Project Management

CSE 4407

Md. Bakhtiar Hasan

Assistant Professor
Department of Computer Science and Engineering
Islamic University of Technology

May 29, 2025

The Spark: Origins of Systems Projects

- Project Triggers
  - Problems: Things aren't working as they should. Performance gaps, inefficiencies, errors
  - Opportunities: Chance to improve, upgrade, or innovate. New tech, changing markets, e-commerce potential
- Driving Forces
  - Adapting to organizational change (growth, new strategies)
  - Responding to external shifts (legal, industry, e-commerce trends)
- Key Takeaway: Many ideas are suggested, but only some become formal projects after evaluation

Spotting Troubles: Signs of Underlying Problem

- Where to Look: How do we *detect* these problems?
  - Check Output Against Performance Criteria
    - Too many errors?
    - Work completed too slowly?
    - Work done incorrectly?
    - Work done incompletely?
    - Work not done at all?
  - Observe Employee Behavior
    - Unusually high absenteeism?
    - High job dissatisfaction (complaints, low morale)?
    - High employee turnover (People leaving frequently)?
  - Listen to External Feedback
    - From: Vendors, Customers, Suppliers, Service Providers
    - Via: Complaints, Suggestions for improvement, Loss of sales, Lower-than-expected sales

Okay, We See Symptoms... Now What? Define the Problem!

- Why: It's the critical first step in any structured approach (like SDLC or O-O). Sets the foundation for the entire project
- (Analogy: Like a doctor needing a clear diagnosis before prescribing treatment)
- Components of a Problem Definition
  - Problem Statement: Brief summary (1-2 paragraphs)
  - Issues: The specific, independent pieces of the problem (The "Pain Points") → *Current State*
  - Objectives: What needs to be achieved to address the issue (The "Gain") → *Desired State*
  - Requirements: Specific things the system *must* do (Functionality, security, usability, etc.)
  - Constraints: Limitations or boundaries (Budget, deadlines, technology restriction - often include "not")

Uncovering the Real Issues: Analyst Detective Work

- How Analysts Identify Issues (During info gathering - interviews, observation, etc.)
  - Repetition: Same topic/theme mentioned multiple times, by different people
  - Metaphors: Users describe the business in consistent ways (“it’s a battle,” “we’re family,” “it’s a well-oiled machine...or not!”)
  - Storytelling: Users narrate problems with a beginning, middle, end, obstacles, heroes
  - Air Time: User spends significant time talking about specific topic
  - Direct Statements: “Listen, THIS is a major problem!”
  - Emphasis: Body language (leaning in, pointing) or vocal tone shows importance
  - Primary: It’s the very first thing the user brings up

Turning Problems into Goals: Setting Objectives

- Linking Issues and Objectives
  - Objectives should directly address the identified Issues (point-by-point if possible)
  - Example
    - Issue: Work completed too slowly
    - Objective: Reduce process completion time by 25%
- Clarification: May need follow-up interviews to make objectives specific and measurable
- Prioritization: Determine the *relative importance* of objectives
  - Why: Limited resources (time, money)
  - Who decides: Users! They are the domain experts
- Validation: Analyst should try to witness the problem firsthand if possible

Case Study: Catherine's Catering Conundrum

- The Business: Small catering company (meals, receptions, banquets). Started small, good reputation led to growth
- Growth Pains
  - Using spreadsheets/word processing became inefficient
  - Handling routine phone calls (menu info, dietary options) was time-consuming
  - Managing last-minute changes (guest counts) was difficult
  - Scheduling growing number of part-time staff led to conflicts and understaffing
  - Ordering supplies per-event was inefficient (missed bulk discounts)
  - Failure in identifying overall trends
- The Decision: Hired IT/business consultants to help

Demo: Problem Definition - Catherine's Catering

Catherine's Catering: Translating Objectives to User Requirements

- From Objectives to Action: User requirements specify *how* objectives will be met from a user perspective
- Catherine's User Requirements
  - (Web System) Dynamic website for viewing products/pricing
  - (Web System) Allow clients to submit catering requests online → routed to manager
  - (Update Guests) Add clients to DB, assign UserID/Password
  - (Update Guests) Client website area to view/update guest counts (with 5-day cutoff)
  - (Key Personnel Change) Software to communicate directly with event facilities
  - (Part-Time Scheduling) HR system (buy or build) for scheduling part-timers (with constraints)
  - (Summary Reports) Provide queries/reports for summary info
- Next Step: These requirements drive the creation/modification of Use Cases or Data Flow Diagram (more on these later!)

Catherine’s Catering: Thinking Ahead - Testing

- Why
    - Helps ensure requirements are *clear* and *testable*
    - Starts early, evolves over time
- Catherine’s Preliminary Test Plan
    - Test viewing all product types
    - Test submitting requests (valid and invalid data) → correct routing
    - Test adding clients (validation, correct credentials)
    - Test client event viewing and update functionality (including the 5-day rule)
    - Test HR system (adding employees, scheduling logic, constraints)
    - Test reports queries for accuracy

Not All Projects Are Created Equal: Selection Criteria

- Reality Check: Organizations have limited resources (time, money, people)
- Beware
  - Project proposed *only* for political gain or personal power (Likely ill-conceived, poor adoption)
  - Ignoring the **Systems Perspective** (Chapter 2): How does this project impact the *whole* organization? (*Interdependencies!*)
- Goal: Select projects that provide genuine value and align with the organization's direction

Project Go/No-Go: The Big Questions (1/2)

- Five Key Criteria for Selection
  1. Backing from Management?
     - Essential! Need endorsement from those controlling the budget/resources
     - Does not mean others are not involved, but top-level support is vital
  2. Appropriate Timing?
     - Is the organization *ready* for this change now? (Capacity, other initiatives)
     - Can the systems team/analyst commit the necessary time?
  3. Improves Strategic Organizational Goals?
     - Does it align with the big picture? E.g., Improve profits? Support competitive strategy? Enhance vendor/partner cooperation? Improve internal operations (efficiency)? Improve internal decision-making? Improve customer service? Boost employee morale?

Project Go/No-Go: Practicality and Worth (2/2)

- Five Key Criteria for Selection
  4. Practical in Terms of Resources?
     - Do *we* (analysts/dev team) have the necessary skills and tools?
     - Does the *organization* have the capacity (staff, infrastructure)?
     - Recognize limitations – some projects might require external expertise
  5. Worthwhile Compared to Other Options?
     - Is this the *best* use of limited resources right now? (Opportunity Cost)
     - Compare against other potential projects or improvements, such as Speeding up processes, Streamlining (removing steps), Combining processes, Reducing input errors (better forms/screens), Moving systems to the cloud, Reducing redundant storage/output, Improving system integration, etc.

Selected the Project... But Is It Doable?

- Next Question: Just because it's a *good idea*, doesn't mean it's *possible* right now
- Enter the *Feasibility Study*
  - A preliminary assessment, *not* a full systems study (yet!)
  - Gathers broad data for management
  - Goal: Decide whether to commit to a *full* systems study
- Analogy: Checking if you have the time, budget, and necessary items *before* planning a trip

Can We *Really* Do This? The T.E.O. Test

- **T**echnical Feasibility: Do we have the tech and skills?
  - Add on to present system
  - Technology available to meet users’ needs
- **E**conomic Feasibility: Does it make financial sense?
  - Systems analysts’ time
  - Cost of systems study
  - Cost of employees’ time for study
  - Estimated cost of hardware
  - Cost of packaged software or software development
- **O**perational Feasibility: Will people actually use it effectively?
  - Whether the system will operate when put in service
  - Whether the system will be used

Feasibility Deep Dive: Technical Check

- Key Questions
  - Can we enhance/upgrade the *current* system?
  - If not, does the required *technology* exist? (Is it proven?)
  - Do we have the *in-house* expertise (developers, testers, specialists)?
    - If not, can we realistically hire or outsource?
  - Are *packaged solutions* (off-the-shelf software) available?
    - If yes, how much *customization* is needed? (Heavy customization adds risk and cost)
- Focus: Availability and capability of technology and skills

Feasibility Deep Dive: Economic Check

- Key Questions
    - Do the expected *benefits* outweigh the *costs*?
    - Costs to Consider
        - Analyst and team time (salary/consulting fees)
        - Cost of the full systems study (including time from business users involved)
        - Business employee time (training, transition)
        - Hardware (servers, workstations, network gear)
        - Software (licenses for packaged software, development tools)
        - Custom software development costs
    - Value Proposition: Can the organization *see* the value?
        - Are long-term gains $>$ short-term costs?
        - Is there an immediate reduction in operating costs?
- If Not Economically Viable: Stop the project

Feasibility Deep Dive: Operational Check

- Assumes: Tech is possible, Economics make sense
- Key Questions
  - Will the system *operate correctly* within the organization’s environment?
  - Will people *actually use* the system once it’s deployed?
  - How will it impact workflows and processes?
- Watch Out For
  - Strong user resistance to change (happy with the old system)
  - Lack of user involvement in requesting the system
  - Poor user interface design (Covered later in Ch. 14)
- Positive Signs
  - User themselves requested the change
  - Users see clear benefits (efficiency, accessibility, reliability)
- Focus: Human resources, organizational culture, process integration

Supporting Feasibility: Estimating Workloads

- Why?
  - Essential for assessing Technical (Hardware needs) and Economic (Processing costs) feasibility
  - Ensures new hardware can handle *current* AND *future* demands (avoids costly early replacement due to growth)
- How?
  - Sample key tasks and measure resource usage (CPU, storage, network)
  - Project future growth based on business plans
  - Compare existing vs. proposed system performance
- Example Outcome
  - Proposed system significantly reduces human and computer time
  - Supports economic and operational feasibility arguments

Demo: Workload Comparison

Feasibility Passed! Now, What Gear Do We Need?

- Next Step: Dive deeper into the specifics of Hardware and Software requirements
    - Builds upon Technical Feasibility assessment
    - Involves inventory, estimation, and evaluation
- Goal: Make informed decisions about acquiring the right tools for the job

Demo: Steps in Choosing Hardware and Software

Taking Stock: The Hardware Inventory

- Why?
  - Can't make good decisions without knowing the starting point
  - Identifies usable existing hardware (potential for reuse/upgrades)
  - Reduces guesswork
- What to Record? (If no up-to-date inventory exists)
  - Type/Model/Manufacturer
  - Operational Status (Working? In Storage? Needs repair?)
  - Estimated Age and Projected Life
  - Physical Location
  - Responsible Department/Person
  - Financial Arrangement (Owned? Leased? Rented?)
- Link to Staffing: Helps assess if current staff skills match existing/needed hardware

Choosing the Right Tools: Evaluating Hardware

- Shared Responsibility: Management, Users, and Systems Analysts
  - Analyst oversees objectively
  - Analyst educates others on pros/cons
- Key Activities
  - Review vendor information/specs against requirements (from workload estimates)
  - Benchmarking: Simulate projected workloads on different hardware options (including existing systems) to compare performance
- Performance Criteria for Evaluation
  - Average transaction time (input to output)
  - Total volume capacity (throughput before issues)
  - CPU/Network idle time (efficiency)
  - Memory size
- Important: Define required/desired functions *before* vendor demos!

Own It or Rent It? Buy vs. Cloud Hardware

- Paths for Acquiring Hardware Infrastructure
  - Buy: Purchase servers, storage, networking gear
  - Cloud: Rent infrastructure from a provider (e.g., AWS, Azure, Google Cloud)

Feature	**Buy**	**Cloud**
Advantages	Full control over HW/SW Often cheaper in the long run (if chosen well) Tax advantages (depreciation)	Maintenance/upgrades by provider Agility: Change HW/SW rapidly Scalability: Grow/shrink easily Consistency across platforms No capital tied up/Lower initial cost
Disadvantages	High initial cost Risk of obsolescence Risk of being stuck with wrong choice Full responsibility of operation/maintenance	Company doesn't directly control own data Potential data security risks (provider trust) Reliability depends on Internet/Provider Proprietary APIs may hinder switching providers

- Hybrid Approach: Often organizations use a mix (some owned, some cloud)

Cloud Flavors: SaaS, PaaS, IaaS

- Beyond just Hardware
  - Infrastructure as a Service (IaaS) → Like renting the land and utilities
    - Renting the basic building blocks (compute, storage, network)
    - You manage OS, applications
  - Platform as a Service (PaaS) → Like renting a workshop with tools provided
    - Renting infrastructure *plus* operating systems, databases, development tools
    - You manage applications and data
  - Software as a Service (SaaS) → Like renting a fully furnished apartment
    - Renting ready-to-use software applications over the internet (e.g., Google Workspace, Salesforce, Office 365)
    - Provider manages everything
- Focus [1]
  - Primarily IaaS for hardware needs
  - SaaS later for software
  - PaaS is in between

Beyond the Box: Evaluating Vendor Support

- Key Areas to Evaluate: What happens *after* the sale?

Support Category	**Key Considerations**
Hardware Support	Full line offered? Quality? Warranty terms?
Software Support	Bundled OS/software? Custom Programming? Warranty terms?
Installation and Training	Commitment? In-house training? Technical help?
Maintenance	Routine/Preventive procedures? Emergency response time? Loaner equipment?
Cloud Services	Specific services offered (hosting, storage, etc.)? Uptime guarantees (SLAs)?
Disaster Recovery [2]	24/7 recovery options? Ransomware mitigation? Data center migration support?

- Don't Forget
  - Check vendor stability
  - Read the fine print (SLAs, contracts) → involve legal if needed!

Trend Watch: Bring Your Own Device (BYOD)

- What: Employees using their *personal* devices for work
- Why?
  - Potential for lower initial hardware costs for the organization
  - Can improve employee morale/convenience
  - Supports remote/flexible work
  - Leverages user familiarity with their own devices
- Analyst's Role
  - Observe what devices are *actually* being used
  - Design with popular platforms in mind (e.g., designing dashboards for iPads if executives use them)
- Major Drawback
  - SECURITY RISK! (Lost/stolen devices, malware, insecure Wi-Fi usage, unauthorized access)
  - Requires strong security policies and management tools (MDMs)

Now for Software: Build, Buy, or Rent?

- Paths for Acquiring Software
  - Create Custom Software: Develop it in-house or hire developers
  - Purchase COTS: Buy Commercial-Off-The-Shelf packages (e.g., Microsoft Office, SAP)
  - Use SaaS Provider: Subscribe to software delivered over the cloud (e.g., Salesforce, Google Workspace)
- Summary Trade-offs

Option	**Advantages**	**Disadvantages**
Custom Software	Customization, Innovation, In-house support, Ownership	High cost, Dev team, Maintenance
COTS Software	Low cost, Functionality, Reliability, Proven, Documentation	Generic, Rigid, Vendor risk, Common, Integration
SaaS Provider	No IT, Focus, Quick setup, Scalability, Auto-updates	Less control, Security, Provider risk, Lock-in, Missing features

- Reality: Many systems use a mix! (e.g., COTS for accounting, Custom for core process, SaaS for CRM)

Judging Software Quality and Support

- Objective Evaluation: Don’t rely solely on vendor claims your demos! Use *your* data, involve users
- Key Evaluation Criteria

Evaluation Category	**Key Considerations**
Perf. Effectiveness	Does it do *all required* tasks? Desired tasks? Good screen design? Handles load?
Perf. Efficiency	Fast response? Efficient input/output/storage/backup?
Ease of Use	Good UI? Help available? Flexible interface? Good feedback/error recovery?
Flexibility	Options for input/output? Integrates with other software
Quality of Docs	Well-organized? Online tutorials? FAQs?
Manufacturer Support	Tech support online? Newsletters/Emails? Website with updates?

- Remember: Vendors certify software works, but don’t guarantee it’s error-free in all situations or compatible with everything else you run. *Test thoroughly*!

Weighing the Scales: Costs vs. Benefits

- The Deciding Factor
  - While meeting requirements is key, the final ‘Go/No-Go’ often hinges on **Cost-Benefit Analysis**
  - Does the value justify the expense?
- Interdependence: Costs and Benefits must be considered *together*
- Forecasting is Key
  - To analyze costs/benefits over time, we need to *predict* key variables (e.g., future usage volume, labor costs, sales)
  - Relies on historical data or judgment methods (if no data)

Predicting the Future (Sort Of): Forecasting Basics

- Why: Needed for credible cost-benefit analysis over the system's life
- Methods Depend on Data
  - No Historical Data? Use Judgment Methods
    - Sales force estimates, Customer surveys, Delphi technique (expert consensus), Scenarios, Historical analogies
  - Historical Data Available? Use Quantitative Methods
    - Conditional: Find relationships (Regression, Leading indicators - more complex)
    - Unconditional: Identify patterns without needing causes (Simpler, cheaper) → Focus on graphical judgment, moving averages, time-series analysis
- Example: Moving Average → Average change of data series over time
  - Smooths out fluctuations to reveal trends
  - Calculates average over a fixed period (e.g., 3-month average predicts month 4)
  - Simple, but sensitive to extreme values

What's the Upside? Identifying Benefits

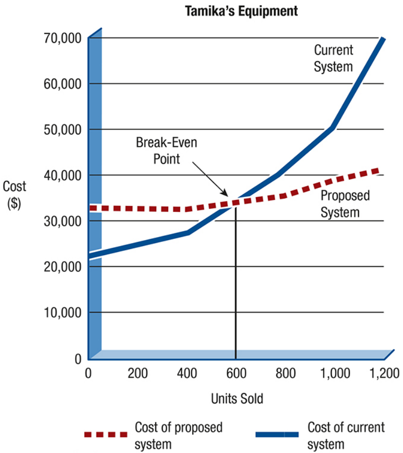
- Two Types of Benefits
  - Tangible Benefits: Measurable in dollars
    - Increased processing speed → Reduced labor time
    - Access to new information → Better decisions leading to profit/savings
    - Timelier information → Faster response to opportunities/threats
    - Superior calculation power → Complex analysis possible
    - Decreased employee time on tasks → Labor cost savings
    - Reduced errors → Lower correction costs
  - Intangible Benefits: Difficult to quantify, but still important!
    - Improved decision-making process
    - Enhanced data accuracy
    - Improved customer service
    - Reputation
    - Increased employee job satisfaction
- Crucial: Include BOTH tangible and intangible benefits in the proposal for a complete picture

What's the Downside? Identifying Costs

- Two Types of Costs (Parallel to Benefits)
  - Tangible Costs: Can be accurately projected and quantified
    - Hardware/Software purchase cost
    - Analyst and Developer time (salaries/fees)
    - Business user time (participation in study, training)
    - Operational costs (maintenance contracts, cloud subscription fees, electricity)
  - Intangible Costs: Difficult to estimate, but represent real risks
    - Losing competitive edge → If the system fails or is delayed
    - Damaged company image → Due to system errors/outages
    - Reduce employee morale → Due to difficulty/frustrating system
    - Ineffective decisions → Due to system providing poor/untimely information
- Crucial: Include BOTH tangible and intangible costs in the proposal for a balanced picture

Technique 1: Break-Even Analysis

- Purpose: Determines the point where a new system becomes more *cost-effective* than the current system based on volume
- Compares: Total Costs of Current System vs. Total Costs of Proposed System
  Total Costs = One-time Development/Setup Costs + Recurring Operational Costs
- Break-Even Point: The volume (e.g., units sold, transactions processed) at which the total costs of both systems are equal
- Beyond this point, the proposed system is cheaper *per unit of volume*



Technique 2: Payback Analysis (and Break-Even Limits)

- Break-Even Analysis Limitations
  - Primarily focuses on *costs*, assuming benefits remain constant (often not true!)
  - Doesn't explicitly show *when* the initial investment is recouped
- **Payback Analysis**
  - Purpose: Determines how long it takes for the accumulated *benefits* (especially tangible ones) of the new system to "pay back" the initial development and setup costs
- Combined View: Often, both are used alongside other financial metrics for a full economic picture

Year	Cost ($)	Cu. Costs ($)	Ben. ($)	Cu. Ben. ($)
0	30,000	30,000	0	0
1	1,000	31,000	12,000	12,000
2	2,000	33,000	12,000	24,000
3	2,000	35,000	8,000	32,000
4	3,000	38,000	8,000	40,000
5	4,000	42,000	10,000	50,000
6	4,000	46,000	15,000	65,000

From Big Idea to Done Deal: Managing Time and Activities

- Challenge: Systems projects, especially large ones, can get complex and unwieldy
- Project Management Goals
  - Complete project *on time*
  - Complete project *within budget*
  - Deliver all promised *features/functionality*
- First Step: Break the project down into smaller, manageable pieces

Divide and Conquer: The Work Breakdown Structure (WBS)

- What: A hierarchical decomposition of the total scope of work to be carried out by the project team
- Method: Decomposition - Start big, break it down into smaller pieces until tasks are manageable
- Properties of Good WBS Tasks
  - Single Deliverable: Each task produces one tangible outcome (a report, a coded module, a test plan)
  - Assignable: Can be assigned to one person or group
  - Accountable: Has a responsible person monitoring it
- Completeness
  - All tasks must add up to 100% of the project work
  - Tasks can vary in duration and team size

Ways to Structure the WBS

- Two Common Approaches
  - Product-Oriented WBS
    - Breaks down the work based on the *components* of the final product
    - Example: Website → Home Page, Product Pages, FAQ Page, Contact Page, E-commerce Module
    - Each component has sub-tasks
  - Process-Oriented WBS
    - Breaks down the work based on the *phases* or *processes* involved (e.g., SDLC phases)
    - Example: Website → Emphasizes on what to do in Initiation, Planning, Analysis, Design, and Launch
- Choice Depends On
  - Project type
  - How you want to manage/track progress
  - Process-oriented is common in SAD

Demo: Sample Process Oriented WBS

How Long Will It *Really* Take?

- Goal: Arrive at realistic estimates for each task in the WBS
- The Difficulty: Accurately estimating time for WBS tasks is challenging, but essential for scheduling and budgeting
- Five Common Techniques
  1. Relying on Experience
  2. Using Analogies
  3. Three-Point Estimation
  4. Function Point Analysis
  5. Using Time Estimation Software

Estimation Techniques: Drawing From The Past

- **Relying on Experience**
  - Best approach if you have done the same tasks before
  - Providers estimates based on real-world knowledge (including potential pitfalls)
  - Gives "most likely" and "pessimistic" estimates
- **Using Analogies**
  - Used when direct experience is lacking, but you've done *something* similar
  - Identify a past project (even unrelated) with comparable structure/complexity
  - Compare the WBS/Network diagrams of both projects
  - Base estimates for the new project on the known durations from the analogous one (adjusting for differences)

Estimation Techniques: Three-Point Method

- Concept
  - Combines optimistic, pessimistic, and most likely estimates to get a weighted average
  - Accounts for uncertainty
- Steps
  - Estimate a = Best-case scenario
  - Estimate b = Worst-case scenario time (disasters happen!)
  - Estimate m = Most-likely scenario time
  - Weighted Average: $E = \frac{a+4\times m+b}{6}$
- Example: Coding a module
  - Best: 8 days
  - Most likely: 10 days
  - Worst: 30 days
  - $E = \frac{8+4\times 10+30}{6}$ = 13 days
- Result: Providers a more realistic single estimate than just using ‘m’

Estimation Techniques: Specialized

- **Function Point Analysis (FPA)**
  - Estimates effort based on the system's *functional size* and complexity, NOT lines of code initially
  - Measures five components: External Inputs, External Outputs, External Queries, Internal Logical Files, External Interface Files
  - Complexity ratings are applied
  - Can be used to compare estimated effort across different Programming languages
  - Resource: International Function Point Users Group (IFPUG)
- **Using Time Estimation Software**
  - Based on models like COCOMO II or COSYSMO (e.g., SystemStar software)
  - Analyst inputs estimated system size (e.g., lines of code, function points) AND other factors (team experience, platform, required reliability, etc.)
  - Software calculates rough estimates for effort, duration, staffing
  - Estimates become more refined as the project progresses

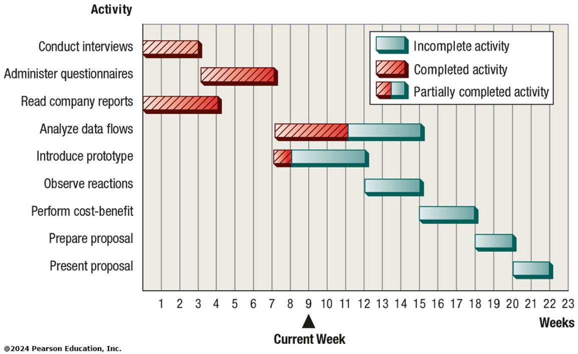
Scheduling the Work: Planning and Control

- Planning: Selecting the team, assigning tasks, estimating time, *creating the schedule*
- Control: Monitoring progress against the plan, using feedback, taking corrective action (expediting, rescheduling), motivating the team
- Foundation: A detailed WBS
- Key
    - Detail must be sufficient for scheduling and control
    - Time estimates are added

Phase	Activity	Detailed Activity	Wk.
Analysis	Data gathering	Conduct interviews	3
		Administer questionnaires	4
		Read company reports	4
		Introduce prototype	5
		Observe reactions to prototype	3
	Data flow and decision analysis	Analyze data flow	8
	Proposal preparation	Perform cost-benefit analysis	3
		Prepare proposal	2
		Present proposal	2
Design	Data entry design	-	-
	Input design	-	-
	Output design	-	-
	Data organization	-	-
Implementation	Implementation	-	-
	Evaluation	-	-

Visualizing the Time: Gantt Charts

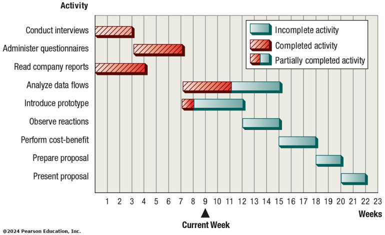
- What: A horizontal bar chart showing project tasks against a timeline
- Structure
  - Y-axis: List of project activities/tasks
  - X-axis: Time (days, weeks, months)
  - Bars: Represent tasks; length indicates estimated duration; position indicates start/end times
- Advantages: Simple, easy to create and understand, good for communicating progress visually



©2024 Pearson Education, Inc.

Gantt Chart Limits and A More Powerful Tool: PERT

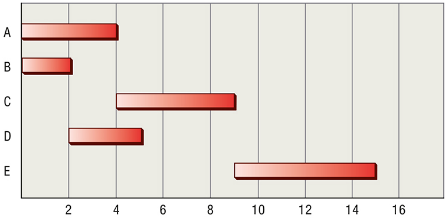
- Gantt Chart Weakness: Doesn't clearly show **dependencies** or **precedence**
  - Can't easily tell *why* a task starts after another one finishes. Is it required, or just coincidence?
  - Doesn't highlight which tasks are *critical* to the overall project duration
- Enter PERT: Program Evaluation and Review Technique
  - Developed for complex projects (US Navy Polaris)
  - A network diagram showing tasks and their dependencies
  - Excellent for projects where tasks can happen **in parallel**
  - Helps identify the **task precedence** and **critical path**

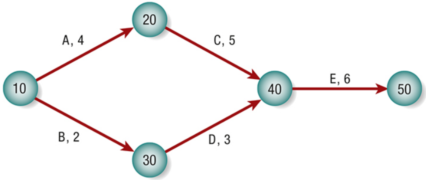


©2024 Pearson Education, Inc.

Visualizing Dependencies: Gantt vs. PERT

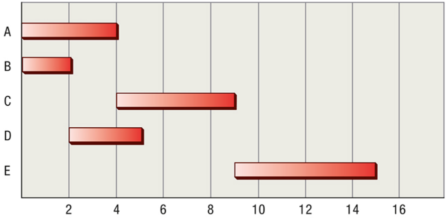
- Gantt Chart: Represents the tasks as bars against time
- PERT Diagram: Represents the same tasks as a network
    - Circle (Nodes/Events) → Start/end points of activities
    - Arrows (Activities) → Tasks labeled with name and duration
- Key Difference: PERT explicitly shows precedence
    - *C* can't start until *A* is done at node 20
    - *E* can't start until *both C* and *D* are done

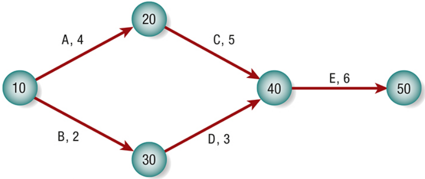




PERT Concepts: Critical Path

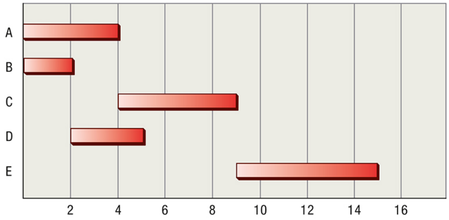
- Path: A sequence of connected activities from the start event to the end event
- Path Length: Sum of the durations of all activities on a path
- Critical Path: The **longest path** through the PERT network → Project Duration
  - Determines the **shortest possible completion time** for the entire project
  - **Any delay** on a critical path activity directly delays the project completion date
- Example
  - Path 1 (A-C-E): 10 → 20 → 40 → 50 **(15 weeks)**
  - Path 2 (B-D-E): 10 → 30 → 40 → 50 (11 weeks)

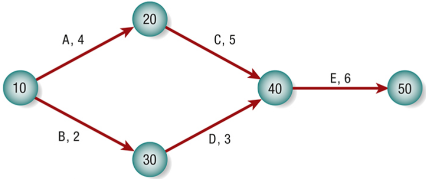




PERT Concepts: Slack Time

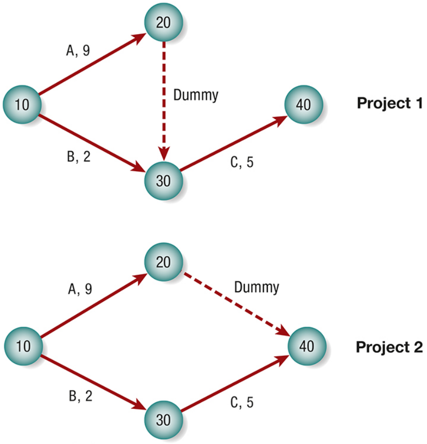
- The amount of time a task or path can be delayed *without* delaying the entire project
- Exists only on non-critical paths
- Slack Time for a Path = Project Duration - Non-critical Path Duration
- Example
  - Slack on Path 2 = Project Duration - Path 2 Duration = 4 weeks
  - Implies Tasks B or D could slip by a total of 4 weeks without impacting the project duration
  - Also implies Tasks A, C, and E have zero slack





PERT Concept: Dummy Activities

- What: Activities with ZERO duration, usually shown as dashed lines
- Purpose: Used to maintain correct logic and precedence relationships, especially when tasks share *some* but not *all* predecessors
- Key: Dummies clarify precedence when standard arrows alone would create incorrect logic



©2024 Pearson Education, Inc.

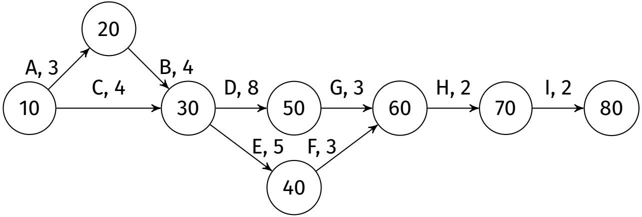
PERT Example: Data Gathering and Proposal

- Scenario: Scheduling the initial phases of a systems project

Activity		**Predecessor(s)**	**Duration (Weeks)**
A	Conduct interviews	None	3
B	Administer questionnaires	A	4
C	Read company reports	None	4
D	Analyze data flow	B, C	8
E	Introduce prototype	B, C	5
F	Observe reactions prototype	E	3
G	Perform cost-benefit analysis	D	3
H	Prepare proposal	F, G	2
I	Present proposal	H	2

PERT Example: Constructing the Network

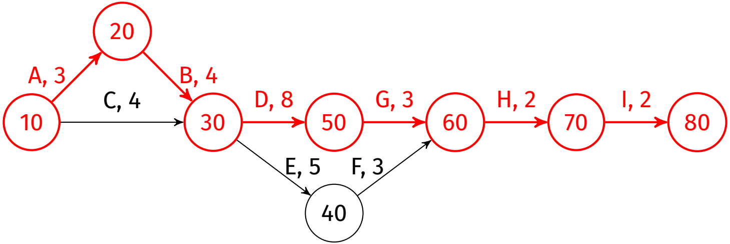
- Process: Start with activities having no predecessors. Add activities sequentially based on their listed predecessors. Ensure all dependencies are represented.



Activity		Predecessor(s)	Duration (Weeks)
A	Conduct interviews	None	3
B	Administer questionnaires	A	4
C	Read company reports	None	4
D	Analyze data flow	B, C	8
E	Introduce prototype	B, C	5
F	Observe reactions prototype	E	3
G	Perform cost-benefit analysis	D	3
H	Prepare proposal	F, G	2
I	Present proposal	H	2

PERT Example: Identifying the Critical Path

- Method: Calculate the total duration of *every possible path* from start (10) to end (80). The longest one is critical.



- Results
  - Path 1: 22 weeks, Path 2: 19 weeks, Path 3: 19 weeks, Path 4: 16 weeks
  - Critical Path = Path 1 (= Project Duration)
- Management Focus: Activities, *A*, *B*, *D*, *G*, *H*, and *I* must be carefully monitored. Activities *C*, *E*, and *F* have some slack.

Staying on Track: Controlling the Project

- Reality Check: Things go wrong! Scope changes, delays happen, costs fluctuate
- Project Control: The ongoing process of:
  - Monitoring actual progress vs. the plan (schedule and budget)
  - Using feedback to identify deviations
  - Taking corrective action (rescheduling, expediting, budget changes)
  - Keeping the team motivated and informed
- Key Control Areas: Cost, Risk, Time

Controlling Costs: Estimation

- Builds on WBS and Schedule: Need cost estimates for each activity
- Main Resource Cost: Project team time! (Also special equipment/tools)
- Cost Estimation Approaches: Similar to Time Estimation
  - Top-Down
    - Base estimates on similar past projects (experience driven)
    - Adjust for known differences
  - Bottom-Up
    - Get estimates from team members responsible for each WBS task
    - Analyst reviews/aggregates
    - Can be time-consuming, variable
  - Parametric Modeling
    - Use parameters/formulas (e.g., cost per line of code, cost per hour) + Project size estimates
    - Software like COCOMO II can assist
- Common Practice: Use a combination of methods

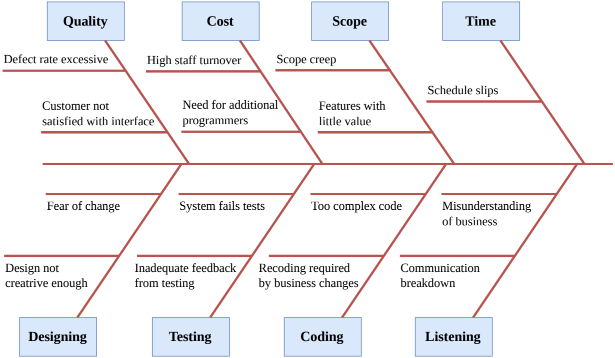
Controlling Costs: Pitfalls and Budgeting

- Why Cost Estimates Often Fail
  - Over-Optimism
    - Believing everything will go perfectly
    - Underestimating effort/complexity (e.g., lines of code)
    - “Happy path” estimating
  - Rushing: Spending too little time on estimation just to get to the real work
- Key: Be as accurate as possible, knowing estimates *will* be revised
- Preparing the Budget: A critical project deliverable!
  - Clients/Management need it early
  - Often uses standard organizational forms/templates
  - Details costs by category (Team time, HW, SW, Training, etc.)

Demo: Sample Budget

Managing Risk: Looking Out for Trouble

- Best Defense: Thorough initial analysis, feasibility studies, understanding motivations, experience!
- But Problems Happen: Need to anticipate and plan → Projects are not immune
- Common Project Failure Causes
  - Unrealistic deadlines
  - Myth: Adding more people always speeds things up
  - Reluctance to seek outside expertise
- Remember: Management has the final say, but team's reputation is linked to project success


Quality
Cost
Scope
Time
Defect rate excessive
High staff turnover
Scope creep
Schedule slips
Customer not satisfied with interface
Need for additional programmers
Features with little value
Fear of change
System fails tests
Too complex code
Misunderstanding of business
Design not creatrive enough
Inadequate feedback from testing
Recoding required by business changes
Communication breakdown
Designing
Testing
Coding
Listening

Need for Speed? Expediting Activities

- Crashing: Speeding up project activities to finish earlier → Costs extra!
    - Crash Time: Absolute minimum time an activity *can* take
    - Cost/Week: The additional cost incurred to reduce the activity duration by one week
- Why
    - Potential bonus for early completion
    - Free up resources/team members for other projects sooner
    - Recover from earlier delays

Activity	Estimated Duration	Crash Time	Cost/Week
A	3	1	$800
B	4	2	500
C	4	2	400
D	8	6	1,000
E	5	5	1,000
F	3	3	800
G	3	3	800
H	2	2	400
I	2	1	600

How to Expedite: The Analysis

- Rule #1: Only expedite critical-path activities
- Rule #2: Pick the cheapest critical-path activity per time saved
- Rule #3: Never expedite an activity below its minimum (crash) duration

Eligible Activities	Activity Chosen	Time for Each Path				Cost	Cu. Cost
		(22)	19	19	16		
A, B, D, or I	B	(21)	18	19	16	$500	$500
A, B, D, or I	B	(20)	17	19	16	500	1,000
A, D, or I	I	(19)	17	18	15	600	1,600
A or D	A	(18)	16	(18)	15	800	2,400
A and C, or D	D	(17)	16	(17)	15	1,000	3,400
A and C, or D	D	(16)	(16)	(16)	15	1,000	4,400
A and C	A and C	(15)	(15)	(15)	14	1,200	5,600

Project Duration: 22 weeks → 15 weeks

Cost: $5,600

Comprehensive Control: Earned Value Management (EVM)

- Purpose: Integrates project **scope** (work done), **schedule** (time), and **cost** ($$$) into a unified framework to measure performance and predict outcomes
- Requires: Updated budget and schedule baseline
- Key EVM Measures
  - Budget at Completion (BAC): The total planned budget for the whole project (or task)
  - Planned Value (PV): The budget cost of work *scheduled* to be completed by a certain point in time → Where *should* we be?
  - Actual Cost (AC): The actual amount of money spent to complete work by that same point in time → What *did* we spend?
  - Earned Value (EV): The value (in terms of the original budget) of the work *actually completed* by that point in time → What work *did* we get done?

$$EV = PV \times \text{Work done so far, } p\%$$

- PMBOK definition: $PV = BAC \times \text{Expected } p\%$ and $EV = BAC \times \text{Actual } p\%$

EVM in Action: Website Example

Project Budget: $18,000

At the End of	Stage	Estimated Cost	Cu. Estimate	Estimated Duration	Stage Completed	Actual Cost of Stage to Date	Actual Cost of Project to Date
Month 1	1	$6,000	$6,000	1 month	100%	$6,000	$6,000
Month 2	2	3,000	9,000	1 month	100%	3,000	9,000
Month 3	3	3,000	12,000	1 month	100%	3,000	12,000
Month 4	4	3,000	15,000	1 month	50%	5,000	17,000
Month 5	5	3,000	18,000	1 month	0%	Not yet begun	Not yet begun

- EVM at End of Month 4
  - $BAC = \$18,000$
  - $PV = \$15,000$ (Cumulative Estimated Cost)
  - $AC = \$17,000$
  - $EV =?$ (Needs to be calculated)

EVM Calculations: Where Do We Stand?

After 4 months,

- Calculate % Work done so far (p): $\frac{100\%+100\%+100\%+50\%}{100\%+100\%+100\%+100\%} = \frac{350}{400} = 0.875$
- Calculate Earned Value (EV): $PV \times p = \$15,000 \times 0.875 = \$13,126$
  (Meaning: The work *actually completed* is worth $13,125 based on the original budget)
- Calculate Variances
  - Cost Variance (CV): $EV - AC = \$13,125 - \$17,000 = -\$3,875$ (Negative = OVER budget)
  - Schedule Variance (SV): $EV - PV = \$13,125 - \$15,000 = -\$1,875$ (Negative = BEHIND schedule, expressed in $ terms)
- Calculate Performance Indices
  - Cost Performance Index (CPI): $EV/AC = \frac{\$13,125}{\$17,000} = 0.772$
    (Less than 1.0 = Post cost performance; getting $0.77 worth of work for every $1 spent)
  - Schedule Performance Index (SPI): $EV/PV = \frac{\$13,125}{\$15,000} = 0.875$
    (Less than 1.0 = Behind schedule; progressing at 87.5% of the planned rate)

EVM Forecasting: Where Are We Headed?

- Purpose: Use CPI to predict future costs
- Key Forecasting Metrics
  - Estimate TO Complete (ETC): How much *more* money is likely needed to finish the project from this point, assuming current performance continues?

$$ETC = (BAC - EV)/CPI$$

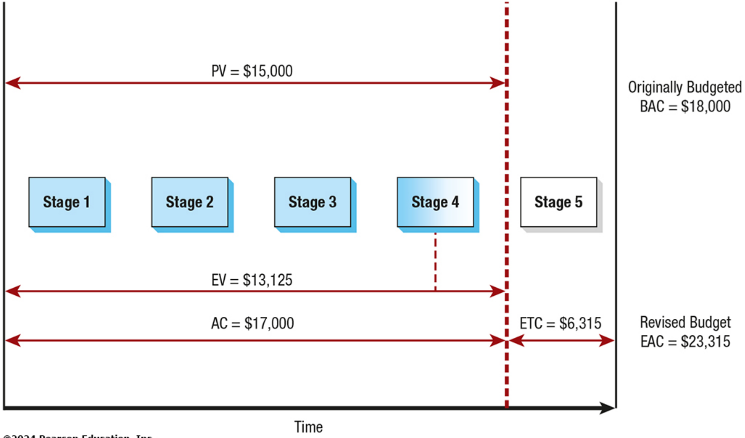
  - Estimated AT Completion (EAC): What is the *total revised estimated cost* for the entire project upon completion?

$$EAC = AC + ETC$$

- Website Example: $ETC = \frac{\$18{,}000 - \$13{,}125}{0.772} \approx \$6,315$ and $EAC = \$17,000 + \$6,315 = \$23,315$
- Result: Project likely to cost ~$23,315 instead of the planned $18,000

EVM Visualization: Key Takeaway

- Analyst's Role: Balance Cost, Time, and Scope based on this information



@2024 Pearson Education, Inc.

It’s About People: Managing the Project Team

- Equally Important: Managing the people doing the work
- Key Aspects
    - Assembling the right team
    - Fostering effective communication and dynamics
    - Setting achievable productivity goals
    - Motivating team members
    - Understanding unique project contexts (e.g., E-commerce)
    - Formalizing expectations (Project Charter)

Who’s On the Team? Assembling the Right Mix

- Core Values: Look for shared values (teamwork, quality, on-time/on-budget delivery)
- Desirable Characteristics
    - Good work ethic, Honesty, Competence
    - Willingness to lead based on expertise
    - Motivation and Enthusiasm for the *project*
    - Trustworthiness and Ability to trust teammates
- Diversity and Inclusion Matters
    - Why: Diverse teams outperform individuals, make faster/better decisions, solve problems faster (cognitive diversity) [3], [4]
    - How: Fair hiring, pay equity, support for individual success, ensuring every voice is heard [5]

The Right Skills for the Job: Team Composition

- Essential Roles/Skills
  - Business Knowledge: At least one person who deeply understands the business area/domain (e.g., Marketing expert for e-commerce site)
  - System Analysts: Ideally two or more (support, peer review, workload sharing)
  - Programming Skills: Obvious need for developers
  - Quality Assurance: People skilled in walkthroughs, reviews, testing
  - Documentation Skills: Ability to clearly document the system
  - Mix of Perspectives: Both “big picture” thinkers and detail-oriented individuals
- Also Valuable
  - Experience: Especially for time/cost estimation and avoiding pitfalls (Experienced devs can be much faster)
  - Enthusiasm and Imagination: Drives innovation and problem-solving
  - Communication Skills: Strong writers/speakers for proposals, user interaction, etc.
  - Usability Expert: Focuses on making the system user-friendly

Making the Team Tick: Communication and Dynamics

- Team Personality: Each team develops a unique interaction style
- Balancing Act: Teams consistently balance
  - Task: Getting the work done
  - Relationship: Keeping the team functioning smoothly socially
- Dual Leadership Roles: Often emerge
  - Task Leader: Focuses on achieving goals, assigning work, monitoring progress
  - Socioemotional Leader: Focuses on team morale, resolving conflicts, maintaining harmony
- Managing Tension
  - Ignoring tension leads to dysfunction
  - Open communication and feedback are key to resolving issues arising from the task/relationship balance

Unwritten Rules: Understanding Team Norms

- What: Collective expectations, values, and standard ways of behaving within a *specific* team (Can be explicit or implicit)
- Norms are Contextual: They belong to the team, don't automatically transfer, and change over time
- Functional vs. Dysfunctional
    - Functional Norms: Help the team achieve its goals (e.g., "We always test code before check-in," "We openly discuss disagreements respectfully")
    - Dysfunctional Norms: Hinder the team's progress (e.g., "Only senior members speak in meetings," "We avoid conflict at all costs," "Junior members do all the scheduling")
- Action: Teams need to make norms explicit and periodically assess if they are helping or hindering. *Change should be the norm!*

Aiming High: Setting Goals and Motivating the Team

- Setting Productivity Goals
  - Based on team expertise, past performance, project nature
  - Goals should be challenging, but achievable
  - Team participation in goal-setting increases buy-in
- Motivation Factors
  - Basic Needs: Salary, job security (met by being employed)
  - Higher-Level Needs: Affiliation (belonging), Control (influence), Independence (autonomy), Creativity. Projects can help fulfill these
- How Goal Setting Motivates
  - Clarity: Team knows exactly what is expected
  - Achievement Focus: Goals act as targets, creating focus
  - Autonomy: Often defines the "what", allowing team members' flexibility in the "how"
  - Feedback: Performances measured against clear goals simplifies reviews

Unique Challenges: Managing E-commerce Projects

- Key Differences
  - Scattered Data
    - Cross-department (Marking, Sales, Inventory, Finance)
    - Increases complexity and dept. politics
  - Diverse Teams
    - Skills: Dev, Marketing, DB, Security, Integration
    - External partners; fluid teams
  - Integration Focus
    - Front-end ↔ Inventory, Billing, Shipping
    - Often the hardest part
  - Heightened Security
    - System directly exposed to internet
    - Requires dedicated standalone project
- Management Tips
  - Align goals and foster cross-department integration
  - Engage partners early; keep communication clear

Getting It In Writing: The Project Charter

- Purpose
  - A written document clarifying project scope, objectives, and expectations
  - Acts as a contract between the team, users, and management
- Key Points
  - User Expectations/Project **Objectives** → What will it do?
  - Project **Scope** (Boundaries) → What's *in*, what's *out*?
  - **Analysis Methods** to be used
  - Key **Participants** and Time Commitment
  - Project **Deliverables** (Specific outputs)
  - **Evaluation** Criteria and Process → Who evaluates?
  - Estimated **Timeline** and Reporting Frequency
  - **Training** Plan → Who trains whom?
  - **Maintenance** Plan → Who supports it post-launch?
- Result: Shared understanding, reduced ambiguity, clear definition of 'done'

The Grand Finale (of Analysis): The Systems Proposal

- What: A formal written document detailing
  - systems study
  - findings
  - alternatives, and
  - **recommendations**
- Purpose:
  - Goes beyond the initial Project Charter
  - Provides justification for the recommended course of action
  - Serves as a key decision-making tool for stakeholders
- Audience: Management, IT Task Force, Key Users

What Goes Inside? The 10 Proposal Sections

- Standard Structure: Preliminary Materials

  1. Cover Letter
     - Friendly intro
     - Study objectives
     - Team members
  2. Title Page
     - Project name
     - Team names
     - Submission date
  3. Table of Contents
     - For longer proposals
     - Usually $>$ 10 pages
  4. Executive Summary
     - The “TL;DR” (Who, What, When, Where, Why, How)
     - Recommendations and Desired Action (250-375 words)
     - Write LAST

What Goes Inside? The 10 Proposal Sections

- Standard Structure: Study and Analysis

5. Outlines of Systems Study
   - Methods used (interviews, surveys, observation, etc.)
   - Who/what was studied
6. Detailed Results
   - Findings about system/human needs
   - Problems identified
   - Opportunities discovered
7. Systems Alternatives
   - 2-3 possible solutions
   - (including keeping the current system!)
   - Describe costs, benefits, pros/cons, implementation steps for each
8. Systems Analysts' Recommendations
   - The team's chosen solution and *why*
   - Must flow logically from alternatives analysis

What Goes Inside? The 10 Proposal Sections

- Standard Structure: Conclusion and Support
  9. Proposal Summary
     - Brief recap mirroring Executive Summary (objectives, recommendation, importance)
     - Positive conclusion
  10. Appendices
      - Supporting info (detailed data, charts, correspondence, etc.)

Getting the Word Out: Delivery and Presentation

- Distribution
  - Carefully select recipients (key decision-makers)
  - Hand-deliver copies if possible (increases visibility)
- Oral Presentation
  - Schedule a dedicated meeting
  - Prepare a separate presentation — DO NOT just read the report!
  - Focus on **highlights**, key findings, alternatives, and recommendation
  - Keep it brief (30-40 minutes max)
  - Allow ample time for **Questions** and **Discussion**
  - Be dynamic, engaging, and interactive

A Picture is Worth... Supporting Your Words with Figures

- Why?
  - People absorb information differently; visuals help
  - Demonstrate responsiveness to audience needs
  - Capture and communicate complex data effectively
- Rule: Figures **supplement** the text; they don't replace it
  - Always **interpret** figures in your written explanation
  - Don't make the reader guess the takeaway message
  - Number and title all figures sequentially

Organizing Data: Effective Use of Tables

- Purpose: Present statistical or alphabetical data in an organized, structured way
- Guidelines
  - Integrate: Place tables within the relevant text body, not just appendices (unless very large/supplementary)
  - Fit: Try to keep a table on a single page
  - Number and Title: Place clearly at the top; title should be meaningful
  - Label: Clearly label all rows and columns
  - Format: Boxed tables with vertical lines improve readability
  - Footnotes: Use for explanations or source information if needed
- Proposal Examples
  - Cost-benefit comparison tables
  - Break-even/Payback data
  - Hardware/Software option comparisons

Visualizing Trends and Comparisons: Effective Graphs

- Purpose: Illustrate comparisons (Line, Column, Bar) or composition (Pie, Area)
- Guidelines
  - Choose Appropriately: Select graph type that best suits the data and the message (e.g., line for trends over time, pie for percentages of a whole)
  - Integrate: Place graphs within the relevant text body
  - Number and Title: Place clearly; title should be meaningful
  - Label Everything: Axes (with units!), lines, bars, pie slices
  - Key/Legend: Clearly explain colors, shading, or symbols used
- Proposal Examples
  - Break-even analysis graph
  - Payback period visualization
  - Comparison of performance benchmarks
  - User satisfaction ratings

References I

[1] D. Cearley and D. Reeves, "Cloud computing innovation key initiative overview," Gartner, Inc., Stamford, CT, Tech. Rep. 2718918, 2011. [Online]. Available: https://www.gartner.com/en/documents/2718918 (cit. on p. 23).

[2] R. Blair and J. Hewitt, "Market guide for disaster recovery as a service," Gartner, Inc., Stamford, CT, Tech. Rep. Document No. 4364899, 2024. [Online]. Available: https://www.gartner.com/en/documents/4364899 (cit. on p. 24).

[3] PM Editorial, "Diversity drives better decisions," (2017), [Online]. Available: https://www.peoplemanagement.co.uk/article/1742040/diversity-drives-better-decisions (cit. on p. 63).

[4] A. Reynolds and D. Lewis, "Teams solve problems faster when they're more cognitively diverse," (2017), [Online]. Available: https://hbr.org/2017/03/teams-solve-problems-faster-when-theyre-more-cognitively-diverse (cit. on p. 63).

[5] M. Alexander, "How to build more diverse and inclusive project teams," (2021), [Online]. Available: https://www.techrepublic.com/article/how-to-build-more-diverse-and-inclusive-project-teams/ (cit. on p. 63).