Week 14: Project Management and Risk Analysis

YZM2021 - Software Engineering Principles

Instructor: Ekrem Çetinkaya
Date: 30.12.2025

Today's Learning Outcomes

By the end of this lecture, you will be able to:

1. **Explain** the key activities involved in software project management.
2. **Apply** different estimation techniques (Expert Judgment, COCOMO, Story Points).
3. **Create** project schedules using Work Breakdown Structures and Gantt charts.
4. **Identify** and categorize project risks using systematic techniques.
5. **Analyze** risks using probability-impact matrices and quantitative methods.
6. **Develop** risk mitigation, avoidance, and contingency strategies.
7. **Understand** the principles of Agile project management and velocity tracking.
8. **Apply** Earned Value Management to track project progress.

Software Project Management

What is Software Project Management?

> **Project Management** is the discipline of planning, organizing, and managing resources to bring about the successful completion of specific project goals and objectives.

Success Criteria (Sommerville)

1. **Deliver software on time** to the customer
2. **Keep costs within budget**
3. **Meet customer expectations** for functionality
4. **Maintain a well-functioning team**

> Good management cannot guarantee success, but **bad management usually results in failure**.

Why Software PM is Uniquely Challenging

Software engineering differs from other engineering disciplines:

1. The Product is Intangible

- You cannot see software being built like a bridge
- Progress is invisible—managers rely on reports, not observation
- Difficult to measure completion percentage

2. Large Projects are Often One-Off

- Every project is different in some way
- Past experience may not transfer
- **Rapid technology changes** make experience obsolete

3. Processes are Variable and Organization-Specific

- No universal *correct* software process
- What works in one organization may fail in another
- Difficult to predict when a process will lead to problems

Why Software PM is Uniquely Challenging

4. AI is Changing Everything

- **Vibe Coding** makes traditional estimation models unreliable
- A junior dev + AI can sometimes outperform a senior dev
- Progress can be explosive… then stall completely
- There is now *agent sync* issue along with human sync

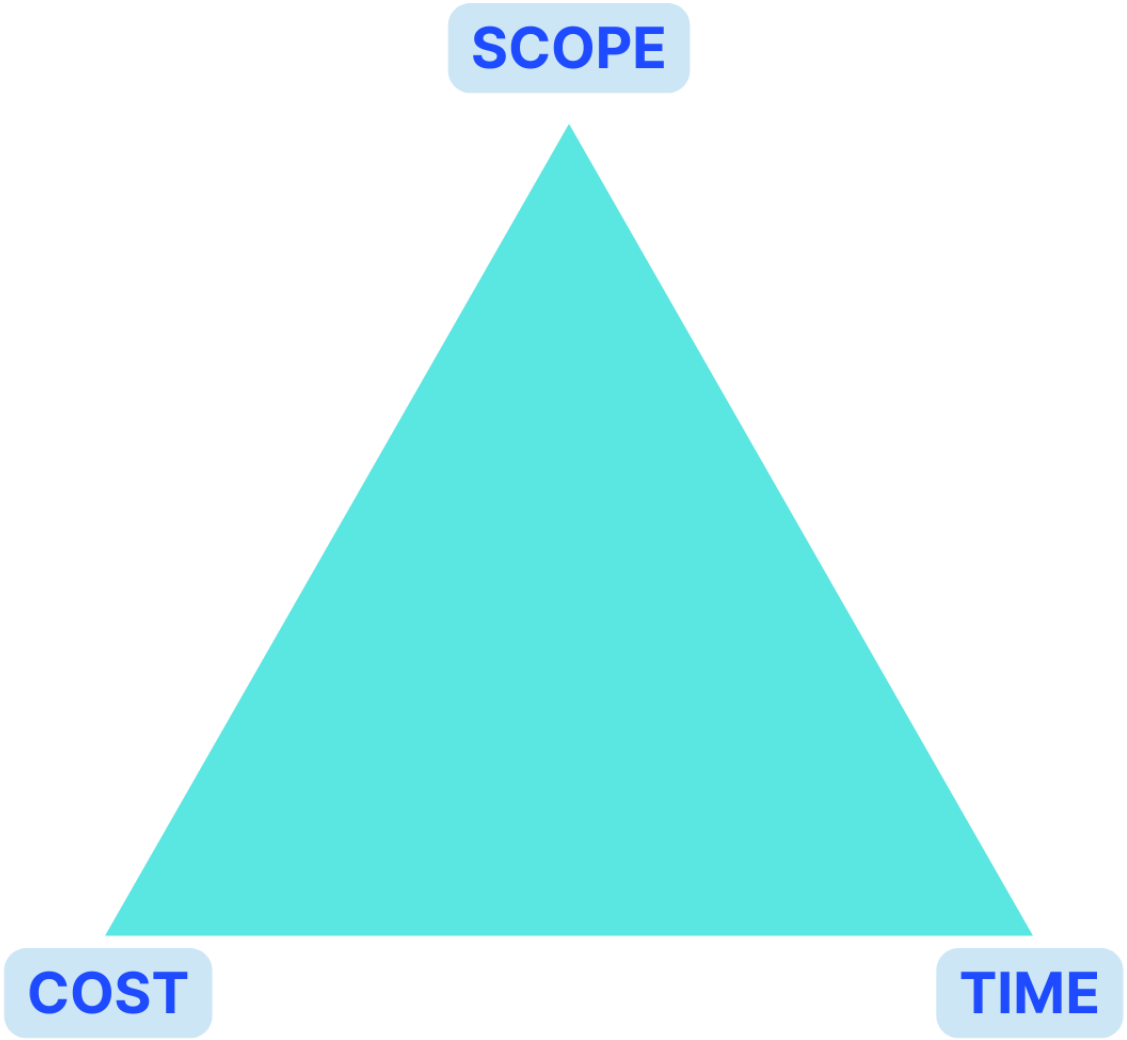
The Iron Triangle

Every project is constrained by three interdependent factors

1. **Scope:** What features will be delivered?
2. **Time:** When will it be delivered?
3. **Cost:** How much will it cost?

You can control two, but the third will adjust

- Fixed scope + fixed time = Variable cost
- Fixed cost + fixed scope = Variable time
- Fixed time + fixed cost = Variable scope (Agile!)



The Project Manager's Role

Key Responsibilities

Activity	Description
Project Planning	Estimating, scheduling, assigning people to tasks
Reporting	Communicating progress to customers and management
Risk Management	Assessing and monitoring risks, taking action when problems arise
People Management	Choosing people, establishing effective ways of working
Proposal Writing	Writing proposals to win contracts (critical for company survival)
AI Governance	Setting quality gates for AI-generated code, managing new risks

The Soft Skills

- Communication at **multiple levels** (technical details to management summaries)
- Writing concise, coherent documents that abstract critical information
- Negotiation and conflict resolution
- Decision-making under uncertainty

Why Software Projects Fail

The Statistics (Random)

- **70%** of software projects fail to meet expectations
- **52%** experience cost overruns
- **45%** deliver late
- **20%** are cancelled outright

Common Causes of Failure

1. **Unclear requirements** - "What are we building?"
2. **Scope creep** - "Just one more feature..."
3. **Underestimation** - "It should only take 2 weeks"
4. **Poor communication** - "I thought you were doing that"
5. **Technical debt** - "We'll fix it later"
6. **Stakeholder disengagement** - "I'm too busy to review"

The CHAOS Report - Success Factors

The Standish Group's research identifies key factors for project success:

Factor	Impact
Executive Support	Most critical - resources and blockers
User Involvement	Continuous feedback prevents rework
Clear Requirements	Know what you're building
Realistic Expectations	Underpromise, overdeliver
Smaller Project Milestones	Frequent deliveries = less risk
Skilled Team	Competence matters
Agile Process	Flexibility to adapt

> **Key Insight:** Technical excellence is necessary but NOT sufficient. People and process matter more.

Project Planning

Project Planning Overview

> **Planning** is the process of defining what to build, how to build it, who will build it, and when it will be done.

The Planning Paradox

- At the start of a project, we have the **least information** but must make the **biggest decisions**.
- Plans are always wrong; planning is still essential.

> "Plans are worthless, but planning is everything." — Dwight D. Eisenhower

Types of Plans

Plan Type	Horizon	Detail Level	Updates
Strategic	1-5 years	High-level vision	Annually
Release	3-6 months	Features, milestones	Quarterly
Iteration	1-4 weeks	Tasks, assignments	Weekly

Cost Estimation

Why Price Differs from Cost

The price charged for software is **not** simply: `Cost + Profit Margin`

Factor	Effect on Price
Market opportunity	Lower price to enter new market segment
Contractual terms	Price reduced if customer owns source code
Requirements volatility	Higher price for uncertain requirements
Financial health	Lower price if company needs cash flow
Competitive pressure	Price matched to competitors

The Pricing Game

> Software is often priced to **win a contract**, then functionality is adjusted to meet the estimated price.

Reality: Many projects start under-priced and rely on change requests (or VC money) to become profitable.

Milestones and Deliverables

Milestone

A **milestone** is a recognizable end-point of a software process activity.

- Formal output produced (report, document, review)
- Should **never slip** in planning (reschedule activities instead)
- Used to measure progress

Deliverable

A **deliverable** is a project result delivered to the customer.

Type	Examples
Internal Milestone	Design review completed, test plan approved
External Deliverable	Requirements document, software release

Work Breakdown Structure (WBS)

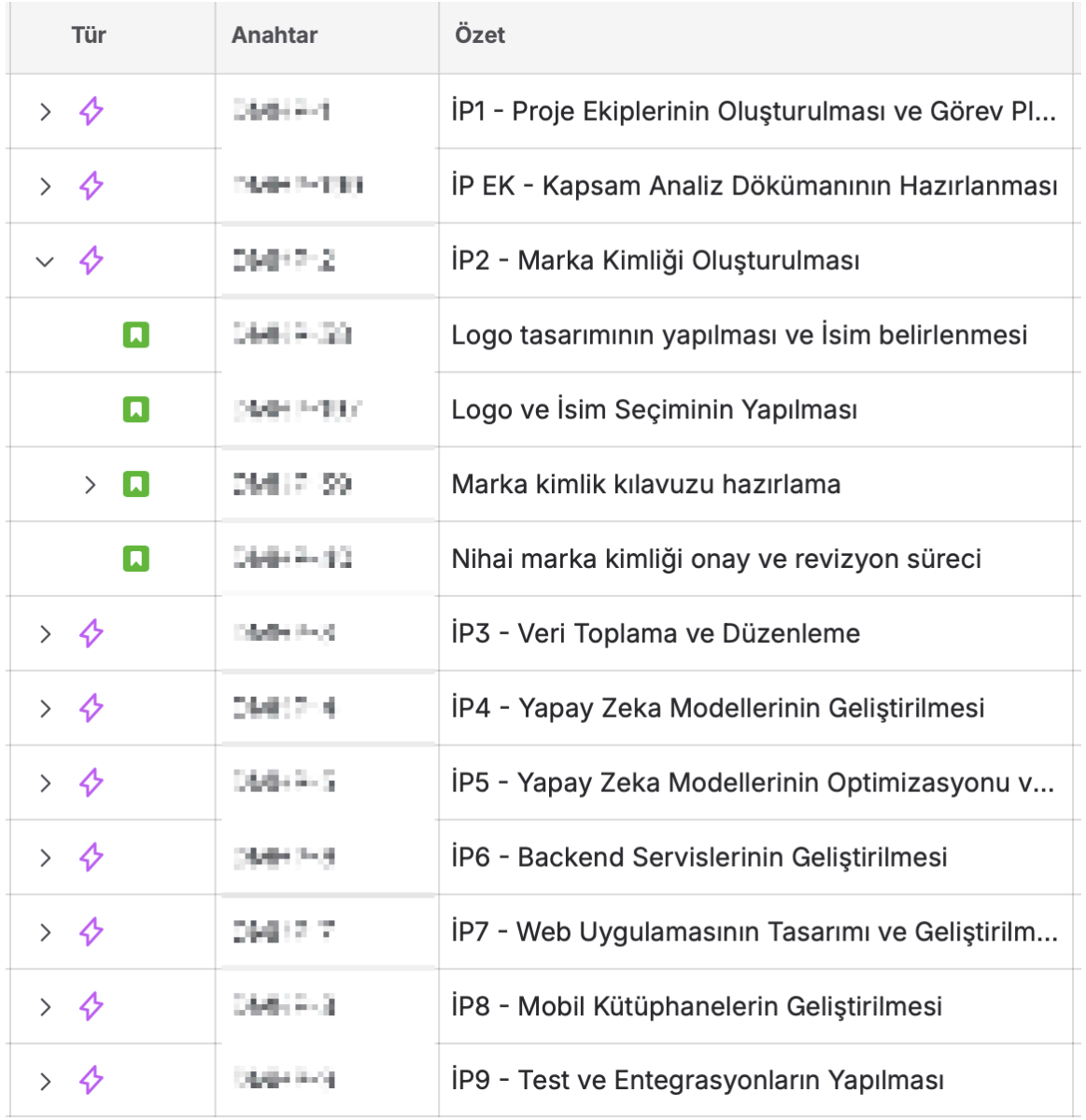
A **WBS** decomposes a project into smaller, manageable work packages.

Principles

1. **100% Rule:** WBS must include 100% of the work
2. **Mutually Exclusive:** No overlap between packages
3. **Outcome-Focused:** Define deliverables, not activities
4. **8/80 Rule:** Work packages should take 8-80 hours (optional)

Benefits

- Makes estimation easier (smaller pieces)
- Enables parallel work assignment
- Provides clear progress tracking
- Identifies dependencies early



Tür	Anahtar	Özet
		İP1 - Proje Ekiplerinin Oluşturulması ve Görev Pl...
		İP EK - Kapsam Analiz Dökümanının Hazırlanması
		İP2 - Marka Kimliği Oluşturulması
		Logo tasarımının yapılması ve İsim belirlenmesi
		Logo ve İsim Seçiminin Yapılması
		Marka kimlik kılavuzu hazırlama
		Nihai marka kimliği onay ve revizyon süreci
		İP3 - Veri Toplama ve Düzenleme
		İP4 - Yapay Zeka Modellerinin Geliştirilmesi
		İP5 - Yapay Zeka Modellerinin Optimizasyonu v...
		İP6 - Backend Servislerinin Geliştirilmesi
		İP7 - Web Uygulamasının Tasarımı ve Geliştirilm...
		İP8 - Mobil Kütüphanelerin Geliştirilmesi
		İP9 - Test ve Entegrasyonların Yapılması

WBS Example - CampusPal

```
CampusPal v2.0
├── 1.0 User Management
│   ├── 1.1 Authentication Module
│   │   ├── 1.1.1 OAuth2 Integration
│   │   ├── 1.1.2 Password Reset Flow
│   │   └── 1.1.3 Session Management
│   └── 1.2 Profile Module
│       ├── 1.2.1 Profile Editing
│       └── 1.2.2 Avatar Upload
├── 2.0 Event System
│   ├── 2.1 Event CRUD
│   ├── 2.2 Registration System
│   └── 2.3 Calendar Integration
├── 3.0 Notification System
│   ├── 3.1 Push Notifications
│   └── 3.2 Email Notifications
└── 4.0 Testing & Deployment
    ├── 4.1 Integration Testing
    └── 4.2 Production Deployment
```

Practice

Scenario: You're building a "Campus Marketplace" feature for CampusPal where students can buy/sell used textbooks.

Task: Create a WBS with at least 3 levels.

- User-facing features (listing, searching, messaging)
- Backend services (payments, moderation)
- Non-functional requirements (testing, deployment)

Answer

```
Campus Marketplace
├── 1.0 Listing Management
│   ├── 1.1 Create Listing
│   │   ├── 1.1.1 Photo Upload
│   │   ├── 1.1.2 ISBN Lookup API
│   │   └── 1.1.3 Price Suggestion
│   ├── 1.2 Edit/Delete Listing
│   └── 1.3 Listing Expiration
├── 2.0 Search & Discovery
│   ├── 2.1 Search by Title/ISBN
│   ├── 2.2 Filter by Category/Price
│   └── 2.3 Saved Searches & Alerts
├── 3.0 Messaging System
│   ├── 3.1 In-App Chat
│   └── 3.2 Notification Integration
├── 4.0 Safety & Moderation
│   ├── 4.1 Content Moderation
│   └── 4.2 User Reporting
└── 5.0 Testing & Launch
    ├── 5.1 Load Testing
    └── 5.2 Beta Release
```

Scheduling - Gantt Charts

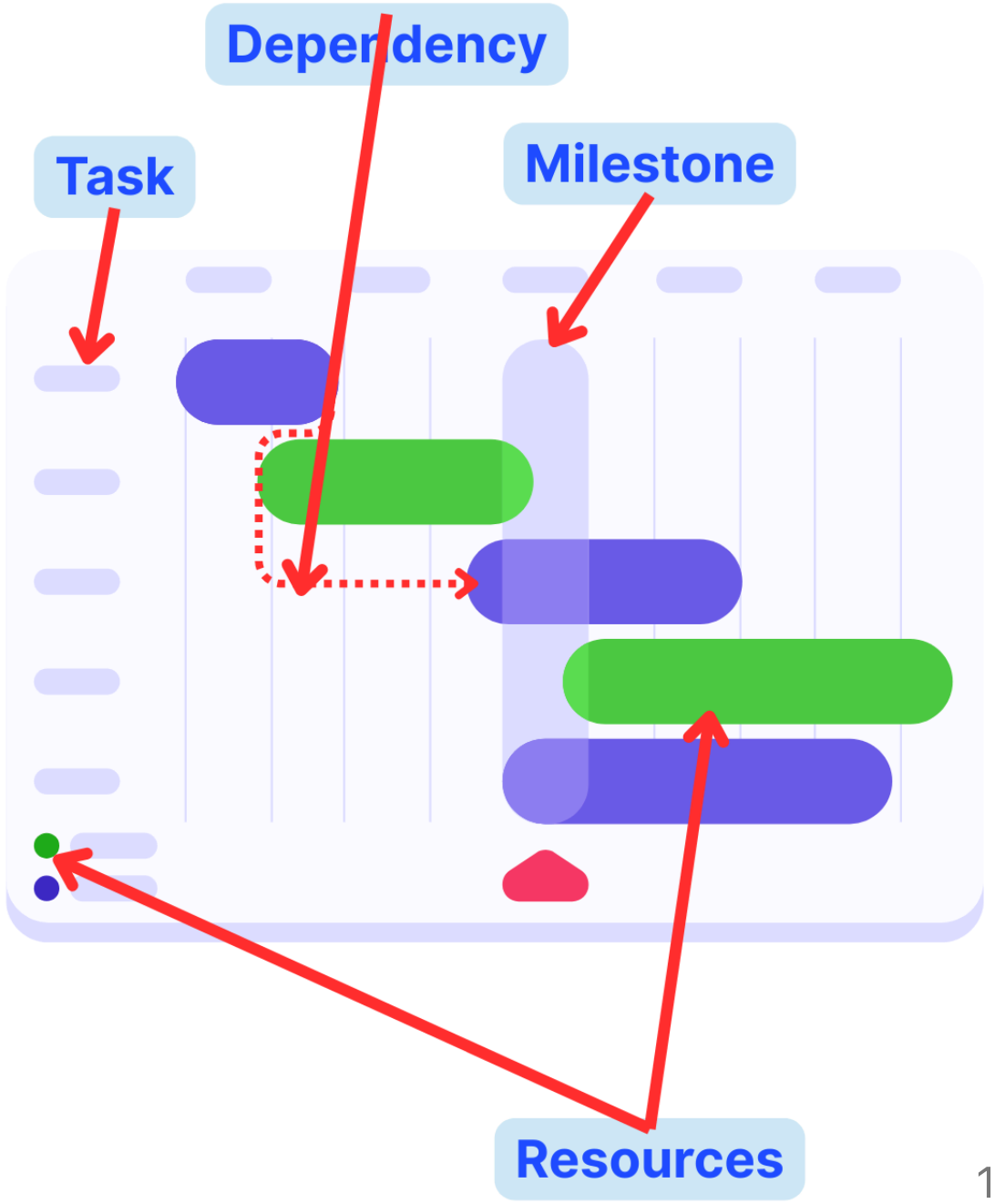
A **Gantt chart** visualizes the project schedule over time

Components

- **Tasks:** Horizontal bars showing duration
- **Dependencies:** Arrows linking related tasks
- **Milestones:** Diamond markers for key dates
- **Resources:** Who is assigned to each task

Creating a Gantt Chart

1. List all tasks from WBS
2. Estimate duration for each task
3. Identify dependencies (what must finish before this starts?)
4. Assign resources
5. Calculate the critical path



Critical Path Method (CPM)

The **Critical Path** is the longest sequence of dependent tasks. It determines the minimum project duration.

Example

Path 1: A -> C -> E = 3 + 5 + 2 = **10 days**
Path 2: B -> D -> F = 2 + 4 + 3 = **9 days**

Critical Path: A → C → E (10 days)

Implications

- Any delay on the critical path delays the project
- Non-critical tasks have **buffer time**
- Focus resources on critical path tasks

Dependencies and Constraints

Types of Dependencies

Type	Description	Example
Finish-to-Start (FS)	B starts after A finishes	Code -> Test
Start-to-Start (SS)	B starts when A starts	Design -> Prototype
Finish-to-Finish (FF)	B finishes when A finishes	Testing -> Bug fixes
Start-to-Finish (SF)	B finishes when A starts	Rare

Constraint Types

- **Must Start On:** Fixed start date (e.g., regulatory deadline)
- **Must Finish On:** Fixed end date (e.g., conference demo)
- **As Soon As Possible:** Default - start when dependencies allow
- **As Late As Possible:** Delay until necessary (risky)

Estimation

The Estimation Challenge

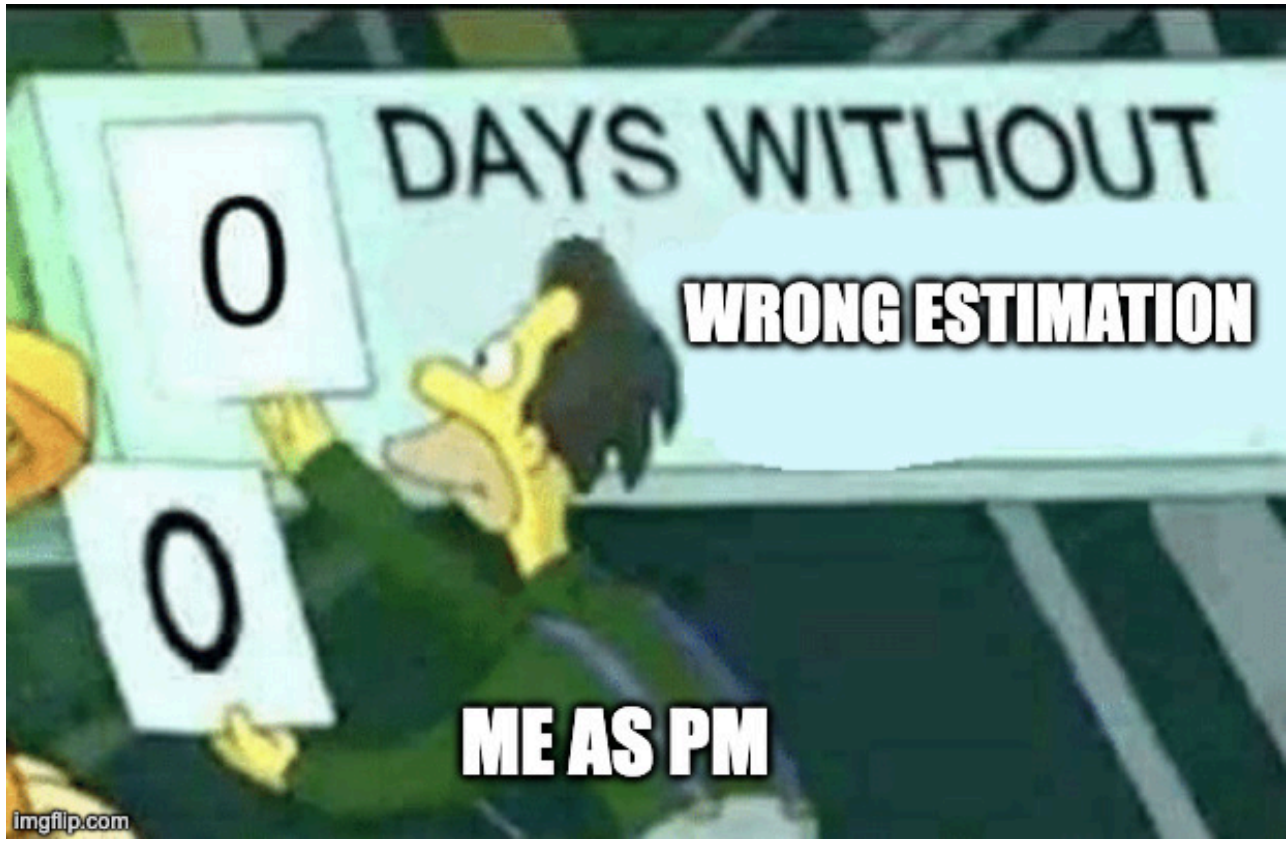
> Estimation is the process of predicting how much effort, time, or cost a project will require.

Why Estimation is Hard

1. **Uncertainty:** We don't know what we don't know
2. **Complexity:** Software has many interdependent parts
3. **Human factors:** Illness, learning curves, motivation
4. **Requirements change:** Scope evolves during development
5. **Cognitive biases:** Optimism, anchoring, planning fallacy
6. **AI unpredictability:** Vibe coding can 10x speed... or add hidden debt

The Cone of Uncertainty

Early estimates can be off by **4x** in either direction. With AI assistance, this variance can be even **wider**—instant prototypes, but the *last 20%* may take longer than expected.



Estimation Techniques

Technique	Description	Best For
Expert Judgment	Ask experienced developers	Small teams, familiar domains
Analogous	Compare to similar past projects	Repeatable projects
Parametric	Use mathematical models (COCOMO)	Large projects, organizational data
Bottom-Up	Estimate each task, sum up	Detailed planning
Three-Point	Optimistic + Pessimistic + Likely	Uncertainty quantification
Story Points	Relative sizing (Fibonacci)	Agile sprint planning
T-Shirt Sizing	XS, S, M, L, XL, XXL	Roadmaps, early-stage planning

Expert Judgment

A structured approach to gathering expert estimates.

The Process

1. **Individual Estimates:** Each expert estimates independently
2. **Anonymous Sharing:** Share estimates without names
3. **Discussion:** Discuss reasons for high/low estimates
4. **Re-estimate:** Experts revise their estimates
5. **Repeat:** Until consensus (or accept range)

Benefits

- Reduces anchoring bias (first estimate doesn't dominate)
- Captures diverse perspectives
- Discussion uncovers hidden assumptions
- Builds team ownership of the estimate

Three-Point Estimation (PERT)

Uses three estimates to calculate expected duration and uncertainty.

The Three Points

- **O (Optimistic):** Best-case scenario (everything goes right)
- **M (Most Likely):** Realistic expectation
- **P (Pessimistic):** Worst-case scenario (everything goes wrong)

PERT Formula

$$E = \frac{O + 4M + P}{6}$$

$$\sigma = \frac{P - O}{6}$$

Three-Point Estimation (PERT)

Example

Task: Implement OAuth2 login

- O = 2 days, M = 5 days, P = 14 days

$$E = \frac{2 + 4(5) + 14}{6} = \frac{36}{6} = 6 \text{ days}$$

$$\sigma = \frac{14 - 2}{6} = 2 \text{ days}$$

COCOMO II - Constructive Cost Model

A parametric model that estimates effort based on size, complexity, and project attributes.

COCOMO II Formula

$$\text{Effort} = A \times \text{Size}^{B} \times M$$

Where:

- **A** = Constant (typically 2.94)
- **Size** = Thousands of lines of code (KLOC)
- **B** = Exponent based on scale factors
- **M** = Multiplier from cost drivers

The Exponent B

$$B = 1.01 + 0.01 \times \sum_{i=1}^{5} SF_i$$

Scale factors (SF) capture project-wide influences on effort.

COCOMO II Scale Factors

Scale Factor	Description	Low = Less Effort
PREC (Precedentedness)	How similar to previous projects?	Very familiar
FLEX (Flexibility)	How rigid are requirements?	Very flexible
RESL (Risk Resolution)	How well are risks addressed?	Thorough
TEAM (Team Cohesion)	How well does team work together?	Seamless
PMAT (Process Maturity)	CMM maturity level	Level 5

Effect on Exponent

- All factors rated "Very Low" -> B ≈ 1.01 (almost linear)
- All factors rated "Very High" -> B ≈ 1.26 (significant diseconomies of scale)

> A 10 KLOC project with B=1.26 requires **80% more effort** than with B=1.01!

COCOMO II Cost Drivers

Cost drivers are **multipliers** that adjust effort based on project specifics.

Category	Examples
Product	Required reliability (RELY), Database size (DATA), Complexity (CPLX)
Platform	Execution time constraint (TIME), Storage constraint (STOR)
Personnel	Analyst capability (ACAP), Programmer capability (PCAP), Experience
Project	Use of tools (TOOL), Multi-site development (SITE), Schedule pressure

Example Multipliers

Driver	Very Low	Low	Nominal	High	Very High
RELY	0.82	0.92	1.00	1.10	1.26
PCAP	1.34	1.15	1.00	0.88	0.76

> Hiring **high-capability programmers** reduces effort by 12% (PCAP High = 0.88)

Story Points - Relative Estimation

Story points measure the **relative complexity** of work items, not absolute time.

Why Story Points?

- Developers are bad at absolute estimates
- Developers are better at **relative** comparison
- Abstracts away individual speed differences
- Team velocity becomes predictable over time

Fibonacci Scale

1, 2, 3, 5, 8, 13, 21...

The gaps grow larger because uncertainty increases with size. Big stories should be broken down.

Planning Poker

A collaborative estimation technique using story points.

The Process

1. **Present Story:** Product owner explains the user story
2. **Discuss:** Team asks clarifying questions
3. **Vote:** Everyone simultaneously reveals their estimate card
4. **Discuss Outliers:** Highest and lowest explain their reasoning
5. **Re-vote:** Repeat until consensus

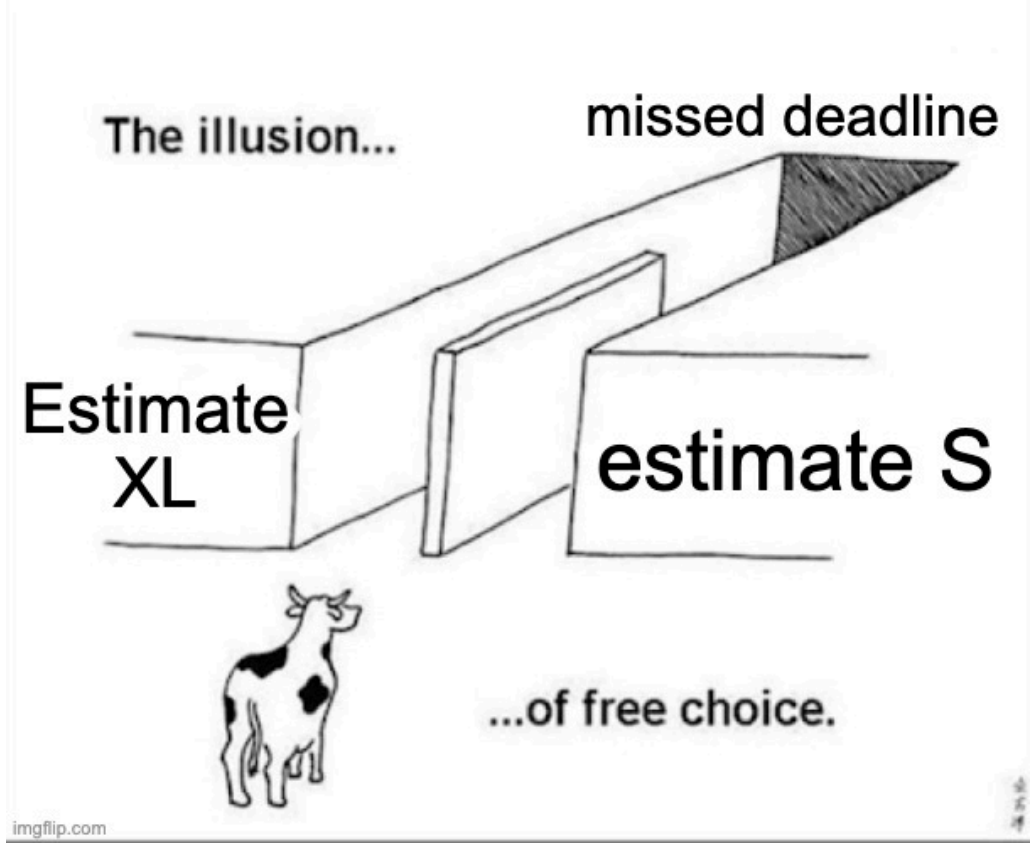
Benefits

- Everyone participates (no dominant voices)
- Discussion uncovers hidden complexity
- Builds shared understanding
- Fast and engaging

T-Shirt Sizing

A quick, intuitive way to estimate effort, especially for early-stage planning.

Size	Meaning	Typical Effort	Example
XS	Trivial change	< 1 day	Fix a typo, update a constant
S	Small, well-understood	1-2 days	Add a form field, simple bug
M	Moderate complexity	3-5 days	New API endpoint with tests
L	Significant work	1-2 weeks	New feature with UI + backend
XL	Large, needs breakdown	2-4 weeks	Integration with external API
XXL	Epic-sized, split it!	> 1 month	New module, major refactoring



Common Estimation Scales Comparison

Scale	Values	Best For	Precision
T-Shirt	XS, S, M, L, XL, XXL	Roadmaps, early planning	Low
Fibonacci	1, 2, 3, 5, 8, 13, 21	Sprint planning	Medium
Powers of 2	1, 2, 4, 8, 16, 32	Technical tasks	Medium
Hours/Days	2h, 4h, 1d, 2d, 3d, 5d	Detailed task breakdown	High
Bucket System	0, 1, 2, 3, 4, 5, 8, 13, 20+	Large backlog estimation	Medium

The "No Estimates" Movement

Some teams argue estimation is waste. Instead:

- Break everything into **small, similar-sized tasks**
- Count tasks, not points
- Focus on **throughput** (tasks/week) instead of velocity

> Works well for mature teams with consistent task breakdown discipline.

Practice

Scenario: Estimate these CampusPal features using Planning Poker (1, 2, 3, 5, 8, 13).

Feature	Your Estimate
Add a "forgot password" email link	?
Implement push notifications for events	?
Add dark mode to the mobile app	?
Integrate with university SSO (SAML)	?
Add real-time chat for event attendees	?

Sample Answer

Feature	Estimate	Reasoning
Forgot password email link	**3**	Well-defined, but needs email service integration
Push notifications for events	**8**	Platform-specific (iOS/Android), backend work
Dark mode for mobile app	**5**	UI changes across many screens, testing
University SSO (SAML) integration	**13**	Complex protocol, depends on IT cooperation
Real-time chat for attendees	**13**	New infrastructure (WebSockets), moderation

Key Insight: The value isn't the number, it's the **discussion** that reveals complexity.

Velocity and Forecasting

Velocity is the average number of story points completed per sprint.

Calculating Velocity

Sprint	Points Completed
Sprint 1	24
Sprint 2	28
Sprint 3	22
Sprint 4	26
Average	**25**

Using Velocity for Forecasting

Remaining work: 200 story points
Average velocity: 25 points/sprint

Estimated sprints remaining: 200 / 25 = **8 sprints**

Risk Management

What is Risk?

> A **risk** is something that you'd prefer not to have happen. Risk management involves anticipating risks and taking action to avoid them.

Risk Components

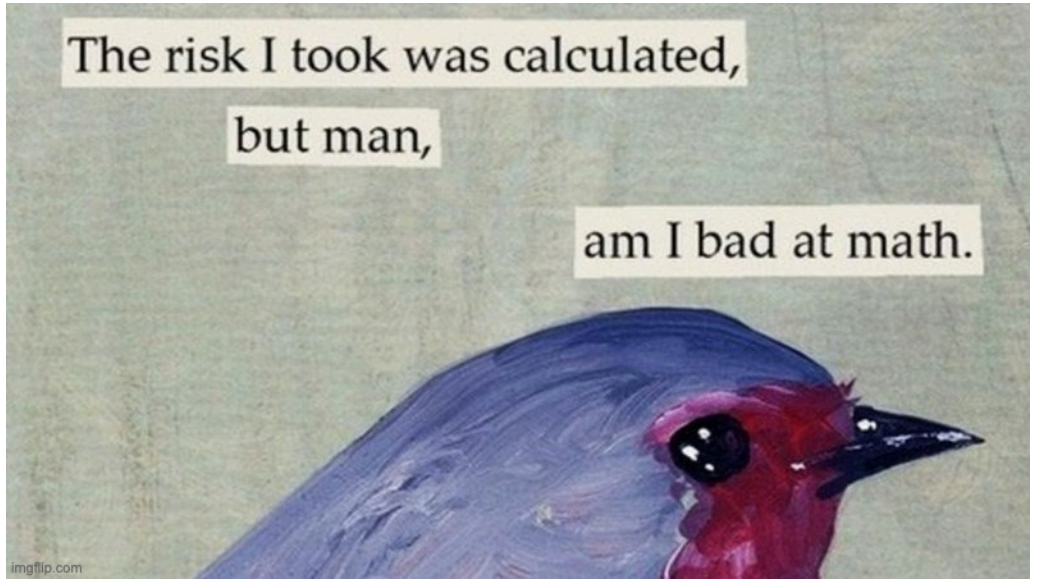
$$\text{Risk Exposure} = \text{Probability} \times \text{Impact}$$

Risk vs. Issue

Term	Definition
Risk	Something that **might** happen
Issue	Something that **has** happened

The Risk Management Goal

Identify risks early, when you can still do something about them!



Three Categories of Risk

Category	Description	Example
Project Risks	Affect the project schedule or resources	Loss of experienced designer delays the schedule
Product Risks	Affect the quality or performance of the software	Purchased component fails to perform as expected
Business Risks	Affect the organization developing the software	Competitor releases a similar product first

Risks Often Overlap

Example: Key programmer leaves the project

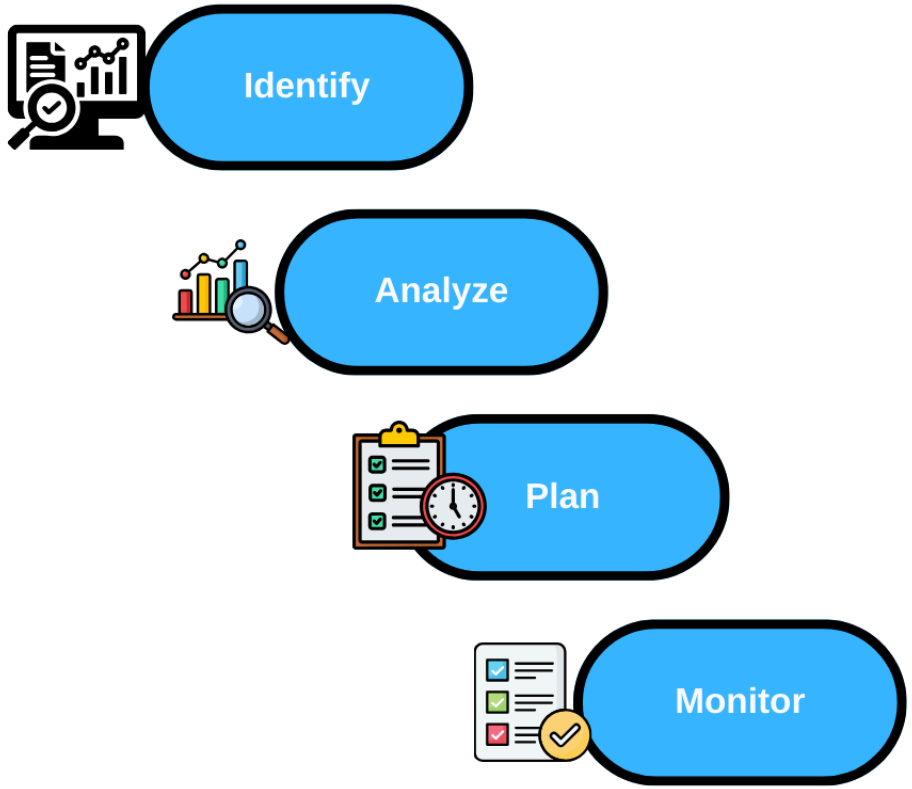
- **Project Risk:** Schedule affected (new person needs time to learn)
- **Product Risk:** Replacement may make more errors (less experience)
- **Business Risk:** Their expertise may be crucial for winning new contracts

Risk Management Process

The risk management process is **continuous** throughout the project

Frequency

- **Identification:** Every sprint planning
- **Analysis:** When new risks emerge
- **Planning:** Once per major risk
- **Monitoring:** Every sprint review



Risk Identification Techniques

1. Brainstorming - Sticky Notes

- Gather the team
- No judgment, all ideas welcome
- Group and categorize afterward

2. Checklist Review

Use historical data from past projects:

- What risks occurred before?
- What surprised us?
- What would we do differently?

3. SWOT Analysis

- **S**trengths: Internal advantages
- **W**eaknesses: Internal vulnerabilities
- **O**pportunities: External positive factors
- **T**hreats: External risks

Common Project Risks

Risk	Affects	Description
Staff turnover	Project	Experienced staff leave before project completion
Management change	Project	New management with different priorities
Hardware unavailability	Project	Essential hardware not delivered on schedule
Requirements change	Project & Product	More changes than anticipated
Specification delays	Project & Product	Interface specs not available on schedule
Size underestimate	Project & Product	System size significantly underestimated
Technology change	Business	Technology the system is built on is superseded
Product competition	Business	Competitor releases before project completes
AI-generated code debt	Product	Hidden bugs, inconsistent patterns in AI code
Knowledge gaps	Project	Team can't maintain code they didn't write

Risk Categories by Domain

Category	Examples
Technical	New technology, integration complexity, performance
Requirements	Unclear scope, changing requirements, stakeholder conflict
Schedule	Optimistic estimates, dependencies, resource availability
Resource	Key person leaves, skill gaps, team conflicts
External	Vendor delays, regulatory changes, market shifts
Organizational	Budget cuts, priority changes, political factors

CampusPal Risk Examples

- **Technical:** Mobile push notification service reliability
- **Requirements:** University changes data privacy policy
- **Schedule:** Final exams reduce developer availability
- **Resource:** Only one developer knows the legacy codebase
- **External:** Google changes OAuth2 requirements

Risk Register

A **Risk Register** is the central document tracking all identified risks.

Risk Register Template

ID	Risk	Category	Probability	Impact	Exposure	Owner	Response	Status
R01	Key dev leaves	Resource	Medium	High	High	PM	Cross-train	Active
R02	API vendor price increase	External	Low	Medium	Low	Lead	Alternative vendors	Active
R03	Performance under load	Technical	Medium	High	High	Tech Lead	Load testing	Mitigated
R04	Scope creep	Requirements	High	Medium	High	PM	Change control	Active

Practice

Scenario: Your team is building CampusPal's new "Campus Shuttle Tracker" feature with real-time GPS tracking.

Task: Identify 5 risks and categorize them.

Risk	Category
1.	
2.	
3.	
4.	
5.	

Sample Answer

Risk	Category
GPS accuracy issues indoors/tunnels	**Technical**
University IT delays API access	**External**
Students don't adopt the feature	**Requirements**
Battery drain from continuous GPS	**Technical**
Privacy concerns about location tracking	**Requirements**
Mobile dev gets sick before launch	**Resource**
Shuttle routes change mid-semester	**External**
Integration with existing maps fails	**Technical**

Quantitative Risk Analysis

For critical decisions, use numbers instead of categories.

Expected Monetary Value (EMV)

$$\text{EMV} = \text{Probability} \times \text{Impact (Cost)}$$

Example

Risk	Probability	Impact ($)	EMV ($)
Server hardware failure	20%	$50,000	$10,000
Vendor contract dispute	10%	$200,000	$20,000
Key developer leaves	30%	$80,000	$24,000
Total Risk Exposure			**$54,000**

Decision Rule

If the cost of mitigation < EMV, then mitigate.

Risk Response Strategies

Strategy	Description	Example
Avoid	Eliminate the risk entirely	Don't use unproven technology
Mitigate	Reduce probability or impact	Add automated testing
Transfer	Shift risk to another party	Buy insurance, use SaaS
Accept	Acknowledge and plan for it	Set aside contingency budget

Risk Response Examples

1. Avoidance

Risk: Using bleeding-edge framework that might have breaking changes

Response: Use stable, well-documented framework instead

2. Mitigation

Risk: Performance issues under high load

Response: Implement load testing early, design for horizontal scaling

3. Transfer

Risk: Payment processing security

Response: Use Stripe instead of building our own payment system

4. Acceptance

Risk: Minor UI bugs in edge cases

Response: Track and fix in future sprints (low priority backlog)

Contingency Planning

A **contingency plan** defines what to do **if** a risk occurs.

Contingency Plan Template

Element	Description
Trigger	What event activates the plan?
Response	What actions will be taken?
Owner	Who is responsible?
Resources	What budget/time is reserved?
Communication	Who needs to be notified?

Contingency Planning

Example: Key Developer Leaves

Element	Content
Trigger	Developer gives notice
Response	1) Knowledge transfer sessions 2) Document critical systems 3) Begin hiring
Owner	Engineering Manager
Resources	$10K contingency for contractor
Communication	Notify stakeholders of potential delay

Practice

Scenario: You've identified this risk for CampusPal:

> **Risk:** The university IT department may not approve our API access request in time for launch.
> **Probability:** Medium (40%)
> **Impact:** High (launch delay of 4 weeks)

Task:

1. Calculate the EMV (assume 4-week delay costs $40,000)
2. Propose a response strategy
3. Create a contingency plan

Answer

1. EMV Calculation

$$\text{EMV} = 0.40 \times \$40,000 = \$16,000$$

2. Response Strategy: Mitigate

- Schedule weekly check-ins with IT department
- Provide all documentation proactively
- Start approval process 8 weeks early (not 4)

3. Contingency Plan

Element	Content
Trigger	2 weeks before launch, no approval
Response	Launch with mock data; enable real API later
Owner	Product Manager
Resources	1 sprint of dev time for mock service
Communication	Notify stakeholders of limited functionality

Risk Monitoring

Risks don't stay static, they evolve throughout the project.

Monitoring Activities

Activity	Frequency	Purpose
Risk Review Meeting	Weekly/Bi-weekly	Update status, identify new risks
Trigger Tracking	Continuous	Watch for contingency triggers
Metric Monitoring	Sprint	Track risk indicators (velocity, defects)
Lessons Learned	End of phase	Capture new risks for future projects

Risk Indicators

- Velocity dropping consistently
- Team morale declining
- Stakeholder engagement decreasing
- Technical debt growing
- Dependencies slipping

Agile Project Management

Agile Project Management

Agile changes **HOW** we manage projects, not **WHETHER** we manage them.

Key Differences from Traditional PM

Aspect	Traditional (Waterfall)	Agile
Planning	Big upfront plan	Rolling wave, adaptive
Scope	Fixed scope, variable time	Fixed time, variable scope
Documentation	Heavy, formal	Light, just enough
Change	Controlled, resisted	Expected, welcomed
Delivery	Big bang at end	Continuous, incremental
Team	Assigned resources	Self-organizing

Scrum Roles and Ceremonies

Roles

Role	Responsibility
Product Owner	Defines what to build, prioritizes backlog
Scrum Master	Facilitates process, removes blockers
Development Team	Builds the product, self-organizes

Ceremonies

Ceremony	Duration	Purpose
Sprint Planning	2-4 hours	Plan the sprint backlog
Daily Standup	15 minutes	Sync, identify blockers
Sprint Review	1-2 hours	Demo completed work
Retrospective	1-2 hours	Improve the process

Sprint Planning

The Goal

Select work from the backlog that the team can complete in one sprint.

The Process

1. **Product Owner** presents prioritized backlog items
2. **Team** asks clarifying questions
3. **Team** estimates effort (story points)
4. **Team** commits to a sprint goal
5. **Team** breaks stories into tasks

Sprint Capacity

```
Team Velocity (average): 30 story points/sprint
Available capacity this sprint: 90% (1 person on vacation)
Sprint Target: 30 × 0.90 = 27 story points
```

The Product Backlog

A prioritized list of everything that might be needed in the product.

Backlog Item Structure

```
As a [user type],
I want to [action],
So that [benefit].

Acceptance Criteria:
- Given [context], when [action], then [result]
- Given [context], when [action], then [result]
```

Example

```
As a student,
I want to receive push notifications for my registered events,
So that I don't miss important campus activities.

Acceptance Criteria:
- Given I'm registered for an event, when it's 1 hour away, then I receive a notification
- Given I'm registered for an event, when it's cancelled, then I receive a notification immediately
```

Tracking Progress - Burndown Charts

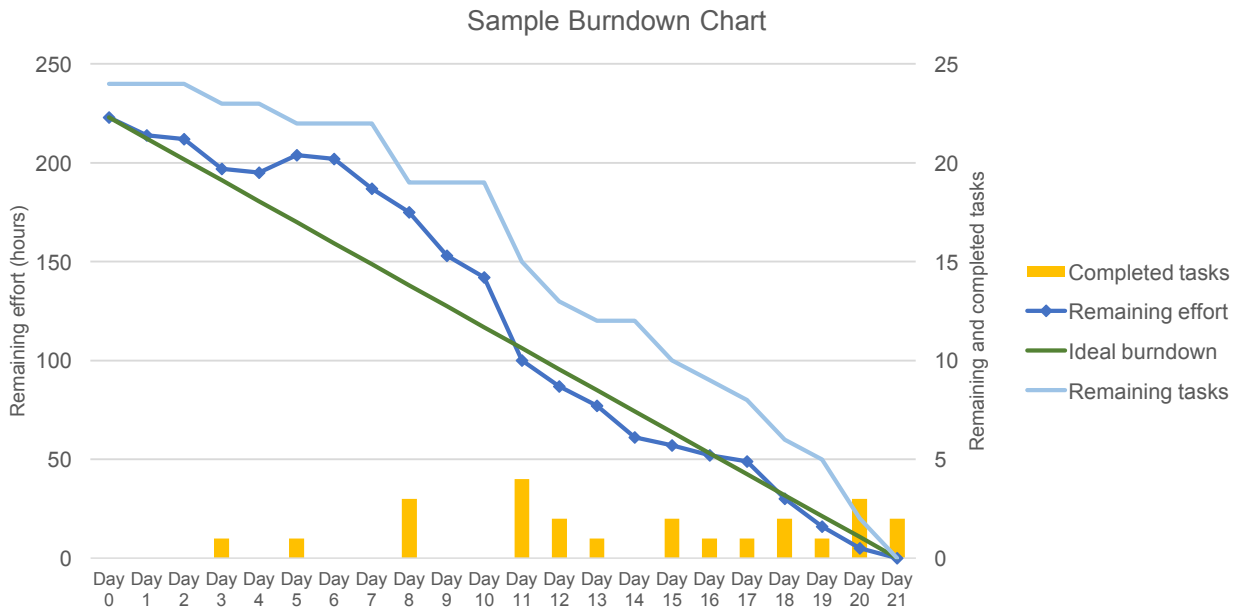
A **burndown chart** shows remaining work versus time.

Reading the Chart

- **Ideal Line:** Linear decrease to zero
- **Actual Line:** Real progress
- **Above Ideal:** Behind schedule
- **Below Ideal:** Ahead of schedule

Warning Signs

- Flat line = blocked work
- Upward spike = scope added mid-sprint
- Steep drop at end = poor estimation



Earned Value Management (EVM)

A quantitative method to track project performance.

Key Metrics

Metric	Formula	Meaning
PV (Planned Value)	Budget × % planned	What we planned to do
EV (Earned Value)	Budget × % complete	What we actually did
AC (Actual Cost)	Actual spending	What it actually cost
SV (Schedule Variance)	EV - PV	Ahead/behind schedule
CV (Cost Variance)	EV - AC	Over/under budget
SPI (Schedule Performance Index)	EV / PV	Efficiency (>1 = ahead)
CPI (Cost Performance Index)	EV / AC	Efficiency (>1 = under)

EVM Example

Project: CampusPal Phase 2
Total Budget: $100,000
Planned Duration: 10 weeks

At Week 5:

- **Planned:** 50% complete
- **Actual:** 40% complete
- **Spent:** $55,000

Calculations

- PV = $100,000 × 50% = **$50,000**
- EV = $100,000 × 40% = **$40,000**
- AC = **$55,000**
- SV = $40,000 - $50,000 = **-$10,000** (behind schedule)
- CV = $40,000 - $55,000 = **-$15,000** (over budget)
- SPI = $40,000 / $50,000 = **0.80** (80% schedule efficiency)
- CPI = $40,000 / $55,000 = **0.73** (73% cost efficiency)

EVM - Forecasting

Use current performance to predict final outcomes.

Estimate at Completion (EAC)

$$\text{EAC} = \frac{\text{Budget}}{\text{CPI}}$$

Example (continued)

Original Budget: $100,000

Current CPI: 0.73

$$\text{EAC} = \frac{\$100,000}{0.73} = \$137,000$$

Interpretation: At current efficiency, the project will cost **$137,000** instead of $100,000—a **$37,000 overrun**.

Practice - EVM

Scenario: Your CampusPal project at Week 8 of 12:

- **Budget:** $60,000
- **Planned completion:** 67%
- **Actual completion:** 75%
- **Actual spending:** $45,000

Calculate:

1. PV, EV, AC
2. SV, CV
3. SPI, CPI
4. Forecast (EAC)
5. Interpretation

Metric	Formula	Meaning
PV (Planned Value)	Budget × % planned	What we planned to do
EV (Earned Value)	Budget × % complete	What we actually did
AC (Actual Cost)	Actual spending	What it actually cost
SV (Schedule Variance)	EV - PV	Ahead/behind schedule
CV (Cost Variance)	EV - AC	Over/under budget
SPI (Schedule Performance Index)	EV / PV	Efficiency (>1 = ahead)
CPI (Cost Performance Index)	EV / AC	Efficiency (>1 = under)

Answer

1. Basic Metrics

- PV = $60,000 × 67% = **$40,200**
- EV = $60,000 × 75% = **$45,000**
- AC = **$45,000**

2. Variances

- SV = $45,000 - $40,200 = **+$4,800** (ahead of schedule!)
- CV = $45,000 - $45,000 = **$0** (on budget)

3. Indices

- SPI = $45,000 / $40,200 = **1.12** (12% ahead)
- CPI = $45,000 / $45,000 = **1.00** (exactly on budget)

4. Forecast

- EAC = $60,000 / 1.00 = **$60,000** (on target!)

5. Interpretation

Project is **ahead of schedule** and **on budget**—healthy project!

Project Staffing

The Mythical Man-Month

> "Adding people to a late software project makes it later." — Fred Brooks

Why?

1. New people need **training time** from existing staff
2. **Communication overhead** increases with team size
3. More time spent on **coordination** than development

Staffing Over Time

Projects follow a natural staffing curve; you don't expect contribution in the first few months.

Avoid rapid staff buildup early, it correlates with schedule slippage.

The AI Effect

AI doesn't need onboarding... but **AI-assisted developers** need time to learn effective prompting. Adding devs who can't use AI tools effectively may slow down an AI-native team.

Team Management

Stage	Characteristics	PM Action
Forming	Polite, uncertain	Set clear expectations
Storming	Conflict, power struggles	Facilitate, mediate
Norming	Trust develops, roles clear	Step back, support
Performing	High productivity, autonomy	Remove blockers
Adjourning	Project ends, transition	Celebrate, lessons learned

Key Insight

- Teams don't skip stages
- Change in membership restarts the cycle
- Most conflicts happen in **Storming**

Communication Management

> "The single biggest problem in communication is the illusion that it has taken place." — George Bernard Shaw

Communication Channels Formula

$$\text{Channels} = \frac{n(n-1)}{2}$$

Team Size	Channels
3	3
5	10
7	21
10	45
15	105

- Smaller teams communicate more effectively
- Large teams need formal communication structures
- Information gets lost as channels increase

Communication Plan

Stakeholder	Information Need	Frequency	Method	Owner
Sponsor	Budget, major risks	Monthly	Presentation	PM
Product Owner	Progress, blockers	Weekly	Meeting	Scrum Master
Dev Team	Tasks, dependencies	Daily	Standup	Team
Users	Features, timeline	Per release	Email/Demo	PO
University IT	Integration needs	As needed	Meeting	Tech Lead

Best Practices

- Right information to right people at right time
- Document decisions (not just discussions)
- Prefer synchronous for complex topics
- Use async for status updates

The AI Revolution in Project Management

How Vibe Coding Changes Estimation

Traditional Estimation Assumptions

Assumption	Reality with AI
Effort scales with LOC	AI generates 1000 LOC in seconds
Past velocity predicts future	AI capability changes weekly
Developer experience matters	Junior + AI ≈ Senior (sometimes)
Complexity = Time	Simple prompts can solve complex problems

New Estimation Challenges

1. **Unpredictable output quality** - Works perfectly or fails completely
2. **The "last 20%" problem** - Easy to get 80%, hard to finish
3. **Integration debt** - AI code may not fit existing architecture
4. **Review overhead** - Someone still needs to verify correctness

New Risks from AI-Assisted Development

Project Risks

Risk	Description	Mitigation
Velocity volatility	Sprint velocity becomes unpredictable	Track "verified" vs "generated" work
Knowledge gaps	Team doesn't understand AI-generated code	Mandatory code reviews, documentation
Tool dependency	Project blocked if AI service is down	Have fallback development plans

Product Risks

Risk	Description	Mitigation
Hidden bugs	AI code looks correct but has subtle errors	Increase test coverage requirements
Security vulnerabilities	AI may introduce insecure patterns	Security scanning in CI/CD
Technical debt	Verbose, non-idiomatic code	Refactoring sprints

The PM's Evolving Role

Traditional PM Skills

- Stakeholder communication
- Risk management
- Team coordination
- Schedule management

New PM Skills

Skill	Why It Matters
AI Tool Literacy	Understand what AI can/cannot do reliably
Prompt Engineering Oversight	Guide team on effective AI usage patterns
Quality Gate Design	Define when AI output needs human review
Hybrid Estimation	Blend traditional and AI-assisted timelines
Technical Debt Awareness	Recognize AI-generated debt accumulation

> The PM doesn't need to write prompts, but must understand AI's **limitations and failure modes**.

Adjusting Processes for Vibe Coding

Sprint Planning Changes

Before: "How many story points can we complete?"

Now: "What's the minimum viable human verification for AI output?"

Definition of Done (Updated)

```
# Traditional DoD

- [ ] Code complete
- [ ] Tests passing
- [ ] Code reviewed

# AI-Era DoD

- [ ] Code complete (human or AI)
- [ ] AI-generated code flagged and reviewed
- [ ] Tests passing (including edge cases AI might miss)
- [ ] Security scan passed
- [ ] Code reviewed by someone who understands the domain
- [ ] Documentation explains WHY, not just WHAT
```

AI-Assisted Risk Management

AI as a Risk Management Tool

Application	Example
Risk identification	"What could go wrong with this architecture?"
Impact analysis	"If we change X, what else breaks?"
Contingency planning	"Generate a fallback plan for this risk"
Historical analysis	"What risks occurred in similar projects?"

Caution: AI Limitations in Risk Management

- AI may **miss novel risks** not in training data
- AI provides **plausible-sounding but wrong** risk assessments
- AI doesn't understand **your organization's context**
- **Human judgment** remains essential for risk prioritization

> Use AI to **augment** risk thinking, not replace it.

Practice

Scenario: Your team is using Cursor/Claude to accelerate CampusPal development. The AI has generated 60% of the codebase in 2 weeks.

Task: Identify 3 AI-specific risks and propose mitigations.

Risk	Category	Mitigation
1.		
2.		
3.		

Answer

Risk	Category	Mitigation
AI-generated code has inconsistent error handling	**Product**	Add error handling linter, standardize patterns
Team can't maintain code they didn't write	**Project**	Pair programming sessions to understand AI code
AI hallucinated a non-existent npm package	**Product**	Lock dependencies, verify packages exist
60% of codebase has no human author to ask	**Business**	Document decisions, maintain MEMORY.md files
Future AI model changes break workflows	**Project**	Document prompts, have manual fallback process

Key Insight

> AI accelerates creation but can **slow down maintenance** if knowledge isn't transferred.

Thank You!

Next Week: Term Project Presentations

Prepare Your Demo!

- **Duration:** 20 minutes presentation (or live-demo) + 10 minutes Q&A
- **Content:** Presentation / Live demo, architecture overview, lessons learned
- **Evaluation:** Functionality, design, presentation quality

Contact Information

- **Email:** ekrem.cetinkaya@yildiz.edu.tr
- **Office Hours:** Tuesday 14:00-16:00 - Room F-B21
- **Book a slot before coming:** Booking Link
- **Course Repository:** GitHub