CS24353 • MODULE 1

# Module 1: Software Process Models, Agile & Project Management

Topics 1 to 14 • Waterfall, Prototyping, Spiral, RAD, Scrum, XP, CPM/PERT Scheduling & Risk Management

**Author:** Shaswat Raj (BIT Mesra) **Academic Year:** 2024–25 (5th Sem CSE) **Standard:** IEEE & ISO/IEC Standards

## Topic 1: Foundations of Software Engineering & The Software Crisis

**Software Engineering (IEEE 610.12)** is defined as *"the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."*

The field originated during the 1968 NATO Software Engineering Conference in Garmisch, Germany, in response to the **Software Crisis** characterized by projects running chronically over budget, missing schedules, exhibiting high defect densities, and failing to satisfy user requirements.

| Dimension | Ad-Hoc Programming (Hacker Paradigm) | Software Engineering Discipline |
|---|---|---|
| **Primary Goal** | Writing code to make a program work for the immediate user. | Building reliable, maintainable, scalable software over a multi-year lifecycle. |
| **Documentation** | Minimal or non-existent; implicit in developer's memory. | Comprehensive formal artifacts (SRS, SAD, Test Plans, SCM logs). |
| **Lifecycle Focus** | Development and debugging phase ($< 20\%$ of lifecycle cost). | Maintenance, evolution, and re-engineering ($> 70\%$ of lifecycle cost). |
| **Team Scalability** | Fails on teams $> 3$ developers due to exponential $O(n^2)$ communication channels. | Hierarchical team structures, interface contracts, and automated CI/CD pipelines. |

## Topic 2: Classical & Modified Software Process Models

A **Software Process Model** is an abstract representation of the software development lifecycle (SDLC). It specifies the sequence of engineering activities, entry/exit criteria, and intermediate deliverables.

| Process Model | Architectural Workflow | Key Strengths | Primary Limitations & Risks |
|---|---|---|---|
| **Classical Waterfall** | Strict linear sequential phases: Inception → Requirements → Design → Implementation → Testing → Deployment. | High discipline, explicit milestones, exhaustive documentation. | Zero customer feedback until final deployment; catastrophic rework if requirements change. |
| **Prototyping Model** | Rapid build of mockups → User evaluation → Requirement refinement → Final engineering. | Discovers tacit user needs; eliminates requirement ambiguity early. | Risk of "prototype becoming production code"; developer scope creep. |
| **Iterative / Incremental** | System is partitioned into functional increments delivered in staggered calendar releases. | Early business value delivery; lowers initial capital risk. | Requires robust modular architecture; risk of interface drift between increments. |
| **RAD (Rapid Application)** | Time-boxed (60–90 days) component assembly using 4GL tools and reusable frameworks. | Ultra-rapid time-to-market; high user collaboration. | Requires heavy team commitment; poor fit for high-performance low-level systems. |
| **Boehm's Spiral Model** | Risk-driven metamodel traversing 4 quadrants iteratively across expanding angular cycles. | Explicit risk assessment in every cycle; ideal for mission-critical complex systems. | High overhead; heavily dependent on expert risk management acumen. |

### Boehm's Spiral Model — Detailed 4-Quadrant Architecture

1. **Quadrant I (Determine Objectives, Alternatives & Constraints):** Identify system goals, performance targets, and architectural options.

2. **Quadrant II (Identify & Resolve Risks):** Perform formal risk analysis, build prototypes, run benchmarks, and simulate failure modes.
3. **Quadrant III (Develop & Verify Next-Level Product):** Execute standard waterfall or incremental engineering for the current spiral iteration.
4. **Quadrant IV (Review & Plan Next Phase):** Customer evaluates current increment; project team plans resources for the next spiral cycle.

## Topic 3: Agile Methodologies, Scrum Framework & Extreme Programming (XP)

The **Agile Manifesto (2001)** established 4 core values and 12 principles prioritizing adaptability over rigid prediction:

> 📜 **The 4 Core Values of the Agile Manifesto**
>
> - **Individuals and interactions** over processes and tools.
> - **Working software** over comprehensive documentation.
> - **Customer collaboration** over contract negotiation.
> - **Responding to change** over following a plan.

### The Scrum Framework — Roles, Artifacts & Ceremonies

- **Roles:**
  - **Product Owner (PO):** Owns the Product Backlog, prioritizes user stories by business value, and accepts/rejects deliverables.
  - **Scrum Master:** Servant leader who removes organizational impediments and facilitates Scrum events.
  - **Developers:** Cross-functional self-organizing team (typically 5–9 members) responsible for delivering a potentially shippable increment.
- **Artifacts:**
  - **Product Backlog:** Dynamically ordered list of user stories (Story Point Estimation via Planning Poker).
  - **Sprint Backlog:** Set of Product Backlog items selected for the current 2–4 week Sprint, decomposed into technical tasks.
  - **Increment & Definition of Done (DoD):** Fully tested, integrated, documented, and deployable code unit.
  - **Sprint Burndown Chart:** Daily plot tracking remaining effort (hours or story points) vs. calendar time.
- **Ceremonies:** Sprint Planning → Daily Scrum (15-min standup: What did I do? What will I do? Any blockers?) → Sprint Review (Live demo to stakeholders) → Sprint Retrospective (Process improvement).

## Topic 4: Project Management, CPM/PERT Scheduling & Critical Path Analysis

Project managers balance the **Iron Triangle (Scope, Time, Cost)** under quality constraints. Network scheduling models analyze task dependencies to identify project bottlenecks.

### 📐 Critical Path Method (CPM) & PERT Statistical Formulas

- **PERT Beta Expected Time ($T_e$):**

$$T_e = \frac{a + 4m + b}{6} \qquad \sigma^2 = \left(\frac{b - a}{6}\right)^2$$

  Where $a = \text{Optimistic time}, m = \text{Most likely time}, b = \text{Pessimistic time}$.

- **Forward Pass (Earliest Times):**

$$ES_j = \max_{i \in \text{Pred}(j)} [EF_i] \qquad EF_j = ES_j + D_j$$

- **Backward Pass (Latest Times):**

$$LF_i = \min_{j \in \text{Succ}(i)} [LS_j] \qquad LS_i = LF_i - D_i$$

- **Total Float (Slack):**

$$TF_i = LS_i - ES_i = LF_i - EF_i$$

$$\text{Critical Path: Set of activities where } \mathbf{TF_i = 0} \text{ (Zero Slack)}.$$

### 🏛️ Solved Numerical 1: Complete CPM Scheduling & Critical Path Calculation

A software engineering project consists of 6 activities with dependencies and durations:

| Activity | Predecessors | Duration ($D$) | $ES$ | $EF$ | $LS$ | $LF$ | Total Float ($TF$) | Critical? |
|---|---|---|---|---|---|---|---|---|
| **A (SRS)** | — | 4 days | 0 | 4 | 0 | 4 | $0 - 0 = \mathbf{0}$ | **YES** |
| **B (DB Design)** | A | 3 days | 4 | 7 | 4 | 7 | $4 - 4 = \mathbf{0}$ | **YES** |
| **C (UI Mockups)** | A | 2 days | 4 | 6 | 6 | 8 | $6 - 4 = \mathbf{2}$ | NO |
| **D (Backend API)** | B | 5 days | 7 | 12 | 7 | 12 | $7 - 7 = \mathbf{0}$ | **YES** |
| **E (Frontend Code)** | C | 4 days | 6 | 10 | 8 | 12 | $8 - 6 = \mathbf{2}$ | NO |
| **F (Integration)** | D, E | 3 days | 12 | 15 | 12 | 15 | $12 - 12 = \mathbf{0}$ | **YES** |

$$\text{Critical Path: } \mathbf{A \rightarrow B \rightarrow D \rightarrow F} \qquad \text{Minimum Total Project Duration} = \mathbf{4 + 3 + 5 + 3 = 15} \text{ Days}$$

## Topic 5: Risk Management & RMMM Strategies

**Risk ($R$)** is a probabilistic future event with negative consequence. **Risk Exposure ($RE$)** is quantified as:

$$\mathbf{RE} = \mathbf{P}(\text{Risk Event}) \times \text{Impact Cost } (\mathbf{C})$$

### The RMMM Plan (Risk Mitigation, Monitoring, and Management)

- **Risk Mitigation (Proactive Prevention):** Concrete actions taken before risk occurs (e.g., cross-training developers to mitigate the risk of key staff turnover).
- **Risk Monitoring (Tracking):** Continuous surveillance of early warning indicator metrics (e.g., tracking sprint velocity drops or defect backlog spikes).
- **Risk Management (Contingency Planning):** Predetermined fallback plans executed when a risk materializes into an active crisis (e.g., activating backup cloud servers upon primary DC outage).

## Topic 6: University Exam Problem Bank & Model Answers

**Q1. Differentiate between Classical Waterfall, Spiral, and Agile Scrum Models across 5 Core Dimensions. (10 Marks)**

| Dimension | Waterfall | Boehm's Spiral | Agile Scrum |
|---|---|---|---|
| **Requirements** | Fixed and frozen upfront. | Refined iteratively per spiral. | Dynamic, captured as user stories. |
| **Risk Handling** | No explicit risk mechanism. | Explicit risk analysis in Quadrant II. | Continuous risk reduction via short 2-week sprints. |
| **Customer Involvement** | Only at start and end. | At completion of each spiral. | Continuous active daily collaboration. |
| **Delivery Paradigm** | Monolithic single release at end. | Progressive spiral releases. | Potentially shippable software every sprint. |
| **Cost of Change** | Exponential $O(n^2)$ late in lifecycle. | Moderate; planned per iteration. | Low; embraced by design. |

**Q2. Explain the Extreme Programming (XP) Core Practices and Why Pair Programming Improves Code Quality. (8 Marks)**

**Extreme Programming (XP - Kent Beck)** pushes software engineering best practices to extreme levels:
1. **Test-Driven Development (TDD):** Write automated unit tests before writing functional code (Red $\rightarrow$ Green $\rightarrow$ Refactor).
2. **Continuous Integration (CI):** Integrate and test code into the main branch multiple times per day.
3. **Pair Programming:** Two engineers collaborate at one workstation (**Driver** writes code; **Navigator** reviews syntax, architecture, and edge cases concurrently). Studies demonstrate that pair programming eliminates $> 60\%$ of defects upfront with only a $15\%$ increase in initial development time, saving massive post-release debugging costs!

**Q3. Detail the Activities involved in Software Project Scheduling using Gantt Charts and PERT Networks. (8 Marks)**

1. **Work Breakdown Structure (WBS):** Decompose project into discrete work packages.
2. **Activity Duration Estimation:** Use PERT 3-point estimation $T_e = (a + 4m + b)/6$.
3. **Precedence Network Modeling:** Construct activity-on-node (AON) graph mapping technological dependencies.
4. **Forward & Backward Pass Computation:** Compute $ES, EF, LS, LF$ to isolate critical activities with zero slack ($TF = 0$).
5. **Resource Leveling:** Smooth peak resource utilization across non-critical tasks utilizing available float.

## Topic 12: The V-Model (Verification & Validation Dual Ladder)

The **V-Model** maps each development phase directly to its corresponding dynamic testing phase, establishing rigorous traceability between specification and verification:

| Left Wing: Decomposition & Verification | Corresponding Right Wing: Integration & Validation | Primary Testing Objectives |
|---|---|---|
| **Business Requirements & Concept** | **User Acceptance Testing (UAT)** | Validates software satisfies customer business goals and operational workflows. |
| **System Requirements Specification (SRS)** | **System Testing** | Verifies complete end-to-end functional and non-functional performance bounds. |
| **High-Level Architecture Design** | **Integration Testing** | Verifies subsystem communication interfaces, data streaming, and API contracts. |
| **Detailed Low-Level Module Design** | **Unit Testing** | Verifies algorithmic correctness of individual classes and private functions. |

## Topic 13: Kanban Framework & Little's Law in Agile Flow

**Kanban** is a Lean workflow management method emphasizing continuous delivery and Work-In-Progress (WIP) constraints:

- **Core Principles:** (1) Visualize workflow on a Kanban board, (2) Limit Work in Progress (WIP limits per column), (3) Manage flow, (4) Make process policies explicit, (5) Implement feedback loops.

- **Little's Law in Software Engineering:**

$$\text{Lead Time} = \frac{\text{Work In Progress (WIP)}}{\text{Throughput (Velocity)}}$$

*Engineering Insight:* Reducing multitasking WIP directly slashes feature delivery lead time without needing additional staff!

### 🏛️ Solved Numerical 7: Statistical PERT Expected Project Duration & Variance

A software engineering sub-project has 3 critical activities with estimated durations in days:

| Activity | Optimistic ($a$) | Most Likely ($m$) | Pessimistic ($b$) | Expected Time ($T_e$) | Variance ($\sigma^2$) |
|---|---|---|---|---|---|
| **Architecture Design** | 4 | 7 | 16 | $\frac{4+4(7)+16}{6} = \mathbf{8.0}$ | $(\frac{16-4}{6})^2 = 2^2 = \mathbf{4.0}$ |
| **Core Microservice Coding** | 6 | 12 | 24 | $\frac{6+4(12)+24}{6} = \mathbf{13.0}$ | $(\frac{24-6}{6})^2 = 3^2 = \mathbf{9.0}$ |
| **Performance Tuning** | 2 | 5 | 8 | $\frac{2+4(5)+8}{6} = \mathbf{5.0}$ | $(\frac{8-2}{6})^2 = 1^2 = \mathbf{1.0}$ |

- $$\text{Total Expected Project Duration } \mu_{\mathbf{T}} = \mathbf{8.0} + \mathbf{13.0} + \mathbf{5.0} = \mathbf{26.0} \text{ Days}$$
- $$\text{Total Project Variance } \sigma_{\mathbf{T}}^{\mathbf{2}} = \mathbf{4.0} + \mathbf{9.0} + \mathbf{1.0} = \mathbf{14.0} \implies \sigma_{\mathbf{T}} = \sqrt{\mathbf{14.0}} \approx \mathbf{3.74} \text{ Days}$$
- $$95\% \text{ Confidence Interval } (2\sigma)\text{: } \mathbf{26.0} \pm \mathbf{2}(\mathbf{3.74}) = [\mathbf{18.52}, \mathbf{33.48}] \text{ Days}$$

## Topic 14: University Exam Problem Bank (Part II)

### Q12. Compare Throwaway Prototyping with Evolutionary Prototyping. (8 Marks)

• **Throwaway (Rapid) Prototyping:** Built rapidly using mockups/scripts solely to clarify ambiguous user requirements. Once the SRS is baselined, the prototype code is completely discarded, and the production system is engineered from scratch with clean architectural standards.
• **Evolutionary Prototyping:** Starts with an initial working core prototype that is refined, refactored, and expanded incrementally through repeated user feedback cycles until it matures directly into the final production system. Requires strict modular discipline from day one to avoid architectural rot!

### Q13. Explain the Concept of Technical Debt and How it is Managed in Agile Projects. (8 Marks)

**Technical Debt (Ward Cunningham)** represents the implied cost of future rework caused by choosing an easy, short-term hack now instead of using a better, disciplined engineering approach.
• **Consequences:** Accumulated technical debt compounds interest over time, degrading development velocity and increasing bug regression rates.
• **Agile Mitigation:** Allocating a dedicated $15–20\%$ capacity in every sprint backlog strictly for code refactoring, dependency upgrades, automated test coverage expansion, and static analysis cleanup.

## Topic 15: Project Network Crashing & Cost-Time Tradeoff Analysis

When a project must meet an accelerated statutory deadline, **Project Crashing** compresses activity durations by allocating additional engineering resources:

### 📐 Activity Cost Slope Formula

$$\text{Cost Slope} = \frac{\text{Crash Cost } (\mathbf{C_c}) - \text{Normal Cost } (\mathbf{C_n})}{\text{Normal Duration } (\mathbf{D_n}) - \text{Crash Duration } (\mathbf{D_c})} = \frac{\mathbf{\Delta C}}{\mathbf{\Delta D}}$$

**Rule of Optimal Crashing:** Always crash the activity on the *Critical Path* that has the **Lowest Cost Slope**!

**🏛️ Solved Numerical 11: Complete Critical Path Network Crashing**

Critical path consists of Activity A ($D_n = 5, D_c = 3, C_n = \$1000, C_c = \$1600$) and Activity B ($D_n = 8, D_c = 5, C_n = \$2000, C_c = \$3200$):

- $$\text{Cost Slope (A)} = \frac{1600 - 1000}{5 - 3} = \frac{600}{2} = \mathbf{\$300}/\text{day}$$
- $$\text{Cost Slope (B)} = \frac{3200 - 2000}{8 - 5} = \frac{1200}{3} = \mathbf{\$400}/\text{day}$$
- $$\mathbf{\text{To crash project by 2 days: Crash Activity A by 2 days at minimum extra cost } = 2 \times \$300 = \mathbf{\$600!}}}$$

## Topic 16: Boehm's Top 10 Software Risk Items & Mitigation Strategies

| Rank | Software Risk Item | Standard Engineering Mitigation Strategy |
|---|---|---|
| **1** | Personnel Shortfalls | Staff with top talent, job matching, cross-training, retention incentives. |
| **2** | Unrealistic Schedules and Budgets | Detailed COCOMO/FP estimation, design-to-cost, incremental feature scoping. |
| **3** | Developing Wrong Software Functions | Prototyping, user journey mapping, early user surveys, acceptance tests. |
| **4** | Developing Wrong User Interface | Task analysis, prototyping, human factors interaction design, usability tests. |
| **5** | Gold Plating (Unnecessary Complexity) | Requirements scrubbing, cost-benefit analysis, strict Change Control Board. |
| **6** | Continuing Stream of Requirements Changes | High-change threshold baseline, information hiding, agile sprints. |
| **7** | Shortfalls in Externally Furnished Components | Benchmarking, vendor audits, formal capability assessments. |
| **8** | Shortfalls in Externally Performed Tasks | Pre-award audits, competitive team design, milestone inspections. |
| **9** | Real-Time Performance Shortfalls | Simulation, high-speed prototyping, architectural instrumentation. |
| **10** | Straining Computer Science Capabilities | Technical analysis, proof-of-concept prototyping, fallback algorithms. |

## Topic 17: The 12 Core Practices of Extreme Programming (XP)

| Practice Category | Core Engineering Practice | Operational Rule & Quality Objective |
|---|---|---|
| **Fine-Scale Feedback** | **1. Test-Driven Development** | Automated unit tests written before functional code; tests must pass 100%. |
| **Fine-Scale Feedback** | **2. Planning Game** | Business determines scope and priorities; engineering estimates story points. |
| **Fine-Scale Feedback** | **3. Whole Team / On-Site Customer** | Real business user sits with the engineering team to provide immediate domain feedback. |
| **Fine-Scale Feedback** | **4. Pair Programming** | Two developers collaborate at one screen to perform continuous real-time code inspection. |
| **Continuous Process** | **5. Continuous Integration (CI)** | Code integrated, built, and tested against trunk multiple times per day. |
| **Continuous Process** | **6. Design Improvement (Refactoring)** | Ruthless restructuring of internal code without changing external behavior. |
| **Continuous Process** | **7. Small Releases** | Deploying working increments into production every 1 to 2 weeks. |
| **Shared Understanding** | **8. Simple Design** | System designed to pass current tests with zero speculative premature optimization. |
| **Shared Understanding** | **9. System Metaphor** | Shared naming taxonomy and conceptual architectural story. |
| **Shared Understanding** | **10. Collective Code Ownership** | Any developer can modify and refactor any module at any time. |
| **Shared Understanding** | **11. Coding Standards** | Enforced uniform formatting and naming conventions across the codebase. |
| **Programmer Welfare** | **12. Sustainable Pace (40-Hour Week)** | Zero sustained overtime to prevent developer burnout and cognitive defect injection. |

### 🏛️ Solved Numerical 15: Activity on Arrow (AOA) vs Activity on Node (AON) Float Calculation

For an activity with duration $D = 6$, predecessor $EF = 10$, successor $LS = 22$, and successor $ES = 18$:

- $$\text{Earliest Start } (ES) = 10 \implies \text{Earliest Finish } (EF) = 10 + 6 = \mathbf{16}$$
- $$\text{Latest Finish } (LF) = 22 \implies \text{Latest Start } (LS) = 22 - 6 = \mathbf{16}$$
- $$\text{Total Float (TF)} = \mathbf{LS} - \mathbf{ES} = \mathbf{16} - \mathbf{10} = \mathbf{6} \text{ Days}$$
- $$\text{Free Float (FF)} = \min(\mathbf{ES}_{\text{succ}}) - \mathbf{EF} = \mathbf{18} - \mathbf{16} = \mathbf{2} \text{ Days}$$
- $$\text{Independent Float (IF)} = \max(\mathbf{0}, \min(\mathbf{ES}_{\text{succ}}) - \max(\mathbf{LF}_{\text{pred}}) - \mathbf{D}) = \mathbf{18} - \mathbf{10} - \mathbf{6} = \mathbf{2} \text{ Days}$$

### 🏛️ Solved Problem 19: Agile Velocity & Release Burndown Calculation

A team completes 35, 40, and 45 story points over three 2-week sprints. Total project backlog is 360 story points:

- $$\text{Average Velocity } \mathbf{V} = \frac{\mathbf{35} + \mathbf{40} + \mathbf{45}}{\mathbf{3}} = \mathbf{40} \text{ Story Points/Sprint}$$
- $$\text{Remaining Sprints Required } \mathbf{S} = \frac{\mathbf{360} - \mathbf{120}}{\mathbf{40}} = \frac{\mathbf{240}}{\mathbf{40}} = \mathbf{6} \text{ Sprints}$$
- $$\text{Estimated Calendar Delivery Time} = \mathbf{6} \times \mathbf{2} \text{ weeks} = \mathbf{12} \text{ Weeks (3 Months)!}$$

### 🏛️ Solved Problem 20: Quantitative Risk Exposure (RE) Ranking & Budget Allocation

Prioritize 3 technical project risks with probabilities and damage costs:

- Risk 1 (Database Data Corruption): $P_1 = 0.05, C_1 = \$200,000 \implies \mathbf{RE_1 = \$10,000}$.
- Risk 2 (Cloud API 24h Outage): $P_2 = 0.30, C_2 = \$50,000 \implies \mathbf{RE_2 = \$15,000}$ (Rank 1: Highest Priority!)}}$.
- Risk 3 (Developer Resignation): $P_3 = 0.20, C_3 = \$30,000 \implies \mathbf{RE_3 = \$6,000}$.

### 🏛️ Solved Problem 28: Dynamic Systems Development Method (DSDM) MoSCoW Prioritization

In DSDM time-boxed iterations, project scope is dynamically adjusted using the **MoSCoW Rules**:

- **Must Have (60% effort):** Critical non-negotiable core features without which the system is illegal or non-functional.
- **Should Have (20% effort):** High-priority capabilities with viable temporary workarounds.
- **Could Have (20% effort):** Nice-to-have features dropped first if time-box deadline is threatened.
- **Won't Have (This Time):** Explicitly deferred out-of-scope backlog items.

### 🏛️ Solved Problem 29: Feature-Driven Development (FDD) 5-Step Process Trace

$$\text{Develop Overall Model} \rightarrow \text{Build Feature List} \rightarrow \text{Plan by Feature} \rightarrow \text{Design by Feature} \rightarrow \text{Build by Feature}$$

### 🏛️ Solved Problem 38: Agile User Story Estimation with Planning Poker & Fibonacci Series

In Planning Poker, team members independently estimate user story complexity using modified Fibonacci cards $[1, 2, 3, 5, 8, 13, 20, 40, 100]$. If estimates diverge widely (e.g., dev estimates 3, tester estimates 13), high and low estimators discuss edge cases and re-vote until consensus is reached, eliminating estimation anchoring bias!

### 🏛️ Solved Problem 39: Software Capability Maturity & Process Performance Baselines

At CMMI Level 4, process performance models use historical statistical control limits $(\mu \pm 3\sigma)$ to quantitatively predict whether a project will finish within target schedule and budget parameters!

### 🏛️ Solved Problem 48: Agile Velocity & Team Capacity Planning

For a sprint of 10 working days with 6 developers working 6 productive hours/day: $\text{Team Capacity} = 6 \times 10 \times 6 = \mathbf{360}\ \text{Productive Engineering Hours}$. If average historical conversion is $8\ \text{hours/story point}$, the team commits to $\frac{360}{8} = \mathbf{45}\ \text{Story Points}$!

### 🏛️ Solved Problem 49: Earned Value Management (EVM) Metrics & Cost Performance Index (CPI)

For a project with Planned Value $\text{PV} = \$100,000$, Earned Value $\text{EV} = \$90,000$, and Actual Cost $\text{AC} = \$110,000$:

- $$\text{Cost Variance (CV)} = \text{EV} - \text{AC} = \mathbf{\$90,000 - \$110,000 = -\$20,000}\ (\mathbf{\text{OverBudget}!})$$
- $$\text{Schedule Variance (SV)} = \text{EV} - \text{PV} = \mathbf{\$90,000 - \$100,000 = -\$10,000}\ (\mathbf{\text{BehindSchedule}!})$$
- $$\text{Cost Performance Index (CPI)} = \frac{\text{EV}}{\text{AC}} = \frac{\mathbf{90,000}}{\mathbf{110,000}} \approx \mathbf{0.818} \qquad \text{SPI} = \frac{\text{EV}}{\text{PV}} = \frac{\mathbf{90,000}}{\mathbf{100,000}} = \mathbf{0.900}$$
- $$\text{Estimate at Completion (EAC)} = \frac{\text{BAC}}{\text{CPI}} = \frac{\mathbf{\$500,000}}{\mathbf{0.818}} \approx \mathbf{\$611,247}$$

### 🏛 Solved Problem 58: Software Process Tailoring & Organization Standards

Process tailoring adapts standard organizational SDLC frameworks to project-specific constraints (e.g., scaling down formal gate reviews for small internal tools while mandating full Fagan inspections for safety-critical medical firmware)!

### 🏛 Solved Problem 59: Agile Sprint Retrospective 4L Technique

$$\text{Liked (Celebrate)} \quad \text{Learned (Knowledge)} \quad \text{Lacked (Gaps)} \quad \text{Longed For (Actionable improvements for next sprint)}$$

### 🏛 Solved Problem 68: Software Process Metric Dashboards (Lead Time vs Cycle Time)

- **Lead Time:** Total calendar duration from initial customer story creation to production deployment.
- **Cycle Time:** Active engineering duration from when a developer begins writing code on a task to when it passes code review and CI tests. Measuring both exposes organizational approval bottlenecks!

### 🏛 Solved Problem 77: Agile Burndown vs Burnup Charts

- **Burndown Chart:** Tracks remaining backlog work downwards towards zero, clearly showing whether the team will hit the sprint deadline.
- **Burnup Chart:** Tracks completed work upwards towards total scope line, distinguishing between team velocity slowdowns and customer scope creep additions!

### 🏛 Solved Problem 78: Pair Programming Driver-Navigator Communication Metrics

Studies show pair programming reduces defect density by $60\%$ with only a $15\%$ increase in initial development time, saving $> 80\%$ in total lifecycle maintenance and bug patching costs!

### 🏛 Solved Problem 91: Software Team Communication Channels Formula

For a team of $n = 10$ software engineers: $\text{Channels} = \frac{n(n-1)}{2} = \frac{10 \times 9}{2} = \mathbf{45}\ \text{Communication Channels}$. In agile, breaking into two 5-person squads slashes channels to $2 \times 10 = \mathbf{20}$, cutting communication overhead by $\mathbf{55.6\%}$!

### 🏛 Solved Problem 92: Agile Definition of Ready (DoR) vs Definition of Done (DoD)

- **DoR:** Criteria a user story must satisfy *before* entering a sprint (clear acceptance criteria, estimated, dependencies resolved).
- **DoD:** Criteria a feature must satisfy *before* shipping (code reviewed, 100% unit tests passed, security scanned, documented)!

### 🏛 Solved Problem 105: Scaled Agile Framework (SAFe) vs LeSS

- **SAFe:** Highly structured multi-tier enterprise agile framework organizing teams into Agile Release Trains (ARTs) aligning 50–125 engineers across Program Increments (PIs).
- **Large-Scale Scrum (LeSS):** Lightweight scaling framework extending single-team Scrum with minimal added overhead and shared Product Backlogs!

### 🏛 Solved Problem 106: Project Post-Mortem & Knowledge Retention

Institutionalize project learnings through documented Root Cause Analysis (5 Whys), archived historical effort metrics in organizational process assets, and automated retrospectives!

### 🏛️ Solved Problem 119: Agile Sprint Spike vs Proof of Concept (PoC)

A **Spike** is a time-boxed research task (e.g., 2 days) in a sprint to explore technical feasibility, measure performance bounds, and eliminate architectural risk before estimating user stories!

### 🏛️ Solved Problem 120: Software Project Risk Mitigation Monitoring & Management (RMMM) Grid

Document detailed trigger conditions, mitigation action owners, and contingency fallback procedures for all top 10 project risks!