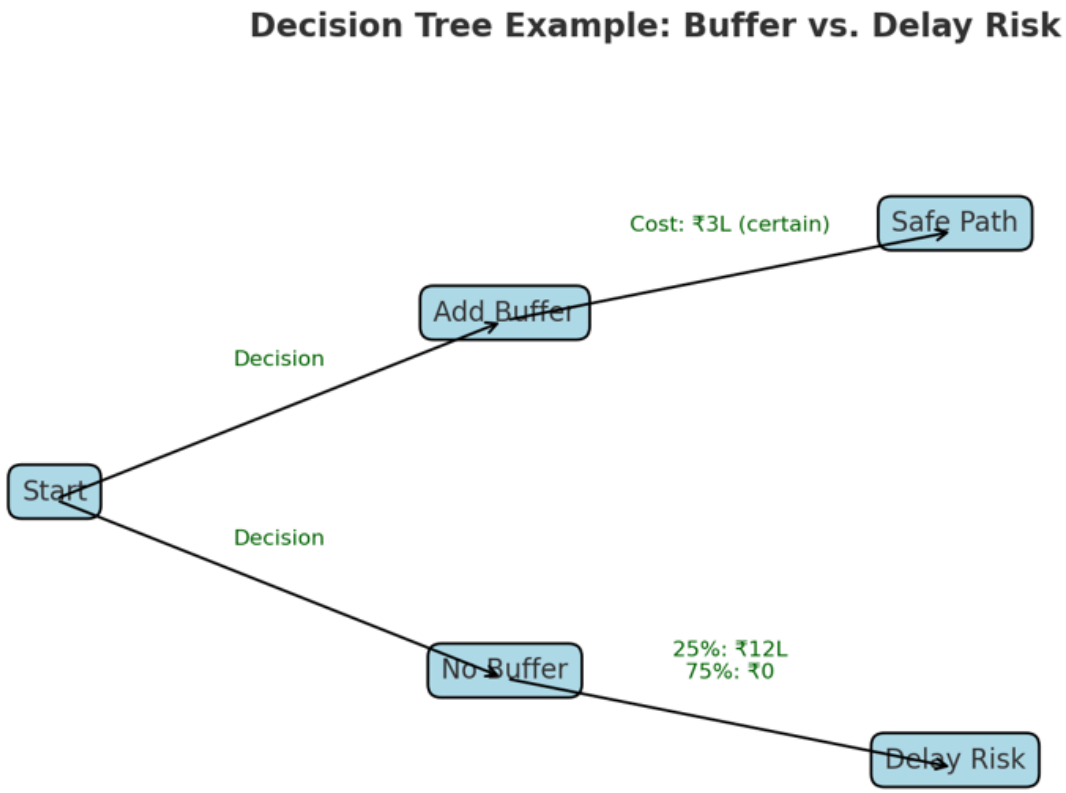
Decision Tree Analysis Symbols

- **Decision node:** A square or rectangle represents a decision point where a choice must be made.
- **Chance node:** A circle represents a point where an uncertain event occurs, with outcomes that are based on chance. The possible outcomes are typically represented by branches with probabilities.
- **Outcome/end node, also called leaf or terminal node:** A triangle, though sometimes represented as a small circle, which is the outcome or result of a decision path.
- **Branches:** While defined above, these are represented by arrows that show the flow of decisions and outcomes in a decision tree chart.
- **Annotations/labels:** Text descriptions along branches or nodes to clarify decisions, probabilities or values.

Decision Tree : Case Study1- Launch of a product



Case Study 2


Decision Tree Example: Buffer vs. Delay Risk
Start
Decision
Add Buffer
Cost: ₹3L (certain)
Safe Path
Decision
No Buffer
25%: ₹12L
75%: ₹0
Delay Risk

Expected Monetary Value

- EMV (Expected Monetary Value) is a quantitative risk assessment technique used by a project manager to calculate the average financial outcome of a risk event by multiplying its probability by its monetary impact.
- To calculate the EMV, you need to consider two factors:
- Probability
- Monetary Value
- **EMV Calculation:** EMV = ∑ (Probability × Outcome Value)

- Software Process Improvement
  - CMM
  - Standards
  - Six Sigma

Software Process Improvement

- *Software process improvement* is the art and science of changing an organization's software process to build better software.
- Software process improvement always involves looking at the big picture.
- This means writing down the entire software process as a whole and making sure that it is followed on each project.
  - It involves looking at entire projects and identifying areas that can be improved.

Frameworks and Methodologies

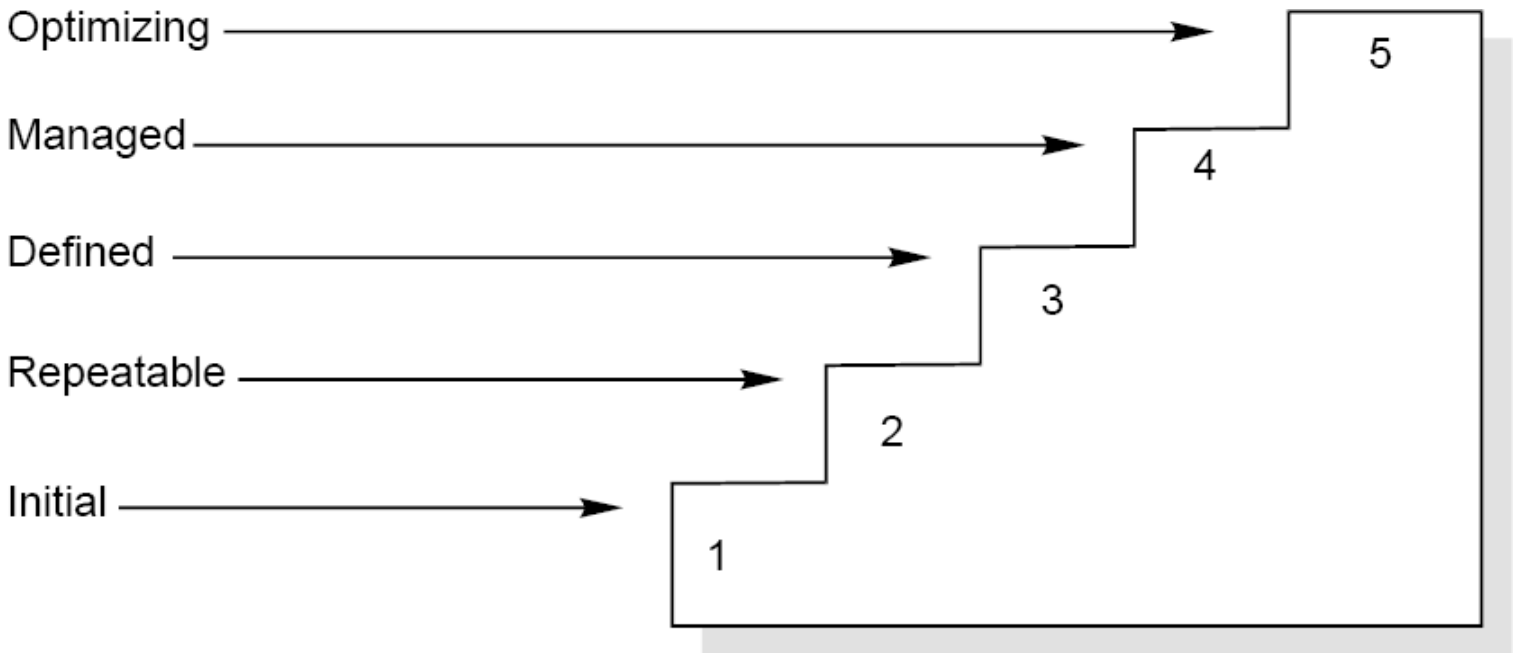
- Models and certifications that help assess the state of the organization's process and serve as a *framework* for improving that process.

- Capability Maturity Model (CMM), Quality management standards (ISO 9000) and Six Sigma are frameworks.

Capability Maturity Model (CMM)

- Capability Maturity Model (CMM)

It is a strategy for improving the software process, irrespective of the actual life cycle model used.



Maturity levels of CMM

Capability Maturity Model

Maturity Levels:

- ✓ Initial (Maturity Level 1)
- ✓ Repeatable (Maturity Level 2)
- ✓ Defined (Maturity Level 3)
- ✓ Managed (Maturity Level 4)
- ✓ Optimizing (Maturity Level 5)

Capability Maturity Model

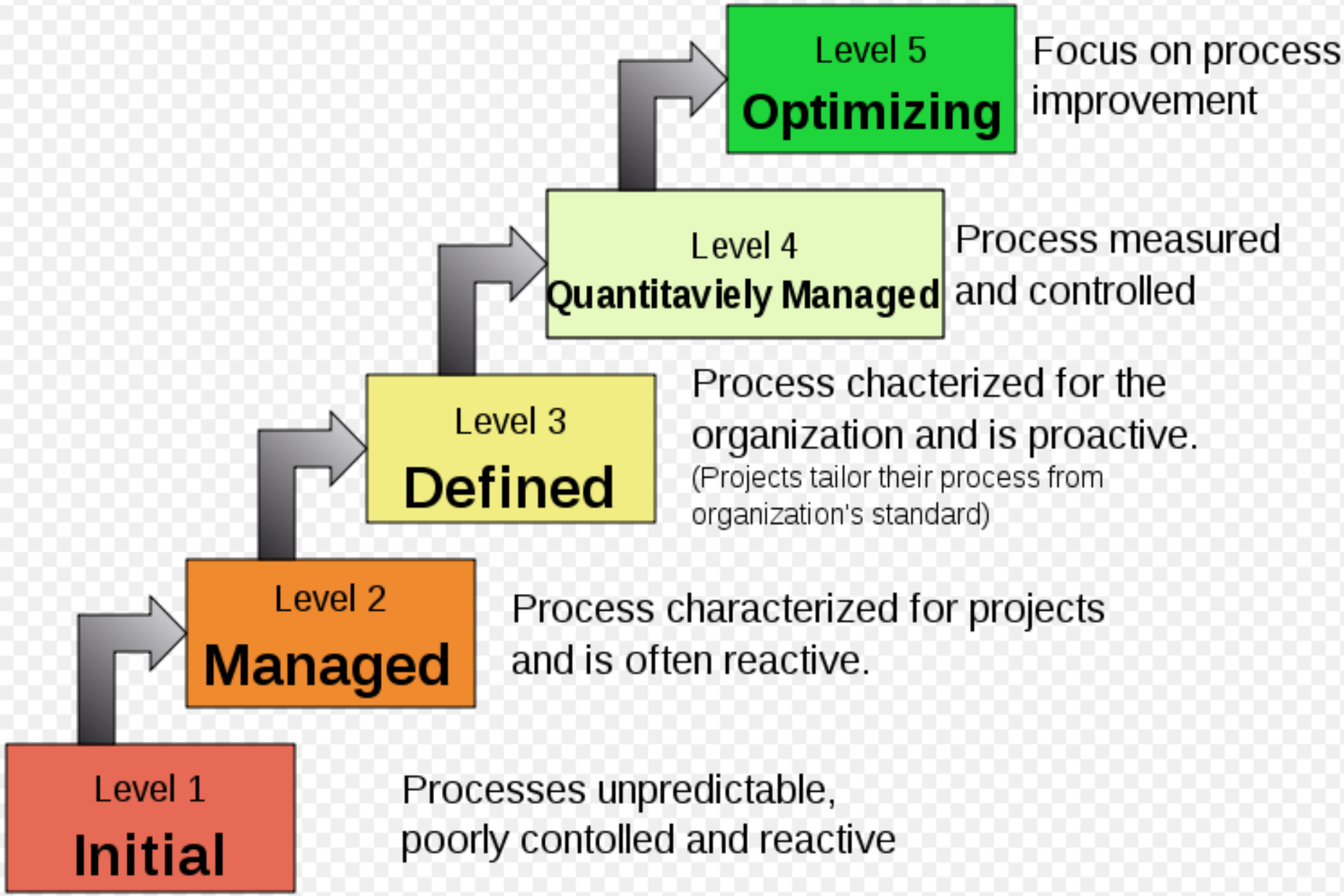
Maturity Level	Characterization
Initial	Adhoc Process
Repeatable	Basic Project Management
Defined	Process Definition
Managed	Process Measurement
Optimizing	Process Control

Fig.24: The five levels of CMM



SEI - CMMI Framework

Characteristics of the Maturity levels



CMM – Level 1 – Initial Level

The organization

- Does not have an environment for developing and maintaining software.

- At the time of crises, projects usually stop using all planned procedures and revert to coding and testing.

CMM – Level 2 – Repeatable level

Effective management process having established which can be

- Practiced
- Documented
- Enforced
- Trained
- Measured
- Improvised

CMM – Level 3 – Defined level

- Standard defined software engineering and management process for developing and maintaining software.

- These processes are put together to make a coherent whole.

CMM – Level 4 – Managed level

- The organizational measurement plan involves determining the productivity and quality for all important software process activities across all projects.

CMM – Level 5 – Optimizing level

Emphasis laid on

- Process improvement
- Tools to identify weaknesses existing in their processes
- Make timely corrections

Maturity Level 2 Process Areas

Maturity Level 2 - Repeatable
CMM
RM - Requirements Management
SPP-Software Project Planning
SPTO- Software Project Tracking and Oversight
SSM-Software Subcontract Management
SQA-Software Quality Assurance
SCM- Software Configuration Management

CMM-Key Process Areas

- Key Process Areas

The key process areas at level 2 focus on the software project's concerns related to establishing basic project management controls, as summarized below:

Requirements Management (RM)	Establish a common relationship between the customer requirements and the developers in order to understand the requirements of the project.
Software Project Planning (PP)	Establish reasonable plans for performing the software engineering and for managing the software project.
Software Project Tracking and Oversight (PT)	Establish adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.
Software Subcontract Management (SM)	Select qualified software subcontractors and manage them effectively.
Software Quality Assurance (QA)	Provide management with appropriate visibility into the process being used by the software project and of the products being built.
Software Configuration Management (CM)	Establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

Maturity Level 3 Process Areas

Maturity Level 3 - Defined
CMM
ISM - Integrated Software Mgmt
IC-Inter Group Coordination
PD-Organization Process Definition
PF -Organization Process Focus
TP-Training Program
SPE-Software Product Engineering
PR-Peer Reviews

CMM-Key Process Areas

The key process areas at level 3 address both project and organizational issues, as summarized below:

Organization Process Focus (PF)	Establish the organizational responsibility for software process activities that improve the organization's overall software process capability.
Organization Process Definition (PD)	Develop and maintain a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organization.
Training Program (TP)	Develop the skills and knowledge of individuals so that they can perform their roles effectively and efficiently.
Integrated Software Management (IM)	Integrate the software engineering and management activities into a coherent, defined software process that is tailored from the organization's standard software process and related process assets.

(Contd.)...

CMM-Key Process Areas

Software Product Engineering (PE)	Consistently perform a well-defined engineering process that integrates all the software engineering activities to produce correct, consistent software products effectively and efficiently.
Inter group Coordination (IC)	Establish a means for the software engineering group to participate actively with the other engineering groups so the project is better able to satisfy the customer's needs effectively and efficiently.
Peer Reviews (PR)	Remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of the defects that can be prevented.

Research and Development Processes

Quality Management System

- Hand Book
- Quality Manual
- Role Based Activity
- Knowledge Bank
- Processes
- Project Documents
- Best Practices

CMMI Implementation

- Project Organization Chart
- SEPG Members
- SQA Members



Quality Objective

To achieve CMMI Level 5 by 2015 and maintain it there after.

Maturity Level 4 & 5 Process Areas

Maturity Level 4 - Quantitatively Managed
QM-Quantitative Process Management
QM-Software Quality Management
Maturity Level 5 - Optimizing
DP-Defect prevetion
TCM -Technology Change Management; PCM-Process Change Management

CMM-Key Process Areas

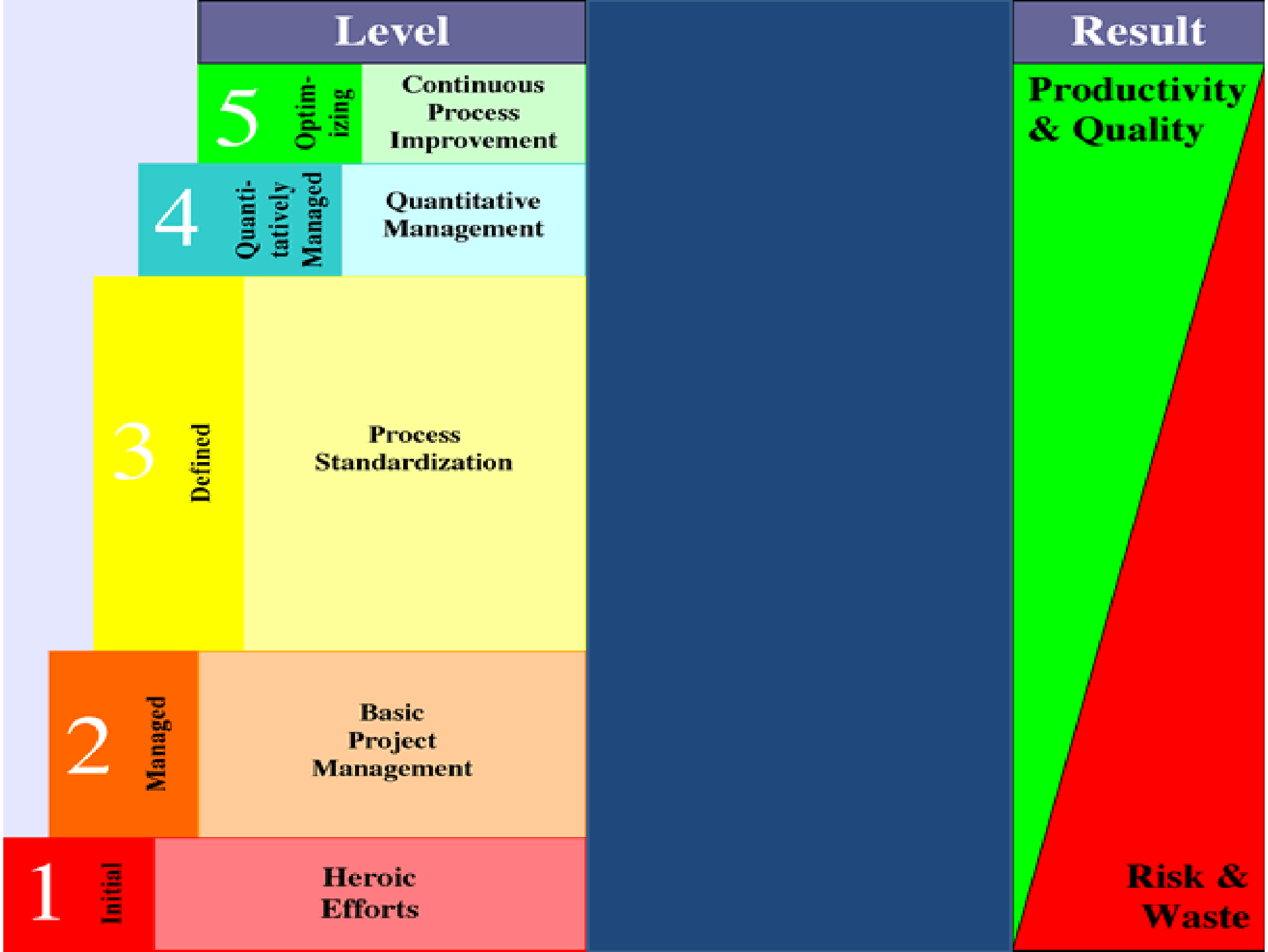
The key process areas at level 4 focus on establishing a quantitative understanding of both the software process and the software work products being built, as summarized below:

Quantitative Process Management (QP)	Control the process performance of the software project quantitatively.
Software Quality Management (QM)	Develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals.

CMM-Key Process Areas

The key process areas at level 5 cover the issues that both the organization and the projects must address to implement continuous and measurable software process improvement, as summarized below:

Defect Prevention (DP)	Identify the causes of defects and prevent them from recurring.
Technology Change Management (TM)	Identify beneficial new technologies (i.e., tools, methods, and processes) and transfer them into the organization in an orderly manner.
Process Change Management (PC)	Continually improve the software processes used in the organization with the intent of improving software quality, increasing productivity, and decreasing the cycle time for product development.


Level
Result
5
Optim-izing
Continuous Process Improvement
4
Quanti-tatively Managed
Quantitative Management
3
Defined
Process Standardization
2
Managed
Basic Project Management
1
Initial
Heroic Efforts
Productivity & Quality
Risk & Waste

Standards

- Standards are the key to effective quality management
- They may be international, national, organizational or project standards
- Product standards define characteristics that all components should exhibit e.g. a common programming style.
- Process standards define how the software process should be enacted.

ISO 9000

- International set ofstandards for quality management
- Applicable to a range of organisations from manufacturing to service industries
- ISO 9001 applicable to organisations which design, develop and maintain products
- ISO 9001 is a generic model of the quality process Must be instantiated for each organisation

Importance of standards

- Encapsulation of best practice- avoids repetition of past mistakes
- Framework for quality assurance process - it involves checking standard compliance
- Provide continuity - new staff can understand the organisation by understand the standards applied

Steps in implementation of ISO - 9000

- Group formation: formation of team, the group leader will be called Management Representative (MR). The team is called steering committee.

- Application: Application is made to one of the following bodies:
  - BVQI – Burea virita quality international at USA
  - DNV – Det Nrske Veritas at Germany
  - BSI – Britsh Standard Institue Quality Assurance at UK. etc.

Contrasting ISO 9001 & CMM

- **Contrasting ISO 9001 and the CMM**

There is a strong correlation between ISO 9001 and the CMM, although some issues in ISO 9001 are not covered in the CMM, and some issues in the CMM are not addressed in ISO 9001.

The biggest difference, however, between these two documents is the emphasis of the CMM on continuous process improvement.

The biggest similarity is that for both the CMM and ISO 9001, the bottom line is **"Say what you do; do what you say"**.

Mapping ISO 9001 & CMM

Management Responsibility

- ISO: Quality Policy be define, documented, understood, implemented and Maintained.
- CMM: Management responsibility for quality policy and verification activities primarily addresses in Software Quality Assurance.

Quality System

- ISO: Documents quality system, including procedure and instructions be established
- CMM: Quality primarily addressed in Software Quality Assurance distributed thru KPA's

Contract Review

- ISO:contracts be reviewed to determine whether the requirements are adequately defined, agree with bid and can be implemented
- CMM: Review customer requirement is spanned in Requirement mgnt.

Design Control

- ISO: Procedure to control& verify Design be established(planning, Inputs, outputs, design)
- CMM:Life cycle activities , design , code, test are described in Software Project Planning

Mapping ISO 9001 & CMM

- **Document Control**
  - ISO:Distribution & modification of documents be controlled
  - CMM:Document control are described in configuration mgmt
- **Purchasing**
  - ISO:purchased products conform their specified requirements(assessment of subcontractors, verification of purchased products)
  - CMM:Addressed in Software Sub contract Management
- **Purchaser -supplied product**
  - ISO: purchaser supplied material verified and maintained
  - CMM: In integrated software mgmt in using purchased software
- **Product identification & Traceability**
  - ISO: During all stages of production delivery & installation
  - CMM:covering Software Configuration Mgmt
- **Process Control**
  - ISO:Production process be defined & planned
  - CMM: Thru KPA' in various actvites

Mapping ISO 9001 & CMM

- **Inspection & Test Status**
  - ISO:Inspection and test be maintained for items as they progress through various processing steps
  - CMM: Testing practices in software product Engineering
- **Corrective Action**
  - ISO: causes of non conforming product be identified,products eliminated, procedures are changed from corrective action
  - CMM: Analysis, updates, patches
- **Training**
  - ISO:Training needs be identified and training provided
  - CMM:Training program, Orientation practices
- **Servicing**
  - ISO:servicing activities be performed as specified
  - CMM:Not a separate process
- **Statistical Techniques**
  - ISO:Appropriate and adequate statistical techniques are identified
  - CMM:Distributed throughtout the KPA's

Quality Certification

- ISO 9000
  - Set of international standards on quality management and quality assurance, critical to international business
- ISO 14000
  - A set of international standards for assessing a company’s environmental performance

ISO 9000 Standards

Requirements

- System requirements
- Management
- Resource
- Realization
- Remedial

ISO 9000 Quality Management Principles

- Customer focus
- Leadership
- People involvement
- Process approach
- A systems approach to management
- Continual improvement
- Factual approach to decision making
- Mutually beneficial supplier relationships

ISO 14000

- ISO 14000 - A set of international standards for assessing a company's environmental performance
- Standards in three major areas
  - Management systems
  - Operations
  - Environmental systems

ISO 14000

- Management systems
  - Systems development and integration of environmental responsibilities into business planning
- Operations
  - Consumption of natural resources and energy
- Environmental systems
  - Measuring, assessing and managing emissions, effluents, and other waste

Six Sigma

- ***Six Sigma*** is a business process for improving quality, reducing costs, and increasing customer satisfaction.

- The Greek letter sigma refers to standard deviation—Six Sigma means “six standard deviations from the mean.”

- Statistically-Having no more than 3.4 defects per million
- Conceptually-Program designed to reduce defects
  - Requires the use of certain tools and techniques

Six Sigma Programs

- Six Sigma programs
  - Improve quality
  - Save time
  - Cut costs

- Employed in
  - Design
  - Production
  - Service
  - Inventory management
  - Delivery

Six Sigma Management

- Providing strong leadership
- Defining performance metrics
- Selecting projects likely to succeed
- Selecting and training appropriate people

Six Sigma Technical

- Improving process performance
- Reducing variation
- Utilizing statistical models
- Designing a structured improvement strategy

Six Sigma Team

- Top management
- Program champions
- Master “black belts”
- “Black belts”
- “Green belts”

Six Sigma Process

- Define
- Measure
- Analyze
- Improve
- Control

DMAIC

Software Configuration Management (SCM) ⚙️

A core discipline in Software Project Management

Introduction to Software Configuration Management (SCM)

Definition: *A discipline for systematically controlling the changes that occur in a software system throughout its lifecycle.*

Traceability

Know who, what, when, and why a change was made.



Consistency

Ensure all team members are using the correct version.



Integrity

Maintain the correct components of the system.



Accountability

Establish a clear audit trail for all changes.

Retaining Versions and Baselines

Version:

A distinct, identified change to a software artifact (e.g., v1.1).

Configuration Item (CI):

Any artifact placed under SCM control (e.g., source code, test script).

Baseline:

A formally accepted snapshot of a software configuration at a specific point in time. Once baselined, it only changes via formal process.

Example Baseline Progression

Functional Baseline
(After Requirements)

Allocated Baseline
(After Design)

Product Baseline
(Final Product)



Software Configuration Elements (SCI's)

SCI: *Any artifact required for software development and maintenance that is placed under configuration control.*



Code

Source code files, build scripts, libraries.



Documentation

Requirements, design specs, test plans, user manuals.



Data

Databases, setup files, configuration files.



Tools

Compilers, libraries, COTS software.

The Four Pillars: SCM Core Activities



Configuration Identification

Naming, storing, and identifying SCI's and their versions.



Configuration Control

Managing changes to baselined SCI's (the Change Control process).



Configuration Status Accounting

Recording and reporting the status of all SCI's (who, what, when).



Configuration Audits

Verifying that a released configuration matches the documented baseline.

Change Control and Management

- **Change Request (CR):** The formal document or ticket requesting a change. This is the starting point for all modifications to a baseline.
- **Change Control Board (CCB):** A dedicated, authorized group (managers, engineers, clients) that reviews, evaluates, and approves/rejects all CRs.
- **Impact Analysis:** The process of assessing the technical, cost, and schedule consequences of a proposed change. This is the primary job of the CCB.
- **The Process:** A formal workflow for requesting, evaluating, approving, implementing, and verifying changes to baselined SCI's.

The SCM Toolkit: Automating the Process



Version Control (VCS)

Essential for code and documents. Manages versions and history.
(e.g., Git, Subversion)



Change/Issue Tracking

Manages the Change Request (CR) workflow and SCM reporting.
(e.g., Jira, Azure DevOps)

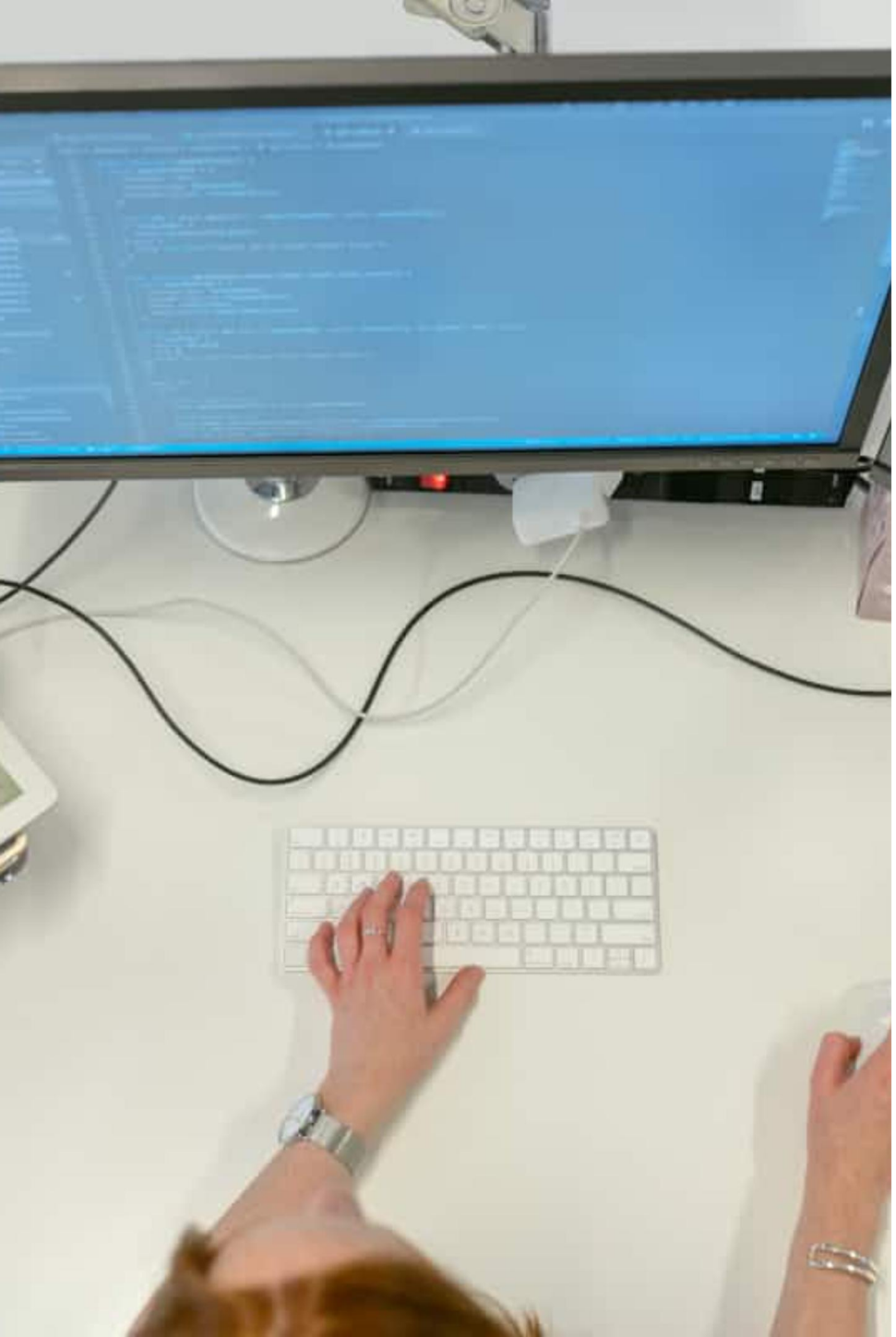


Build Automation

Ensures consistent, reproducible builds from baselined code.
(e.g., Jenkins, GitLab CI/CD)

Summary and Key Takeaways

- **Procurement:** Requires a robust plan, a clear SOW, and formal processes for selection, administration, and closure.
- **Contracts:** Selection of the right type (FP, T&M, CR) is critical and depends on scope certainty.
- **SCM:** Essential for managing product integrity and traceability. It saves you from the chaos of uncontrolled changes.
- **Baseline:** A stable, controlled product configuration that requires formal CCB approval to modify.
- **Remember:** If you can't control it, you can't manage it.



Software Project Management

Procurement & Configuration Management

Managing external vendor relationships and internal project assets through formal processes and proven methodologies.

Today's Agenda

01

Project Procurement Management

The complete lifecycle of acquiring software and services—from build vs. buy decisions through contract closeout.

02

Statement of Work & Contract Planning

Creating the legal foundation with detailed SOWs and selecting appropriate contract types based on risk profiles.

03

Seller Selection & Contract Administration

Objective evaluation methods using weighted scoring and best practices for monitoring vendor performance.

04

Software Configuration Management

Formal processes for controlling project assets including version control, change management, and the role of Change Control Boards.

Part 1: Project Procurement Management

Why Procurement Matters

- No software project exists in isolation.
- You'll acquire cloud infrastructure from AWS, license databases from Oracle, or outsource specialized modules to contracting firms.
- Procurement management is the formal framework for these critical external relationships.

Done Right

Saves time, reduces costs, accelerates delivery

Done Wrong

Project failure, legal disputes, massive overruns



The Contract Statement of Work (SOW)

Your Legal Foundation

> "If it's not in the SOW, it doesn't exist."

1

Scope of Work

Detailed description of what the seller will do. Define boundaries clearly—what's included and explicitly what's excluded.

2

Deliverables & Milestones

Specific outputs with concrete deadlines. Each deliverable must have measurable completion criteria.

3

Requirements & Acceptance Criteria

Technical specifications and the objective standards you'll use to verify and accept their work.

Critical Rule: Never sign a contract with a vague SOW. Ambiguity breeds disputes, scope creep, and project failure.

Contract Types: A Risk Management Decision

Fixed-Price

Best for: Clear, well-defined scope

Example: "Build our website for $10,000"

Risk: Seller bears efficiency risk

Cost-Reimbursable

Best for: R&D, uncertain requirements

Example: Pay all costs + 15% fee

Risk: Buyer bears all cost overruns

Time & Materials

Best for: Staff augmentation, flexible needs

Example: "$90/hour for developers"

Risk: Shared—requires active oversight



The Procurement Management Process

1. Plan

Create SOW, define contract type, make procurement plan.

2. Request

Issue RFP (Request for Proposal) or RFQ (Request for Quotation).

3. Select

Evaluate proposals, negotiate, and award the contract.

4. Administer

Manage relationship, monitor performance, make payments.

5. Close

Verify work, finalize payments, formally close contract.

This is a formal, end-to-end lifecycle. Each phase has specific deliverables and requires diligent project management to ensure success.

Selecting Sellers: Never Choose on Price Alone

Weighted Scoring Matrix Example

Criteria	Weight	Vendor A	Vendor B	Vendor C	Max
Technical Skill	35%	32	18	28	35
Team Experience	25%	23	10	20	25
SOW Understanding	20%	18	14	16	20
Cost	20%	14	20	16	20
Total Score	**100%**	**87**	**62**	**80**	**100**

Winner: Vendor A — Despite being more expensive, Vendor A delivers the strongest technical capability and experience. Vendor B's low cost masks significant risk in execution quality.



Contract Administration & Closeout

Active Administration Phase

- **Monitor Performance:** Track deliverables against SOW milestones and quality standards
- **Manage Changes:** Use formal change control for scope modifications
- **Resolve Disputes:** Address issues promptly through escalation procedures
- **Document Everything:** Maintain detailed records of communications and decisions

Formal Closeout Process

- **Written Acceptance:** Verify work meets SOW acceptance criteria
- **Financial Settlement:** Reconcile all invoices and process final payment
- **Legal Protection:** Obtain sign-off to prevent future claims
- **Lessons Learned:** Document vendor performance for future projects



Part 2: Software Configuration Management

Controlling Your Project's DNA

From External to Internal

Procurement manages relationships with outside vendors. Configuration Management controls the assets you create internally—source code, documentation, test plans, and build configurations.

SCM Scope

Version control, change tracking, release management, and audit trails

SCM is the formal process that ensures your team can reliably reproduce any version of your software, track every change, and maintain control as complexity grows.

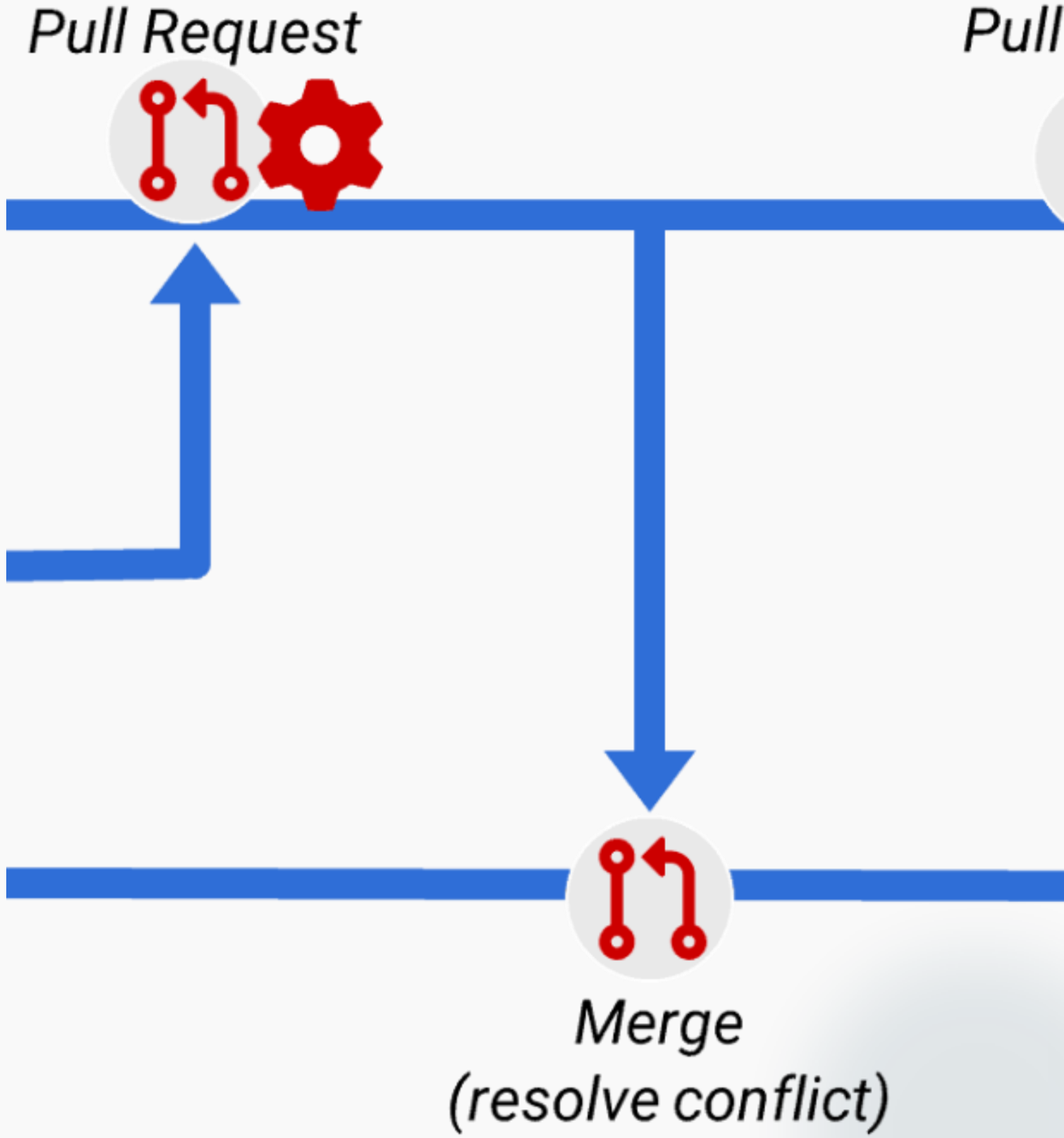
What is SCM?

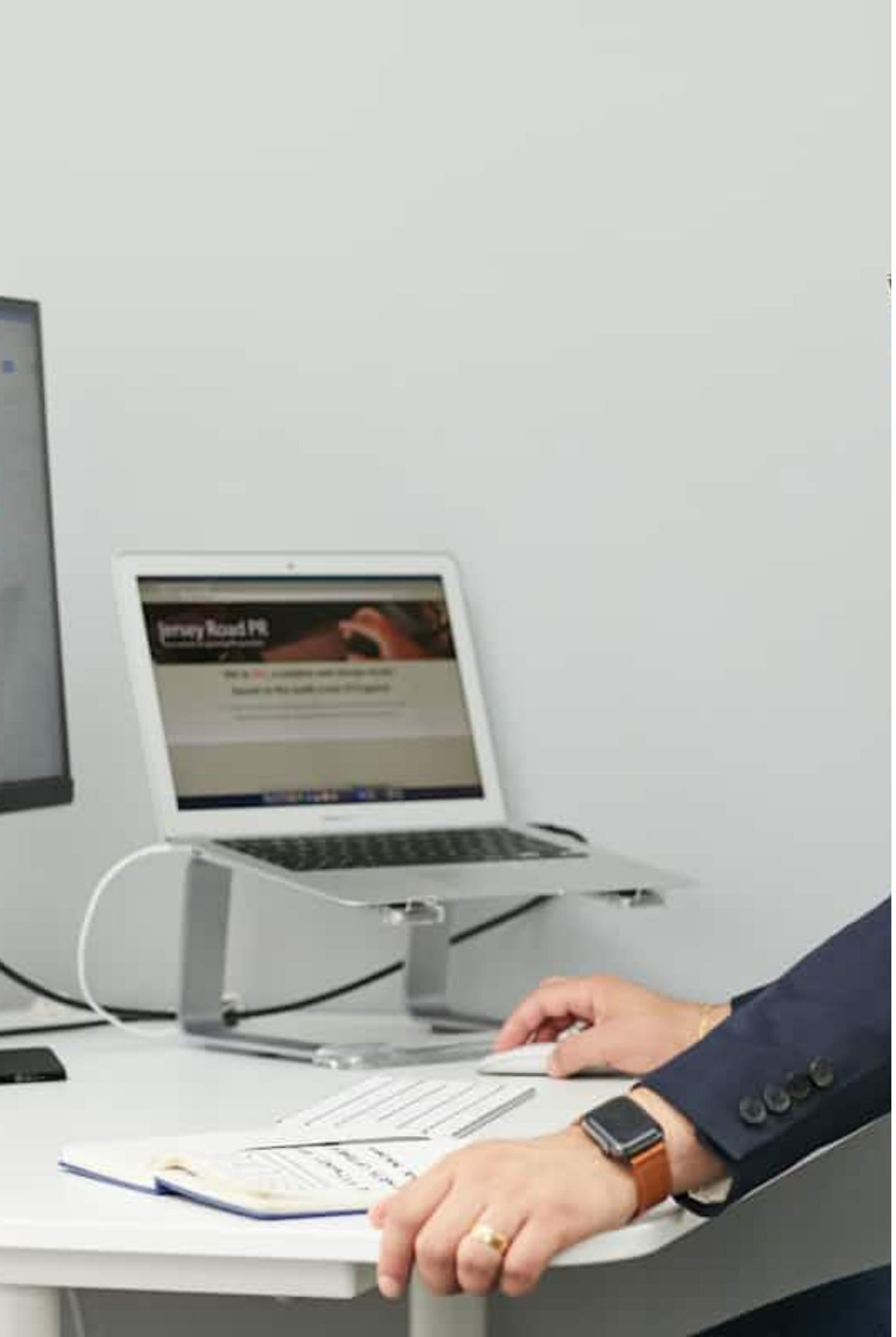
SCM is a set of practices to manage, organize, and control changes to software project assets.

It is the **single source of truth**.

- It answers:
  - What is our "current" build?
  - Who changed this file, and why?
  - What's the difference between v1.1 and v1.2?

Without SCM, you have chaos.





Quantitative Risk Analysis

Moving from qualitative prioritization to numeric impact assessment. We'll analyze our high-priority risks using mathematical techniques to determine their true cost and schedule implications on the overall project.

Decision Tree Analysis

Making Choices Under Uncertainty

Decision trees help visualize and evaluate choices when outcomes are uncertain. By calculating **Expected Monetary Value (EMV),** we can identify the financially optimal path forward.

Key Components

- **Square (□):** Decision Node – choices you control
- **Circle (○):** Chance Node – uncertain outcomes
- **Branch:** Possible paths and their consequences

EMV Formula

$$EMV = Probability \times Impact$$

Probability (P): Likelihood of outcome occurring

Impact ($): Financial consequence (cost or gain)

Rule: Choose the highest EMV or least negative value

Build vs. Buy Decision

Real-World EMV Application

Our authentication module needs replacement. Let's evaluate two strategic paths using quantitative analysis to inform our decision.

Path 1: Build In-House

Initial Cost: $50,000

Success Scenario (60%): Total cost remains $50k

Risk Scenario (40%): Major integration bugs add $40k rework cost, totaling $90k

Path 2: Buy Third-Party Solution

Base Cost: $30,000 license + $30,000 integration = $60,000

Success Scenario (90%): Works perfectly at $60k

Risk Scenario (10%): Heavy customization adds $70k, totaling $130k



EMV Calculation Walkthrough

Determining the Optimal Financial Choice

01

Calculate Build EMV

$$EMV_{Build} = (0.60 \times -\$50,000) + (0.40 \times -\$90,000)$$

$$EMV_{Build} = (-\$30,000) + (-\$36,000) = -\$66,000$$

02

Calculate Buy EMV

$$EMV_{Buy} = (0.90 \times -\$60,000) + (0.10 \times -\$130,000)$$

$$EMV_{Buy} = (-\$54,000) + (-\$13,000) = -\$67,000$$

03

Compare and Decide

Build EMV: **–$66,000**

Buy EMV: **–$67,000**

Conclusion: Despite the "Buy" option having a 90% success rate, the severe impact of its failure scenario makes "**Build**" the better financial choice. The numbers reveal what intuition might miss.

PERT Three-Point Estimating

Developed by the U.S. Navy in the 1950s for the Polaris missile project, PERT revolutionized how we handle massive uncertainty in complex projects—perfect for today's software development challenges.



The Three Estimates That Change Everything

Instead of forcing a single concrete number, PERT asks experts for three different scenarios. This approach acknowledges reality: software projects rarely follow a single predictable path.

Optimistic (O)

Best-case scenario: everything goes perfectly. No bugs, no integration issues, smooth sailing. This is your 1-in-100 shot where Murphy takes a vacation.

Most Likely (M)

The realistic estimate: what developers actually think it will take, accounting for normal, everyday problems and small setbacks that always arise.

Pessimistic (P)

Worst-case scenario: Murphy's Law in full effect. Database corrupts, APIs fail, requirements change mid-sprint. The 1-in-100 disaster estimate.

The Mathematics Behind PERT

Expected Duration

PERT uses a weighted average that heavily anchors to the Most Likely outcome while factoring in risk from optimistic and pessimistic values.

$$E = \frac{O + 4M + P}{6}$$

Notice we multiply M by 4, then divide by 6. This beta distribution approach is far superior to simple averaging.

Standard Deviation

PERT also measures uncertainty—the risk level of each estimate. This metric is pure gold for managers.

$$SD = \frac{P - O}{6}$$

Low risk: When P and O are close (e.g., 5 vs 7), SD is small—high confidence.

High risk: When P and O are far apart (e.g., 5 vs 25), SD is large—significant uncertainty requiring close monitoring.

Estimating a **Single Task**

Let's say we need to develop a new "**User Authentication**" module for our project. We ask our senior developer for her 3-point estimate, and she gives us the following (in days):

- **Optimistic (O):** 3 days
- **Most Likely (M):** 6 days
- **Pessimistic (P):** 15 days

What is the Expected Duration (E) and Standard Deviation (SD) for this task?

$$E = \frac{3 + (4 \times 6) + 15}{6}$$

$$E = \frac{3 + 24 + 15}{6}$$

$$E = \frac{42}{6}$$

$$E = \frac{O + 4M + P}{6}$$

1. Calculate the Expected Duration (E)

E = 7 days

Our single-point estimate for this task, which we will put on our Gantt chart, is 7 days. Notice this is not the same as the "Most Likely" (6 days).

The high pessimistic estimate (15 days) has pulled our expected duration up, which is good! It's forcing us to account for that risk.

Optimistic (O): 3 days | Most Likely (M): 6 days | Pessimistic (P): 15 days

2. Calculate the Standard Deviation (SD):

$$SD = \frac{P - O}{6}$$

- We have an **expected duration of 7 days** with a **standard deviation of 2 days**. In statistics (assuming a normal distribution).
- This tells us we have approximately a 68% confidence that the task will finish between E ±1 x SD (i.e., between 5 and 9 days) and a 95% confidence it will finish between E ± 2 x SD (i.e., between 3 and 11 days).

$$SD = \frac{15 - 3}{6}$$

$$SD = \frac{12}{6}$$

$$SD = 2 \text{ days}$$

Optimistic (O): 3 days | Most Likely (M): 6 days | Pessimistic (P): 15 days

Estimating a Project (A Sequence of Tasks)

Critical path (a sequence of tasks that must be done one after another) for a "User Profile Update" feature.

Task	Predecessor	Optimistic (O)	Most Likely (M)	Pessimistic (P)
A: Design UI	-	4	6	8
B: Code Frontend	A	10	12	20
C: Build API Endpoint	A	8	11	16
D: Integrate & Test	B, C	5	7	15

Assume the critical path (the longest path through the project) is **A → C → D**.



Questions

What is the Expected Duration (E) and Variance (SD^2) for each task on the path A-C-D?

What is the **total Expected Duration** for this path?

What is the **total Standard Deviation** for this path?

What is the probability the project path will be finished in **30 days or less**?

1. Calculate E and SD^2 for Each Task

Task A (Design UI)

- E_A = (4 + 24 + 8) / 6 = 36 / 6 = 6.0 days
- SD_A = (8 - 4) / 6 = 4 / 6 = ~0.67
- Variance_A = $(0.67)^2$ = 0.44

Task C (Build API)

- E_C = ~11.33 days
- SD_C = ~1.33
- Variance_C =1.78

Task D (Integrate & Test)

- E_D = ~8.0 days
- SD_D = ~1.67
- Variance_D = 2.78

$$E = \frac{O + 4M + P}{6} \qquad SD = \frac{P - O}{6}$$

2. Calculate Total Expected Duration ($E_{Project}$)

The total expected duration of a path is simply the sum of the expected durations of the tasks on it.

$$E_{Project} = E_A + E_C + E_D$$

$$E_{Project} = 6.0 + 11.33 + 8.0 = \mathbf{25.33\ days}$$

3. Calculate Total Standard Deviation ($SD_{Project}$)

DO NOT add the Standard Deviations. You must add the Variances, and then take the square root of the total.

$$Variance_{Project} = Variance_A + Variance_C + Variance_D$$

$$Variance_{Project} = 0.44 + 1.78 + 2.78 = \mathbf{5.00}$$

4. Find the project's Standard Deviation

We have a project path that is expected to take 25.33 days, with a standard deviation of 2.24 days.

$$SD_{Project} = \sqrt{Variance_{Project}}$$

$$SD_{Project} = \sqrt{5.00} \approx \mathbf{2.24\ days}$$

Probability of finishing in 30 days or less

A real answer to management. We need to find the Z-score (from your statistics class) for our target date.

The formula is: $Z = \frac{(\text{Target Date} - \text{Expected Date})}{SD_{Project}}$

Target Date (X) = 30 days

Expected Date (E) = 25.33 days

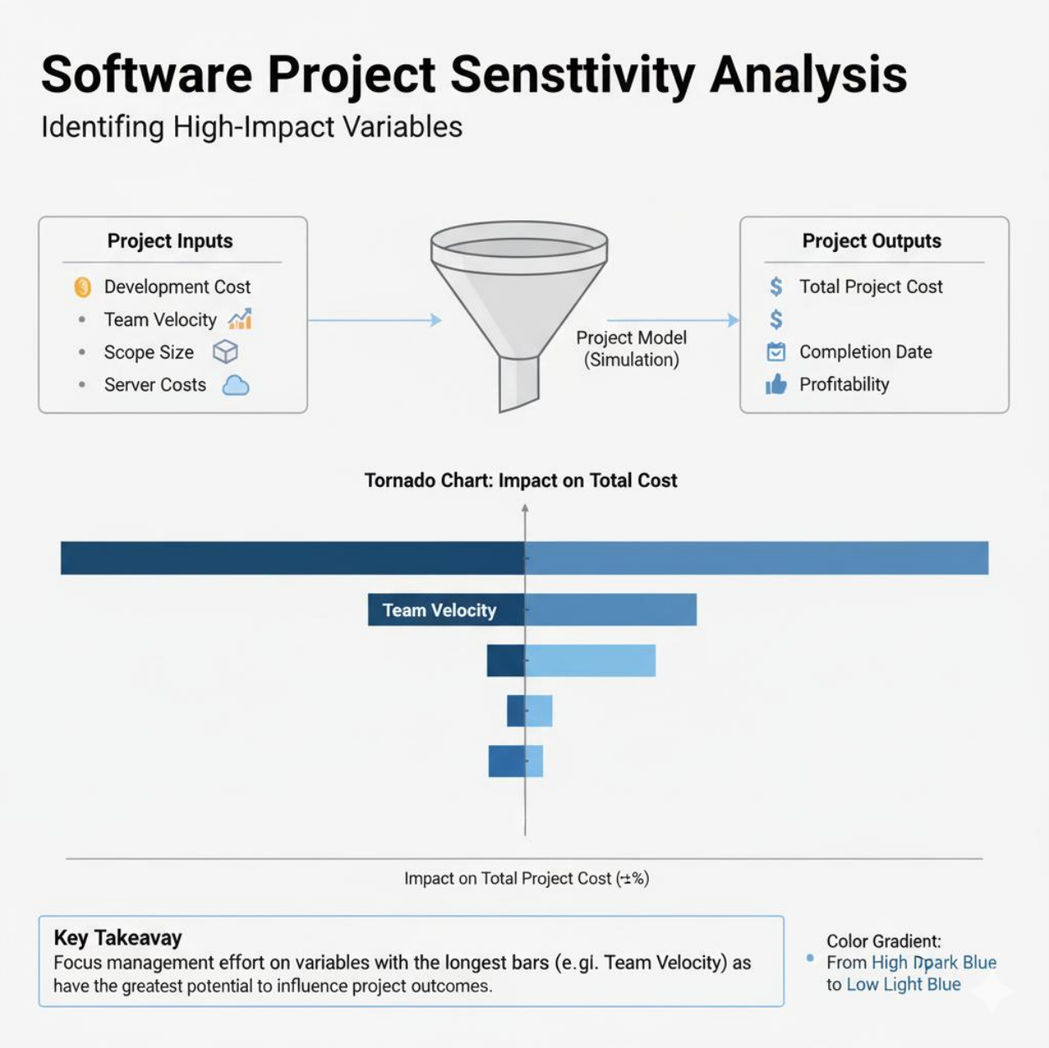
Project SD (σ)= 2.24 days

$$Z = \frac{30 - 25.33}{2.24}$$

$$Z = \frac{4.67}{2.24} \approx +\mathbf{2.08}$$

A Z-score of +2.08 means our target date is just over two standard deviations above the expected time.
A Z-score of +2.08 corresponds to a probability of 0.9812, or 98.12%.

We can tell our stakeholders, Based on our analysis, we have an expected completion time of 25.3 days. We have a **98.12% probability** of completing this feature path **within 30 days**.



Sensitivity Analysis

Identifying Your Most Critical Risks

Purpose and Process

Sensitivity analysis isolates each risk individually to measure its maximum potential impact on project objectives. By "freezing" all other variables, we can see which risks create the largest swing in outcomes.

The resulting **Tornado Diagram** ranks risks by their impact magnitude, creating a clear prioritization roadmap for response planning.

Reading the Tornado

- **Top bars (longest):** Highest sensitivity - these risks deserve immediate attention and robust response strategies
- **Middle bars:** Moderate impact - monitor closely and prepare contingencies
- **Bottom bars (shortest):** Minimal impact - accept or apply basic monitoring

Action: Focus resources on mitigating the top 3-5 risks in your tornado diagram

From Analysis to Action

Planning Risk Responses

Quantitative analysis provides the foundation for Plan Risk Responses. Armed with calculations, we can now develop targeted strategies for our most critical risks.

Avoid

Eliminate the risk by changing the project plan

Example: Use proven stable framework instead of risky beta version

Mitigate

Reduce probability or impact through proactive actions

Example: Build prototypes to test integration before full implementation

Transfer

Shift risk impact to third parties through contracts or insurance

Example: Use fixed-price vendor contracts or purchase project insurance

Accept

Acknowledge the risk exists and prepare contingency reserves

Example: Budget $20k contingency for 10% probability of UI rework

Key Takeaways

Mastering Quantitative Risk Analysis

1 Risk Equals Uncertainty

Our responsibility is proactive management, not reactive firefighting. Risk encompasses both threats and opportunities.

2 Strategic Planning Precedes Execution

The Risk Management Plan establishes your framework, processes, and rules before analysis begins.

3 Systematic Identification

Leverage Risk Breakdown Structure (RBS), Delphi technique, and Assumptions Analysis. Unidentified risks cannot be managed.

4 Qualitative Analysis Filters

The Probability-Impact Matrix efficiently prioritizes risks, revealing your critical "Top 10" requiring quantitative analysis.

5 Quantitative Techniques Drive Decisions

EMV and Decision Trees optimize choices. Monte Carlo generates S-curves for confidence levels and contingency sizing. Sensitivity Analysis (Tornado) prioritizes response efforts.

Understanding Project Risk Management

Identifying, analyzing, and managing risks in software projects through proven methodologies and practical techniques.



What is Project Risk?

"A **risk** is an **uncertain event or condition** that, if it occurs, has a **positive or negative effect** on one or more project objectives (scope, schedule, cost, quality)." - PMBOK® Guide

Uncertainty

A risk *might* happen - it's not an issue yet. An **issue** is a problem today (e.g., "The server is down"), while a **risk** is a potential problem tomorrow (e.g., "The server might crash during launch").

Effect

Every risk must have a measurable consequence on your project objectives. If there's no impact on scope, schedule, cost, or quality, it's not a risk worth tracking.

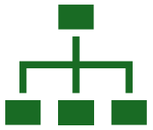
Positive vs. Negative

Threat: A security vulnerability discovered in your framework.
Opportunity: A new team member who's a subject matter expert, potentially cutting development time in half.

Project plans have to be based on assumptions | Risk is the possibility that an assumption is wrong | When the risk occurs, it becomes a problem or an issue

Categories of risk

This is based on Lyytinen's sociotechnical model of risk



Actors

relate to all those involved in the project including both developers, users and managers e.g. a risk could be that high staff turnover leads to information of importance to the project being lost



Technology

both that used to implement the project and that embedded in the project deliverables – risk could be that the technologies selected are not in fact appropriate.



Structure

this includes management procedures, risk here is that a group who need to carry out a particular project task are not informed of this need because they are not part of the project communication network



Tasks

the work to be carried out. A typical risk is that the amount of effort needed to carry out the task is underestimated.

Estimates being wrong could be influenced by problems with actors (e.g. lack of experience with a technical domain) or the structure (over optimism of managers keen to win work).



Plan Risk Management

The Foundation: Creating Your Risk Management Plan

Before identifying any risks, establish *how* you'll manage them throughout the project lifecycle. This proactive approach ensures consistency and clarity for your entire team.

01

Methodology

Define your tools and techniques. Will you use qualitative analysis for all risks? Quantitative EMV for high-priority cost risks?

02

Roles & Responsibilities

Assign clear ownership. Who facilitates risk meetings? Who owns database team risks?

03

Budget & Schedule

Allocate dedicated time and resources specifically for risk management activities.

04

Definitions & Categories

Establish your Risk Breakdown Structure and define what "High," "Medium," and "Low" actually mean with specific thresholds.

Common Sources of Risk in Software Projects

Understanding where risks originate helps teams look in the right places. A well-structured **Risk Breakdown Structure (RBS)** provides a systematic framework for comprehensive risk identification.



Technical Risks

- New or unproven technology
- Integration complexity
- Performance bottlenecks
- Security vulnerabilities



Organizational Risks

- Resource allocation conflicts
- Lack of stakeholder buy-in
- Unclear responsibilities
- Competing priorities



Project Management

- Poor estimation accuracy
- Uncontrolled scope creep
- Communication breakdowns
- Resource conflicts



External Risks

- Vendor or supplier failure
- Regulatory changes
- Market disruptions
- Competitive threats

Risk Identification Techniques

Proven Methods for Discovering Risks Before They Become Issues

1

Brainstorming

Gather your team and ask "What could go wrong?" The key principle: no judgment during idea generation. Capture everything first, filter later.

2

Delphi Technique

Anonymous, iterative expert consultation that prevents groupthink. A facilitator sends questionnaires to experts, summarizes responses, and repeats until consensus emerges.

3

Checklists

Leverage lessons learned from past projects. Review risk registers from similar initiatives to identify recurring patterns and blind spots.

4

Assumptions Analysis

Question every assumption. What are you taking for granted? If that assumption proves false, you've identified a risk.

5

Documentation Review

Scrutinize project charters, requirements, and SOWs. Look for ambiguity, contradictions, or "TBD" sections – these are risk indicators.

Boehm’s top 10 development risks

Personnel shortfalls	Unrealistic time and cost estimates	Developing the wrong software functions	Developing the wrong user interface
• Staffing with top talent; job matching; teambuilding; training and career development; early scheduling of key personnel	• Multiple estimation techniques; design to cost; incremental development; recording and analysis of past projects; standardization of methods	• Improved software evaluation; formal specification methods; user surveys; prototyping; early user manuals	• Prototyping; task analysis; user involvement

Boehm’s top ten risk - continued



Gold plating

Requirements scrubbing, prototyping, design to cost



Late changes to requirements

Change control, incremental development



Shortfalls in externally supplied components

Benchmarking, inspections, formal specifications, contractual agreements, quality controls



Shortfalls in externally performed tasks

Quality assurance procedures, competitive design etc



Real time performance problems

Simulation, prototyping, tuning



Development technically too difficult

Technical analysis, cost-benefit analysis, prototyping , training

The Risk Register

Your Single Source of Truth

If a risk isn't documented in the register, it doesn't exist for your project. This living document evolves throughout the entire project lifecycle, starting during identification and continuously updated as new information emerges.

ID	Risk Description	Category	Cause	Effect
R-001	Database query for user report is too slow under production load	Technical	Inefficient query design; no load testing	Report feature unusable; user dissatisfaction
R-002	Senior back-end developer might leave for a new job	Organizational	High market demand for their specialized skills	Key modules delayed; critical knowledge lost
R-003	Client may request significant UI changes after UAT	Project Management	Vague initial requirements for the user interface	Extensive rework required; schedule and cost overrun

Additional columns will be added during analysis: Probability, Impact, Risk Score, Owner, and Response Strategy.

Qualitative Risk Analysis

Prioritizing Risks Through Systematic Assessment

Qualitative analysis provides the essential first filter for all identified risks. By assessing **probability of occurrence** and **impact on objectives**, teams can quickly prioritize which risks demand immediate attention and deeper analysis.

Fast & Subjective

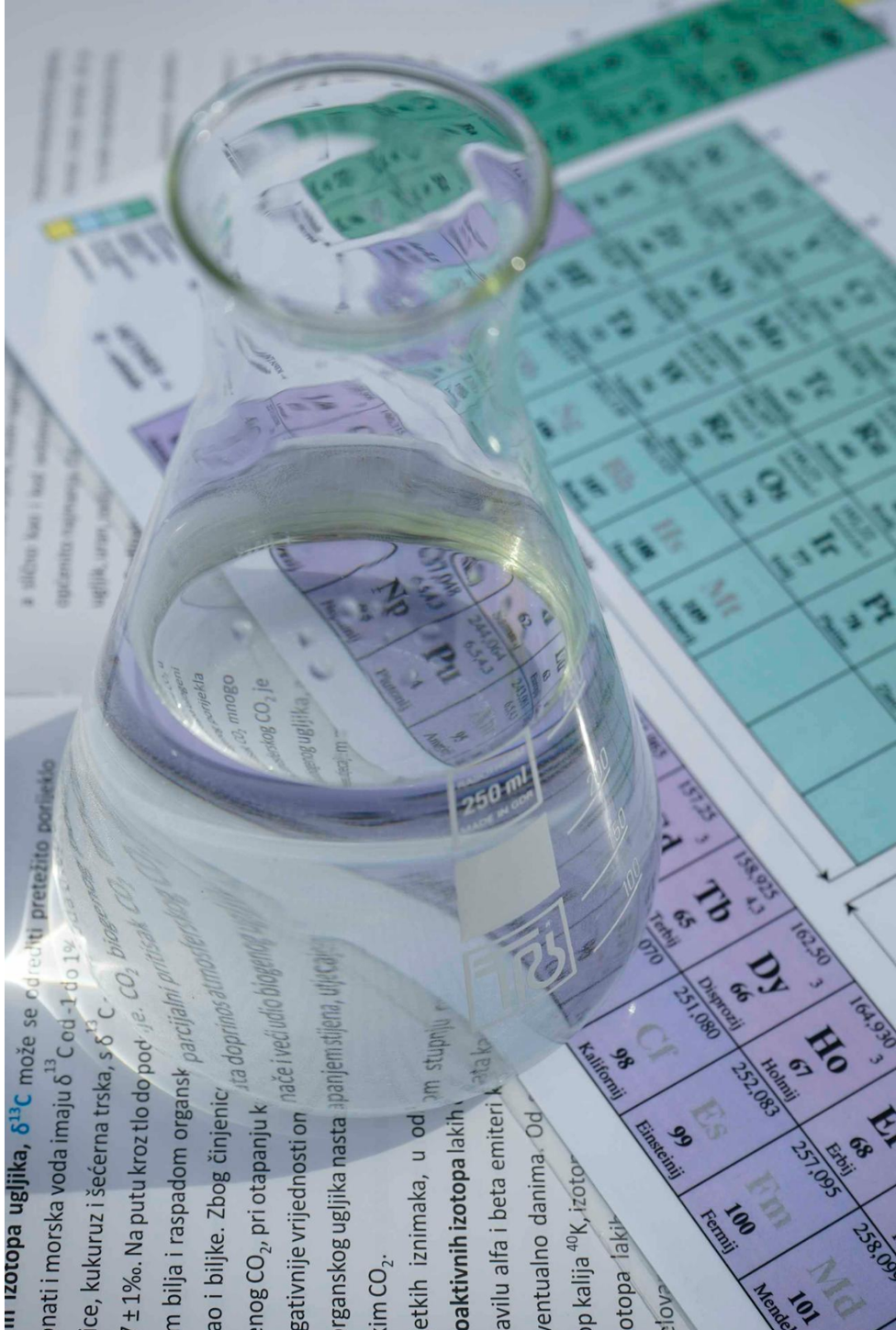
The "quick and dirty" method that separates critical risks from minor concerns

Universal Application

Every project, regardless of size or complexity, should perform qualitative analysis

Essential Inputs

Risk Register, expert judgment, and probability-impact definitions from your Risk Management Plan



Risk probability: qualitative descriptors

Probability level	Range
High	Greater than 50% chance of happening
Significant	30-50% chance of happening
Moderate	10-29% chance of happening
Low	Less than 10% chance of happening





Qualitative descriptors of impact on cost and associated range values

Impact level	Range
High	Greater than 30% above budgeted expenditure
Significant	20 to 29% above budgeted expenditure
Moderate	10 to 19% above budgeted expenditure
Low	Within 10% of budgeted expenditure.

The Probability-Impact Matrix

This powerful visual tool plots each risk across two critical dimensions, creating a heat map that immediately reveals priorities. Calculate your risk score using this formula:

$$Risk\ Score = Probability \times Impact$$

Red Zone

High-High Risks

Demand immediate attention and detailed response planning. These are your top priorities.

Yellow Zone

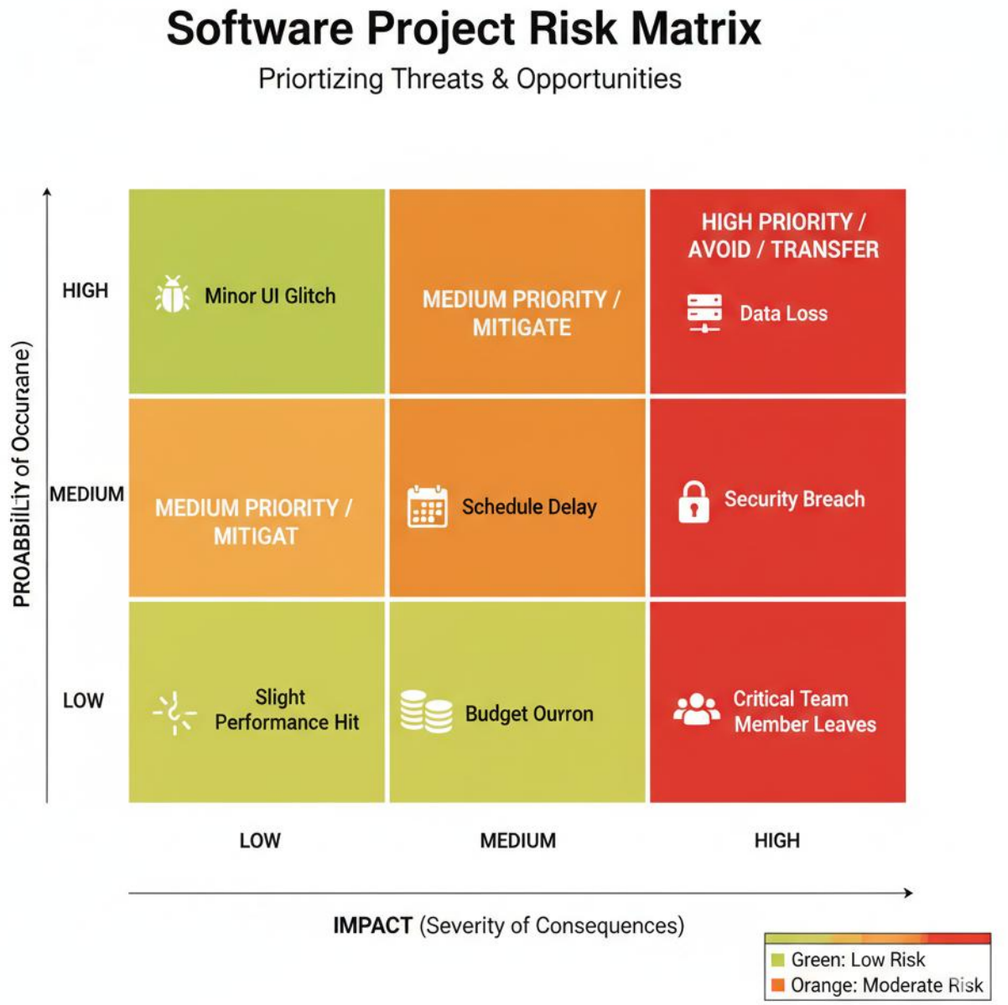
Medium Risks

Monitor closely and develop contingency plans. May escalate to red zone.

Green Zone

Low Risks

Accept and document. Periodic review ensures they haven't changed status.



Bringing It All Together

Updated Risk Register with Qualitative Analysis

After completing qualitative analysis, your risk register transforms into a prioritized action list. This enables data-driven decisions about where to focus resources, and which risks require quantitative analysis.

ID	Risk Description	Category	Probability	Impact	Score	Priority
R-001	DB query is too slow	Technical	High (3)	High (3)	**9**	High
R-002	Senior dev might leave	Organizational	Medium (2)	High (3)	**6**	High
R-003	Client scope creep	Project Mgmt	High (3)	Medium (2)	**6**	High
R-004	UI typo on login page	Technical	Low (1)	Low (1)	**1**	Low

---

The Power of Expert Judgment

Expert judgment isn't a separate step - it's a critical *technique* woven throughout every phase of risk management. Leverage senior technical staff, experienced project managers, subject matter experts, and external consultants to validate your analysis and uncover blind spots.

Caution: Be aware of cognitive biases like optimism bias ("That will never happen to my code!"). Anonymous techniques like Delphi help counter these natural human tendencies.

Risk Management

Risk Response Planning & Risk Monitoring and Control



What is Risk Response Planning?

Risk Response Planning is the systematic process of developing strategic options, selecting appropriate strategies, and establishing concrete actions to address overall project risk exposure while treating individual project risks effectively.

Primary Goal: Develop a comprehensive action plan *before* risk events materialize, enabling proactive rather than reactive management.

Negative Risks

Reduce probability and/or impact of threats

Positive Risks

Increase probability and/or impact of opportunities

01

Appropriate to Risk Significance

Response matches severity level

02

Cost-Effective

Benefits outweigh response costs

03

Timely

Implemented before risk occurs

04

Realistic

Achievable within project constraints

05

Stakeholder Agreement

Approved by all relevant parties

Threat Response Strategies: Avoid & Mitigate

AVOID

Strategy: Change the project plan to completely eliminate the risk by modifying scope, schedule, or technical approach.

Example: Risk of using unstable beta-version library that could crash in production.

Response: Select a stable, mature version of the library or switch to a different, proven library with established reliability.

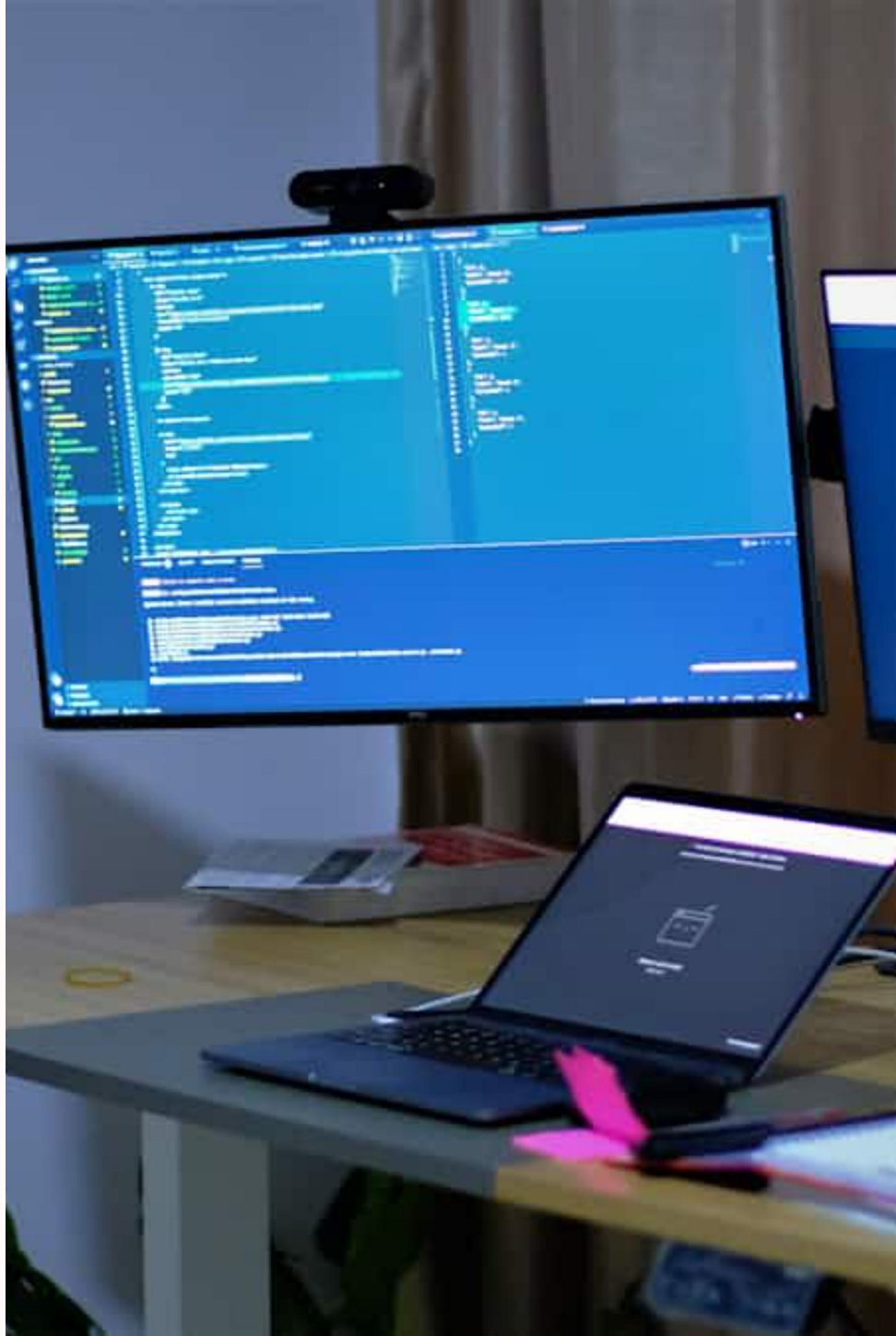
MITIGATE

Strategy: Take proactive steps to reduce either the probability of occurrence or the impact magnitude of the risk.

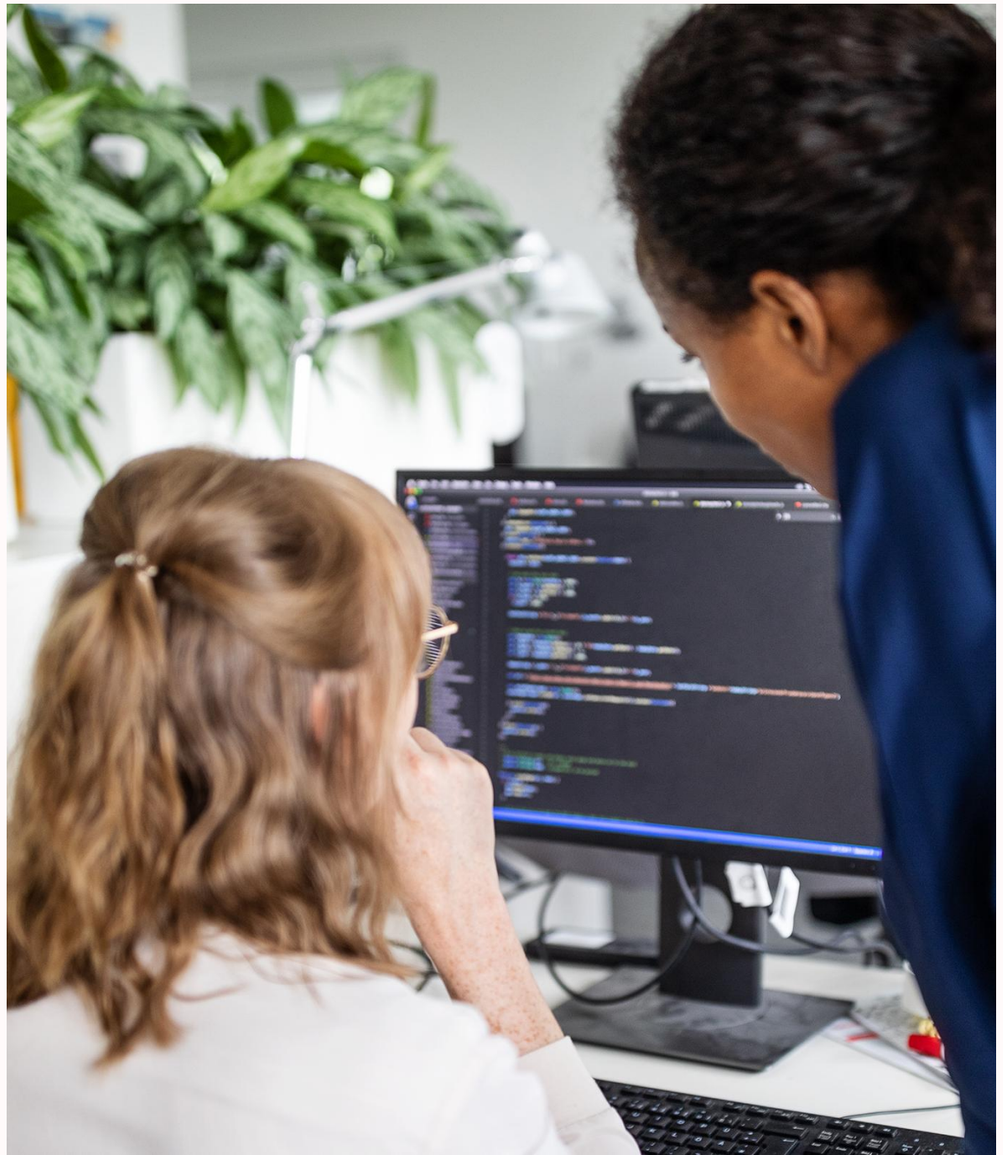
Example: Risk of high defect rate in the critical integration module.

Response Actions:

- Conduct extensive unit testing before integration begins (reduces impact)
- Assign most senior developers to integration tasks (reduces probability)
- Implement mandatory formal code review process



Threat Response Strategies: Transfer & Accept



TRANSFER

Strategy: Shift the risk or its financial consequences to a third party. This doesn't eliminate the risk—it transfers liability.

Example: Risk that servers can't handle traffic load during major launch event.

Response: Deploy on AWS or Azure with high-availability SLA. If hardware fails, the cloud provider handles recovery and may provide compensation.

ACCEPT

Strategy: Acknowledge the risk but take no proactive action. This must be a conscious, documented decision.

Active Acceptance: Document the risk and establish a contingency reserve (time/money).

Passive Acceptance: Simply note and monitor the risk.

Example: Client logo appears pixelated on rare 1024×768 CRT monitors (0.1% probability, minimal impact).

Response: Actively accept—inform client, document decision, move forward without fix.

The Living Risk Register

A dynamic document that evolves throughout the project lifecycle

The Risk Register serves as the central repository for all risk information. It must be reviewed and updated regularly—this is not a "file-and-forget" artifact.

Risk ID	Risk Description	Prob.	Impact	Score	Owner	Strategy	Specific Actions	Triggers	Contingency Plan
R-01	Key developer (Priya) on auth module leaves project	M	H	High	PM	Mitigate	1. Enforce code documentation2. Mandate knowledge-sharing sessions	Priya gives notice	Hire contract developer with security expertise (3-day schedule buffer)
R-02	3rd-party Payment API integration more complex than expected	H	H	High	Tech Lead	Mitigate	1. Build prototype spike in Week 12. Assign senior developer	Spike exceeds 3 days	Switch to backup provider (PayPal) with simpler API
R-03	Server fails during critical load testing	L	M	Low	Infra Lead	Transfer	Deploy on AWS with auto-scaling and SLA	N/A	Handled by AWS SLA

Risk Monitoring & Control

Continuous vigilance throughout the project lifecycle

Risk Monitoring & Control is the continuous process of tracking identified risks, monitoring residual risks, identifying new risks, and evaluating risk response effectiveness. This is an **ongoing background process** that runs for the entire project duration.

MONITORING

"The Watching"

- Track risk triggers and warning signs
- Re-assess existing risks regularly
- Identify new emerging risks
- Validate assumptions remain true

CONTROL

"The Doing"

- Execute response plans when triggered
- Evaluate plan effectiveness
- Implement corrective actions
- Request formal changes as needed



Risk Monitoring & Control Techniques



Risk Re-assessment

Periodically review and re-prioritize the entire risk register during regular team meetings. Determine if low-priority risks have escalated or if new risks have emerged that require attention.



Risk Audits

Formal examination of risk response effectiveness and the risk management process itself. Ask the critical question: "Are we actually implementing what our risk register specifies?"



Variance & Trend Analysis

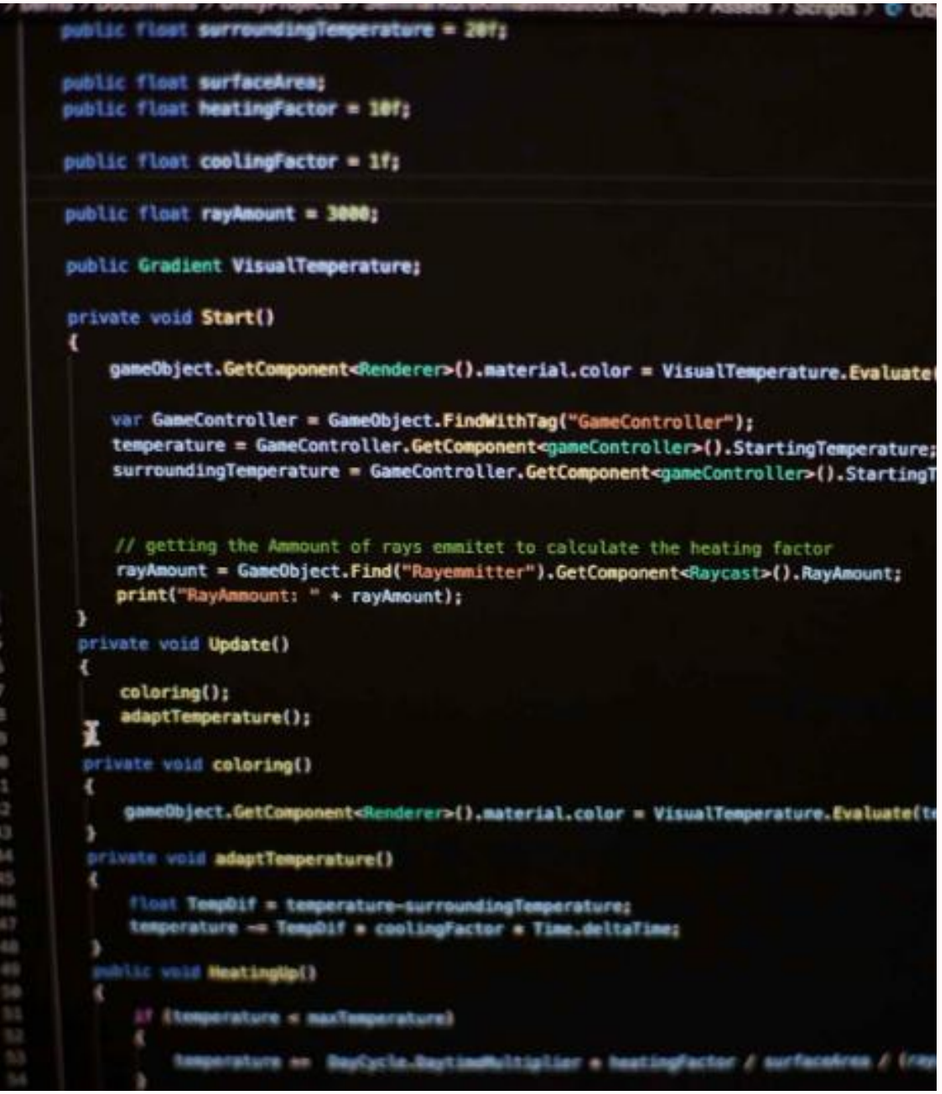
Compare planned results against actuals to identify deviations. Use burn-down charts to track velocity, monitor defect trends, and apply Earned Value Management (EVM) to measure value delivery.



Regular Status Meetings

The simplest yet most effective monitoring tool. Make "Risk Review" a mandatory 10-minute agenda item for every weekly status meeting to maintain continuous awareness.

Key Outputs from Monitoring & Control



Updated Risk Register

Risks are formally closed when no longer applicable. New risks are added as identified. Risk priorities are continuously updated based on current project conditions.

Change Requests

When risk responses or actualized risks impact scope, schedule, or cost, formal change requests must be submitted through the change control process.

Example: "Executing contingency plan R-02 (PayPal switch) adds 5 days to schedule. Submitting formal change request."

Corrective & Preventive Actions

Corrective: Immediate recommendations for current issues requiring action now.

Preventive: Proactive measures to prevent future problems from occurring.

Lessons Learned Database

Comprehensive documentation of all findings, successes, and failures to ensure future projects benefit from accumulated knowledge and avoid repeating mistakes.

The Risk Management Cycle

Identify Risks

Discover and document potential risks

Analyze & Prioritize

Assess probability and impact



Monitor & Control

Track, evaluate, and adjust continuously

Plan Response

Develop strategies and actions

Risk management operates as a continuous cycle throughout the project lifecycle. Each phase informs the next, creating a feedback loop that strengthens your project's resilience against uncertainty.



Key Takeaways

Being Prepared Beats Predicting

1 Response Planning Creates Your Playbook

Establish clear strategies—**Avoid, Mitigate, Transfer, or Accept**—before risks materialize. This is your preparation phase.

2 The Risk Register is Your Central Hub

Every risk must have designated **owners**, defined **triggers**, and documented **contingency plans**. This living document guides all risk activities.

3 Monitoring & Control is Your In-Game Process

Watch continuously and act decisively throughout the entire project. Use trend analysis and status meetings to maintain situational awareness.

Your Role as Project Manager: Your job isn't to prevent all problems—that's impossible. Your job is to ensure that when problems occur, nobody is surprised, and *everyone* knows exactly what to do.

Stakeholder Management



Topics

01

What is a stakeholder?

Defining the core concept and clearing common misconceptions

02

Why do we care?

Understanding how stakeholders can make or break your project success

03

Who are they?

Identifying the complete stakeholder universe for your projects

04

How do we manage them?

The 4-Step PMBOK Process, Power/Interest Grid, and Engagement Matrix

What is a Stakeholder?

A **stakeholder** is any person, group, or organization that can:

- **Impact** your project (e.g., approve funding, write code, make decisions)
- **Be impacted** by your project (e.g., end-users, support teams, communities)
- **Perceive** themselves to be impacted, regardless of actual impact

Key Distinction: This definition is much broader than a **shareholder**, who only has a financial stake. In software engineering, stakeholders include users, clients, team members, legal departments, and many others.





Why Bother? The 'Make or Break' Principle

Stakeholders are the ultimate source of all **requirements** and project success factors.

Scope Creep

The client continuously adds "just one more thing" without formal change control

Gold Plating

Your team builds "cool" features users never requested, wasting valuable resources

User Resistance

End-users refuse to adopt the new system because they were never consulted

Project Cancellation

A key stakeholder (like the sponsor) loses interest and pulls critical funding

Stakeholder management is **proactive risk management** for people.

Who is a Stakeholder?

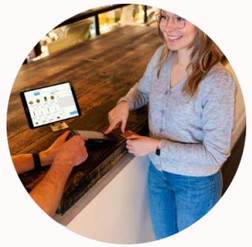
> **Definition:** Any individual, group, or organization who may affect, be affected by, or *perceive themselves* to be affected by the project.

This definition is intentionally broad—and that's critical. You can't manage expectations for stakeholders you don't even know exist.



Project Sponsor

Provides resources and strategic direction



Customers & Users

Ultimate recipients of project deliverables



Project Team

Execute the day-to-day work



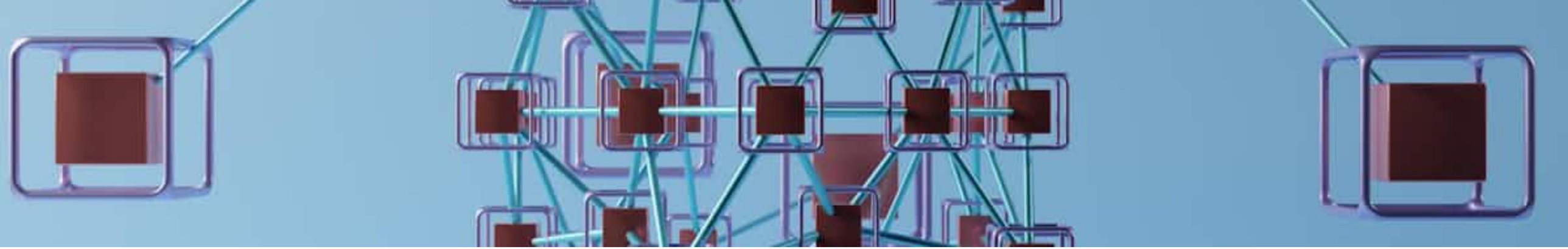
Functional Managers

Control resource allocation



Regulators

Ensure compliance and standards



The Stakeholder Universe

Internal Stakeholders

People inside your organization who have direct involvement:

- Project Manager
- Development Team
- QA and Testing Team
- C-level executives (CEO, CTO, CFO)
- Cross-functional departments (Marketing, Sales, Operations)
- IT Support and Infrastructure teams

External Stakeholders

People outside your organization with vested interests:

- The **Customer/Client** (funding the project)
- The **End-Users** (using the software)
- Suppliers and Vendors
- Government or Regulatory bodies
- Competitors and Industry partners
- The General Public

Real-World Example: Your CSE Project

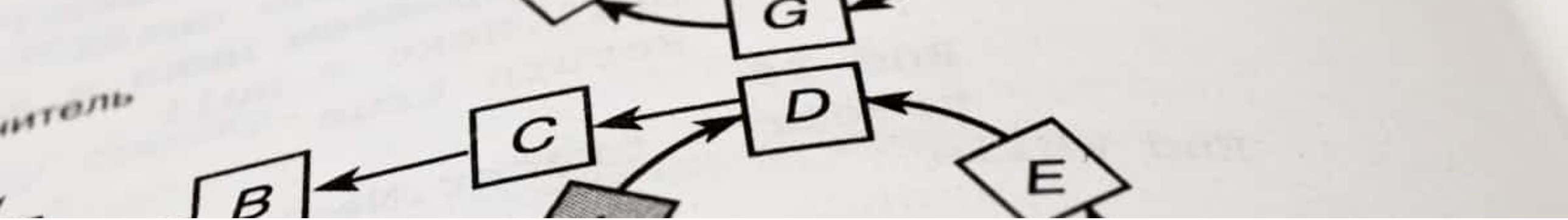
Let's identify stakeholders for a typical 5th-semester software engineering project. You're already managing stakeholders—now let's do it with a structured process.

Internal Stakeholders

- **You and your team:** High interest in grades and learning outcomes
- **Project Guide (Professor):** High power and high interest—approves topics and assigns grades
- **HOD or Dean:** Low interest but high power—sets deadlines and department policies
- **Lab coordinators:** Provide resources and infrastructure support

External Stakeholders

- **End-users:** Students using your "Hostel Mess Feedback App" or similar project
- **Competition judges:** Evaluating your work if presented at hackathons
- **Future employers:** Passive stakeholders viewing your project on your resume
- **Industry mentors:** Providing guidance and potential opportunities



The 4-Step Stakeholder Management Process

The Project Management Body of Knowledge (PMBOK) provides a formal, iterative framework for managing stakeholders throughout the project lifecycle. This is not a one-time activity!

1. Identify Stakeholders

Initiating Phase: Discover everyone with a stake in the project

2. Plan Engagement

Planning Phase: Develop strategies for each stakeholder group

3. Manage Engagement

Executing Phase: Communicate and build relationships actively

4. Monitor Engagement

Monitoring Phase: Track relationships and adjust strategies

Step 1: Identify Stakeholders



Goal

Find everyone who has a stake in your project. This requires thorough brainstorming and investigation beyond the obvious players.

Actions to Take

- Review your project charter and initial documentation
- Conduct interviews with your project sponsor and team
- Ask the critical question: "Who else cares about this project's outcome?"
- Consider both direct and indirect impacts

Key Output: Stakeholder Register

This is not merely a list of names—it's a comprehensive document (typically a spreadsheet) containing:

- Name, role, and organization
- Contact information
- Key requirements and expectations
- Level of influence (High/Medium/Low)
- Level of interest (High/Medium/Low)
- Potential impact on project success

Step 2: Plan Engagement

The Power/Interest Grid

This is the most famous and practical tool in stakeholder management. Once you have your stakeholder register, you must **prioritize** strategically. You cannot—and should not—treat everyone equally.

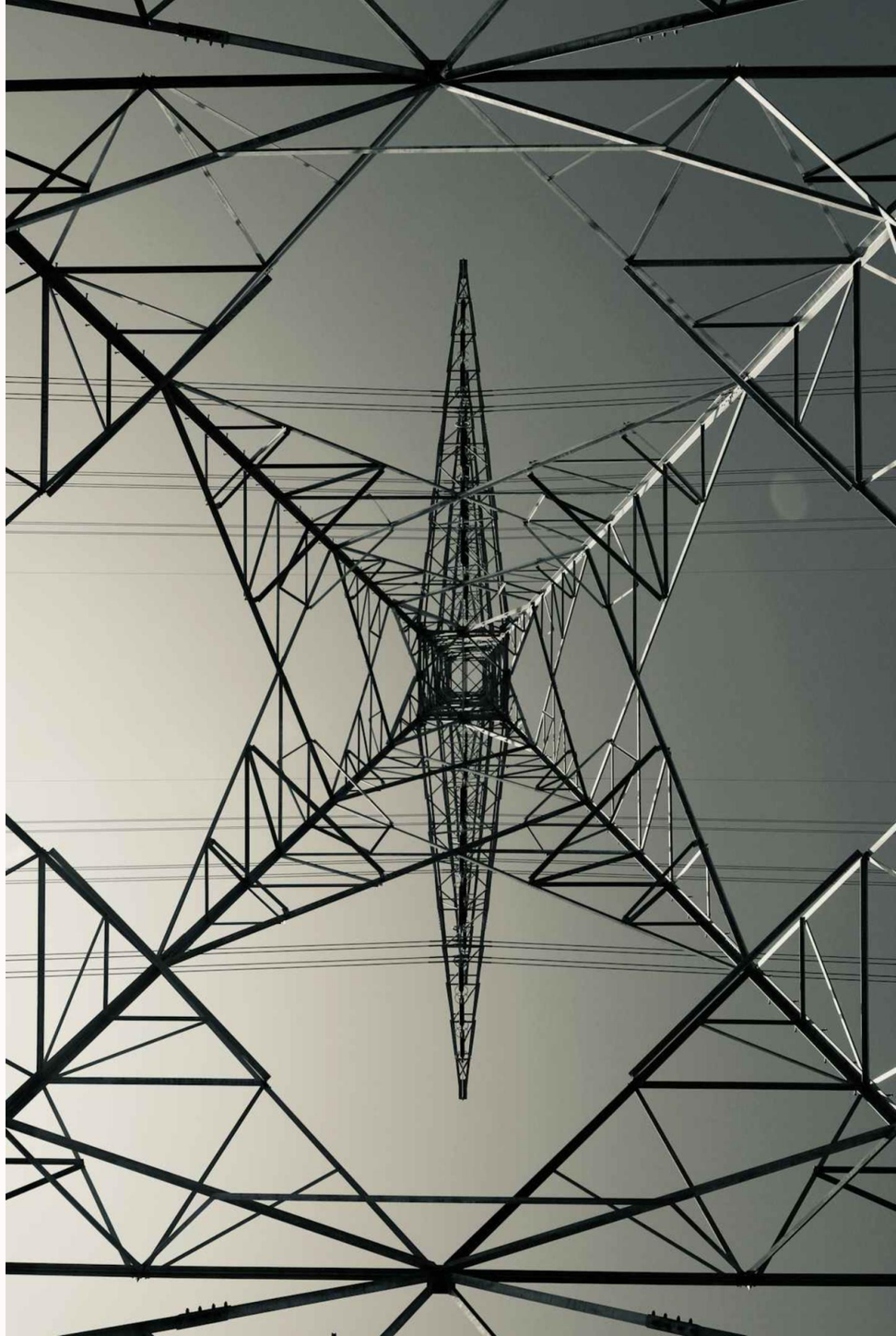
The Two Dimensions

X-axis: Level of **Interest** in your project (Low → High)

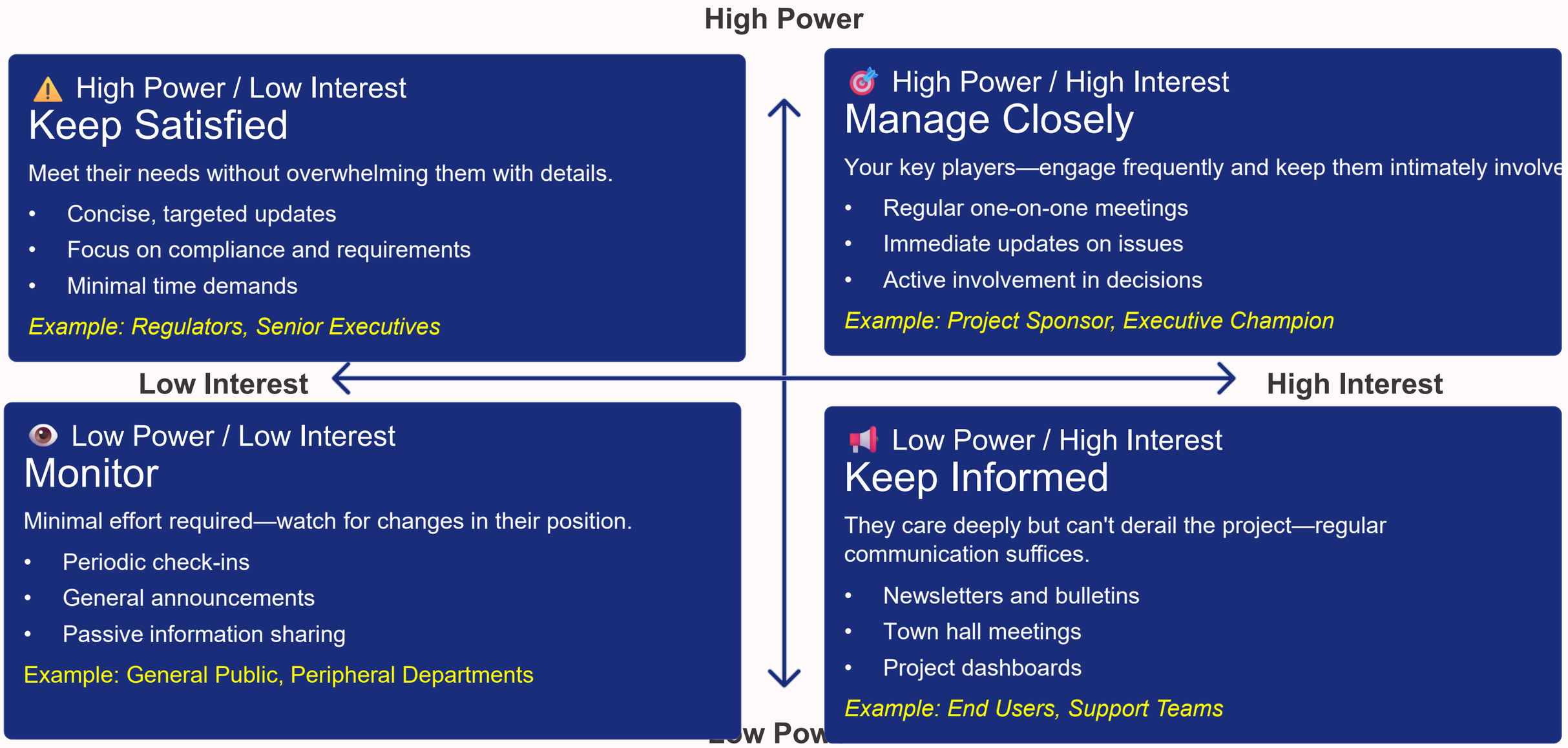
Y-axis: Level of **Power** or Influence over your project (Low → High)

Why It Matters

This grid tells you *exactly* how to allocate your limited time and communication resources for maximum stakeholder satisfaction and project success.



Stakeholder Analysis: The Power/Interest Grid



The Power/Interest Grid Strategies

High Power / High Interest

Example: Your Project Guide or Client

Strategy: Manage Closely

Engage them constantly. Consult them on all key decisions. Make them your active partner in success. Schedule regular check-ins and involve them in problem-solving.

High Power / Low Interest

Example: The Dean or C-Suite Executives

Strategy: Keep Satisfied

Don't overwhelm them with daily updates. Provide concise, high-level summaries only when necessary. Ensure they never become dissatisfied or feel blindsided.

Low Power / High Interest

Example: End-Users or Beta Testers

Strategy: Keep Informed

Send regular newsletters and hold feedback sessions. Make them feel heard and valued, but maintain control of scope. Their enthusiasm shouldn't derail priorities.

Low Power / Low Interest

Example: General Public

Strategy: Monitor

Apply minimal effort. Simply watch for any changes in their level of interest or power that might require a shift in your approach.

Step 2: Plan (Part B) - The Engagement Matrix

Analysis is one thing; strategy is another. For your key stakeholders—especially the "Manage Closely" group—you need a clear picture of their current level of engagement and where you need them to be.

Project Guide

Current: Supportive

Desired: Supportive

Strategy: Keep up the good work. Maintain regular communication and continue leveraging their support.

Lead Developer

Current: Neutral

Desired: Supportive

Strategy: Hold 1:1 meeting to align on project vision and technical approach. Address any concerns early.

Head of Security

Current: Resistant

Desired: Neutral

Strategy: Invite to demo to address security concerns directly. Present compliance measures and risk mitigation plans.

End-User Group

Current: Unaware

Desired: Supportive

Strategy: Set up bi-weekly feedback sessions. Involve them early to build ownership and capture requirements.

Engagement Levels: U = Unaware | R = Resistant | N = Neutral | S = Supportive | L = Leading

Stakeholder	Current (C)	Desired (D)	Strategy to Bridge the Gap
Project Guide	S	S	(Keep up the good work)
Lead Developer	N	S	"Hold 1:1 meeting to align on project vision."
Head of Security	R	N	"Invite to demo to address security concerns."
End-User Group	U	S	"Set up bi-weekly feedback sessions."

Engagement Levels: U = Unaware | R = Resistant | N = Neutral | S = Supportive | L = Leading

Step 3 & 4: Manage & Monitor

This is where the plan transforms into action. One executes the strategy, the other ensures it's working.

Manage Stakeholder Engagement

This is the *act* of communication—the execution phase where your engagement plan comes to life.

- Running feedback sessions and status meetings
- Sending targeted status reports
- Resolving conflicts as they arise
- Building trust through consistent delivery
- Ensuring expectations are met or managed

This is about building relationships and creating a collaborative environment where stakeholders feel heard and valued.

Monitor Stakeholder Engagement

This is the "check" step—measuring effectiveness and adapting your approach based on results.

- Is your engagement plan working?
- Did the Resistant Head of Security move to Neutral?
- Are stakeholder concerns being addressed?
- Have new stakeholders emerged?
- Do engagement strategies need adjustment?

Stakeholders change! People get promoted, leave the organization, or shift their perspectives. Your Stakeholder Register is a living document that requires continuous updates.

Your Role as an Engineer

You Are a Stakeholder

You have interests, concerns, and influence in the project's outcome. Your technical expertise makes you a critical voice in decision-making.

You Are a Stakeholder Manager

Every interaction with your PM, client, or user is stakeholder management. You're constantly identifying needs, analyzing impact, and engaging to drive success.

Everyone Is a Stakeholder

Your Project Manager guides the work. Your Client defines success. Your Users determine value. Each relationship requires intentional management.

> Stakeholder Management is the structured process of identifying, analyzing, planning, and engaging with people to ensure your technically-sound project is also a *successful* one.

You are not "just a coder." You're a problem solver, a communicator, and a bridge between technical excellence and business value. The most successful engineers understand that brilliant code means nothing if stakeholders don't support, understand, or adopt it. Master stakeholder management, and you'll transform from a great engineer into an indispensable technical leader.

Project Team Management & Resource Planning



What is Project HR Management?

Organize

Structure your team for maximum effectiveness and clear communication flows

Manage

Coordinate resources, skills, and workloads to meet project objectives

Lead

Guide the team toward shared goals with clear direction and accountability

The goal: Get the **right people** with the **right skills** in the **right roles** at the **right time**. In software projects, mismatched skills or poor team structure leads to bugs, delays, and project failure.



Step 1: Human Resource Planning

The Foundation: Who Do We Need?

This process identifies and documents project roles, responsibilities, required skills, and reporting relationships. It transforms abstract project needs into concrete staffing plans.



Key Inputs

- Project Scope Statement & Work Breakdown Structure (WBS)
- Company culture & existing organizational structure
- Organizational policies, templates, and historical data

Primary Output

Human Resource Management Plan – your roadmap for team success

Acquiring the Project Team

The Goal

Obtain the human resources needed to complete your project successfully. Think of it as assembling your ultimate tech squad for a high-stakes mission.

Key Planning Inputs

- **Project Management Plan:** What roles do we need? (Frontend Dev, Backend Dev, QA Engineer)
- **Resource Requirements:** Specific skills needed ("Senior Python Developer with Django experience")



Pre-assignment

Team members already chosen for their expertise. "Sarah is our database expert, so she's leading data architecture."

Negotiation

"Borrowing" resources from other departments. Requires strong negotiation skills with functional managers.

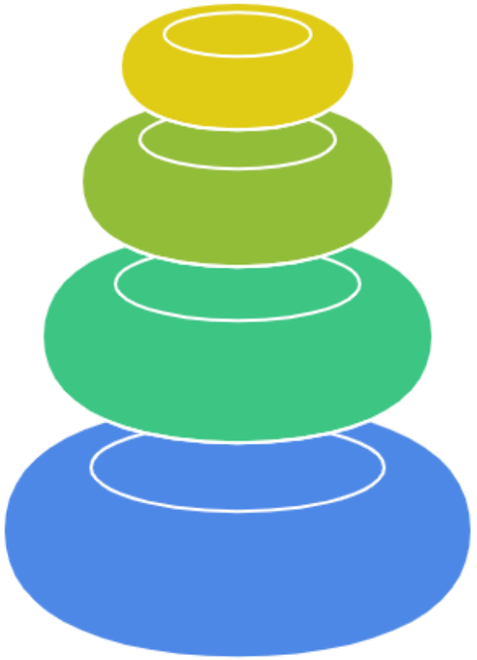
Acquisition

Hiring new employees or contractors when internal skills don't exist.

Virtual Teams

Geographically dispersed remote teams - extremely common in today's tech landscape.

Step 2: Project Organizational Charts



Project Manager

Oversee project execution and delivery

Team Leads

Guide and coordinate team activities

Testers

Ensure software quality and reliability

Developers

Implement and maintain software code

Visualizing Team Structure

Clear Hierarchy

Shows who reports to whom, establishing clear lines of authority and communication

Team Relationships

Illustrates how different roles and departments connect within the project structure

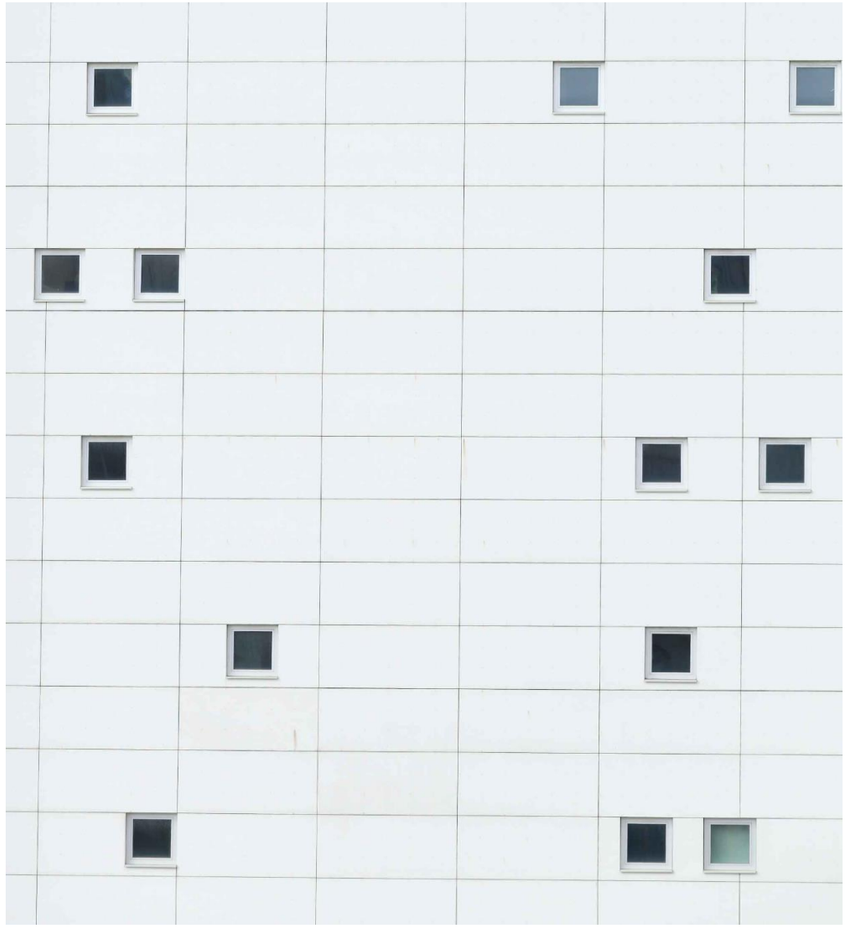
Communication Paths

Defines formal channels for information flow and decision-making processes

A well-designed org chart **eliminates ambiguity** about team structure and provides a **visual reference** for all stakeholders to understand their place in the project ecosystem.

Step 3: Responsibility Assignment Matrix

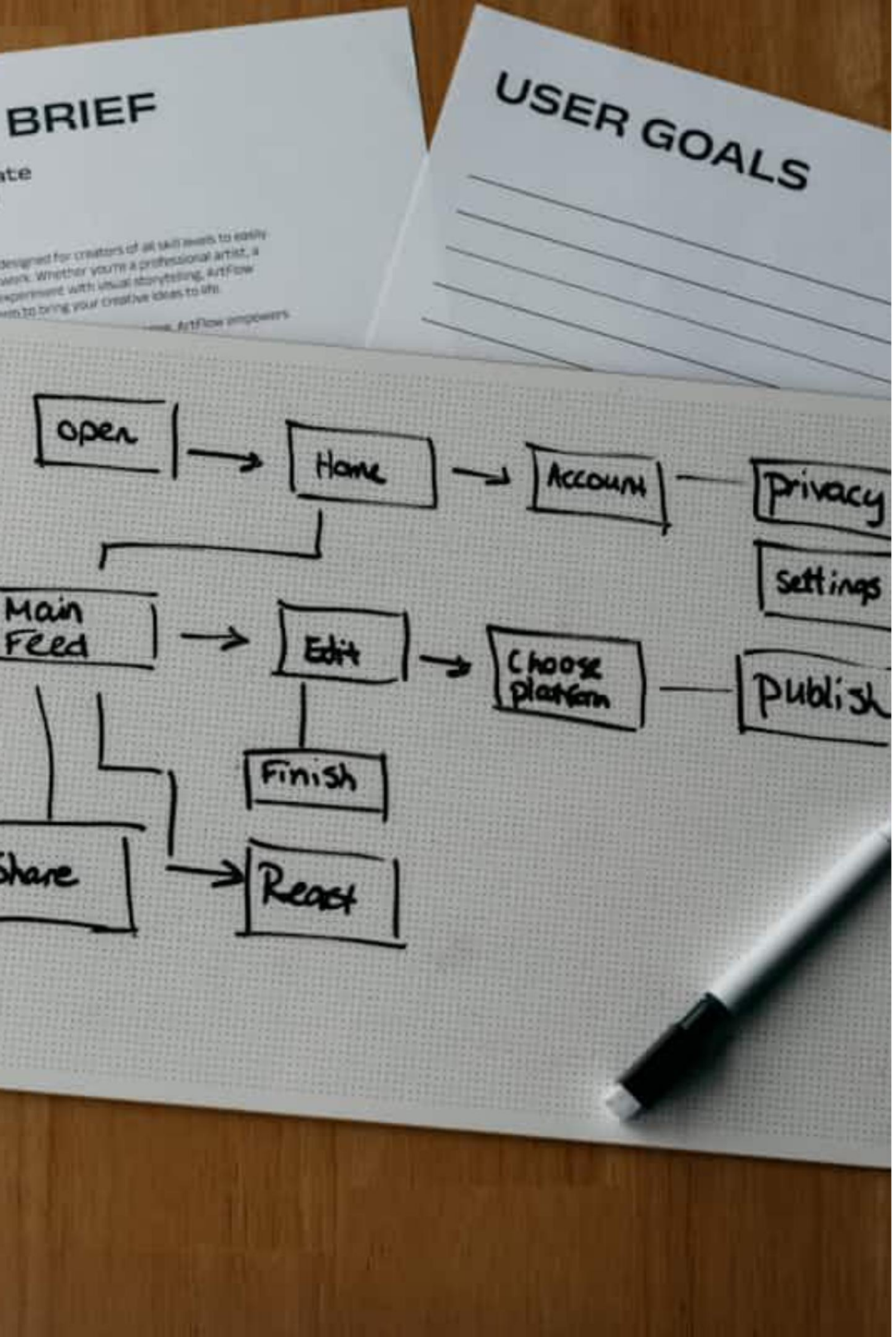
Beyond Structure: Getting Specific



Organizational charts show *authority*, but they don't show who handles specific tasks. **Responsibility Assignment Matrix (RAM)**.

- Links project work packages directly to team members
- Ensures every task has a clear owner
- Eliminates confusion and prevents work from falling through cracks

The most effective type of RAM is the **RACI chart**.



Resource Assignment: Who Does What?

Once you have the team, clarify everyone's roles using the powerful Responsibility Assignment Matrix (RAM).

R - Responsible

The person who *does* the work

Example: The developer writing the code

A - Accountable

The person who *owns* the work and has final say

Rule: Only **one** 'A' per task

C - Consulted

The person providing input or expertise

Example: The subject matter expert or senior architect

I - Informed

The person kept up-to-date on progress

Example: Client stakeholders or department heads

The RACI Chart Explained

Defining Roles with Precision

Responsible (R)

The person who *does* the work. The hands-on contributor who executes the task.

Accountable (A)

The person who *owns* the work. Ultimate authority and answerability for success.

Consulted (C)

Experts whose input is needed. Involves two-way communication and feedback.

Informed (I)

Stakeholders kept updated on progress. One-way communication for awareness.

Golden Rule: There must be only **ONE 'A' (Accountable)** person per task to prevent confusion and finger-pointing.

RACI Chart in Action

Building a User Login Module

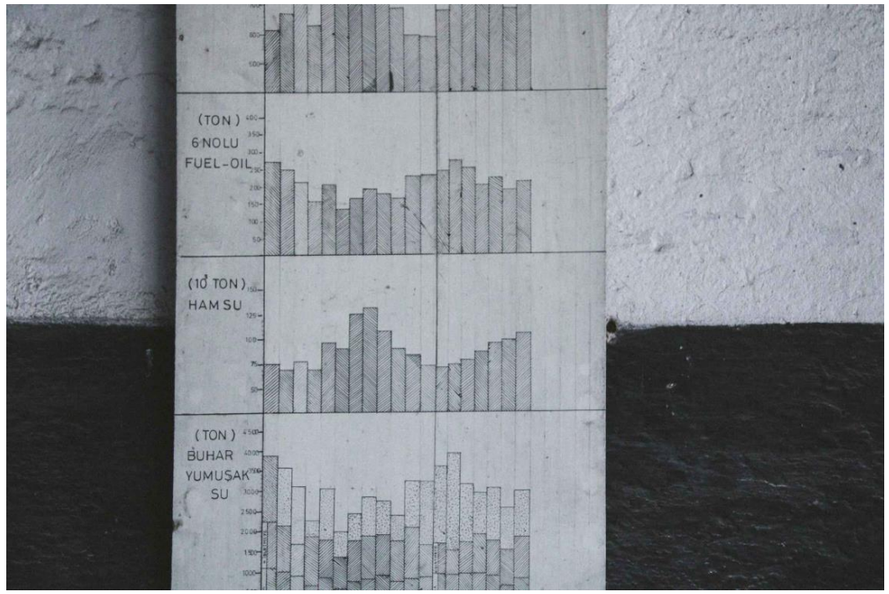
Let's see how RACI works in practice with a real software development scenario:

Activity/Task	Project Manager	Lead Developer	Junior Developer	QA Engineer	UI/UX Designer
Design UI Mockup	A	C	I	I	R
Develop Frontend Code	I	A	R	C	C
Develop Backend API	I	A	R	C	I
Test the Module	A	C	I	R	I
Deploy to Staging	A	R	I	C	I

Notice how each task has exactly **one Accountable (A)** person while multiple people may be Responsible (R), Consulted (C), or Informed (I).

Resource Loading: Visualizing Workload

Resource Loading shows the amount of work assigned to each team member over time. It's your early warning system for burnout and bottlenecks.



Key Benefits

- Identify over-utilized team members before burnout hits
- Spot capacity gaps and underutilized resources
- Make data-driven staffing decisions
- Prevent project delays caused by resource conflicts

The primary visualization tool is a **Resource Histogram** - a bar chart showing scheduled work time over specific periods.

Review resource loading weekly during active development phases to catch over-allocations early and maintain team productivity.

Resource Leveling: Fixing Over-allocations

Resource Leveling resolves conflicts by adjusting schedules to maintain manageable, consistent resource usage.

Resource Smoothing

Uses **free float** or **total float** in non-critical tasks to delay them without impacting project end date.

Advantage: Preserves critical path and deadline

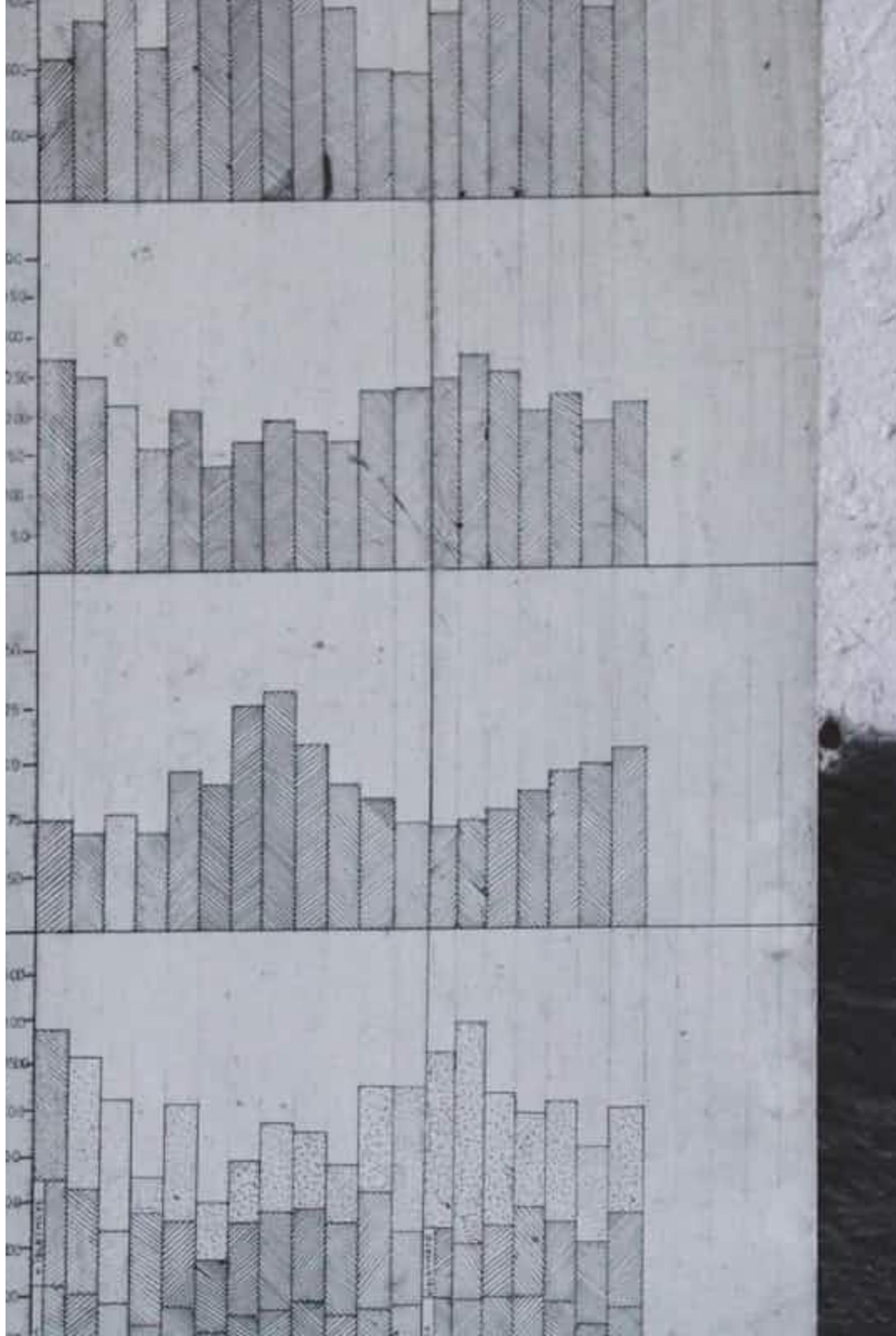
Limitation: Only works when slack is available

Resource-Constrained Scheduling

When no slack exists, extend the project timeline to resolve over-allocation conflicts.

Trade-off: Prioritizes resource health over schedule constraints

Result: Later delivery but sustainable team performance



Team Structures: Organization Impacts Everything

How your team is organized directly affects communication flow, decision-making speed, and project authority.

Structure	Description	Pros	Cons
Functional	Members report to functional manager. PM has low authority.	Deep expertise, clear career path	Low project loyalty, slower decisions
Project-Oriented	Members report directly to PM. High PM authority.	High team loyalty, fast communication	No "home" post-project, skill duplication
Matrix	Hybrid: dual reporting to functional and project managers	Flexible, efficient resource use	"Two bosses" problem, potential conflicts



Matrix Variations

Weak Matrix: Functional manager dominates
Balanced Matrix: Equal power sharing
Strong Matrix: PM has more authority

Project Communication Planning

Effective communication is project lifeblood. A **Communication Management Plan** answers the critical questions:

What information needs communicating?

Status updates, risk alerts, meeting minutes, milestone achievements, and blocker notifications

Who needs the information?

Stakeholders, team members, clients, executives, and functional managers

When will it be communicated?

Daily stand-ups, weekly reports, monthly reviews, and milestone gates

How will it be delivered?

Email, Slack, formal meetings, dashboards, and project management tools

Who owns each communication?

Project Manager, Scrum Master, Tech Lead, or designated team members

Performance Reporting & Next Steps

Essential Report Types

- **Status Report**

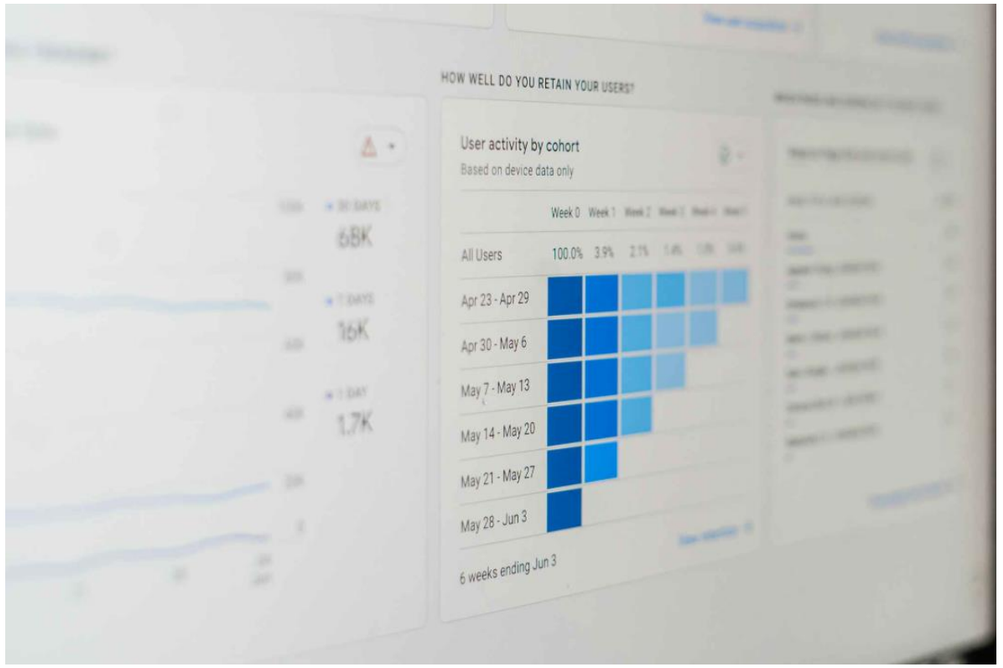
  Current project position: "Login module is 80% complete"

- **Progress Report**

  Accomplishments over period: "Completed authentication and password reset"

- **Forecast Report**

  Future predictions: "Forecasting one-week delay based on current velocity"



Key Performance Metrics

SPI

Schedule Performance

SPI > 1 = Ahead of schedule

CPI

Cost Performance

CPI > 1 = Under budget

Your Action Items

- **Team Acquisition:** Secure the right people with right skills
- **Resource Management:** Use RACI for clarity, histograms for load visualization
- **Structure Selection:** Choose organization model that fits your context
- **Communication:** Plan it deliberately - it's not optional!
- **Reporting:** Use metrics to keep stakeholders informed

Summary & Key Takeaways

HR Planning

Define roles and responsibilities needed for project success

Org Charts

Visualize team structure and reporting relationships

RACI Matrix

Assign specific ownership and accountability for every task

3x

Reduced Conflicts

Clear roles eliminate confusion and territorial disputes

40%

Faster Delivery

Proper planning reduces rework and miscommunication

95%

Team Satisfaction

Everyone knows their role and contribution to success

The time invested in proper HR planning pays massive dividends. It creates clarity, establishes accountability, and dramatically reduces the confusion that derails software projects.

