CCS|CSE|CIS1072- Introduction to Software Engineering

SOFTWARE PROCESS

Romenia Silva
romaniasilva@sjp.ac.lk

Capability Maturity Model (CMM)

What is CMM?

Capability Maturity Model (CMM)

- A framework for improving software development processes
- **Developer:** Software Engineering Institute (SEI) at Carnegie Mellon University
- **Purpose:** Help organizations evolve from chaotic to mature software development practices
- **Key Idea:** Process improvement leads to better software quality and predictable outcomes

Why CMM Matters?

- Reduces project failures and cost overruns
- Improves software quality and customer satisfaction
- Makes development processes more predictable
- Provides competitive advantage

The Problem CMM Solves

Common Software Development Problems

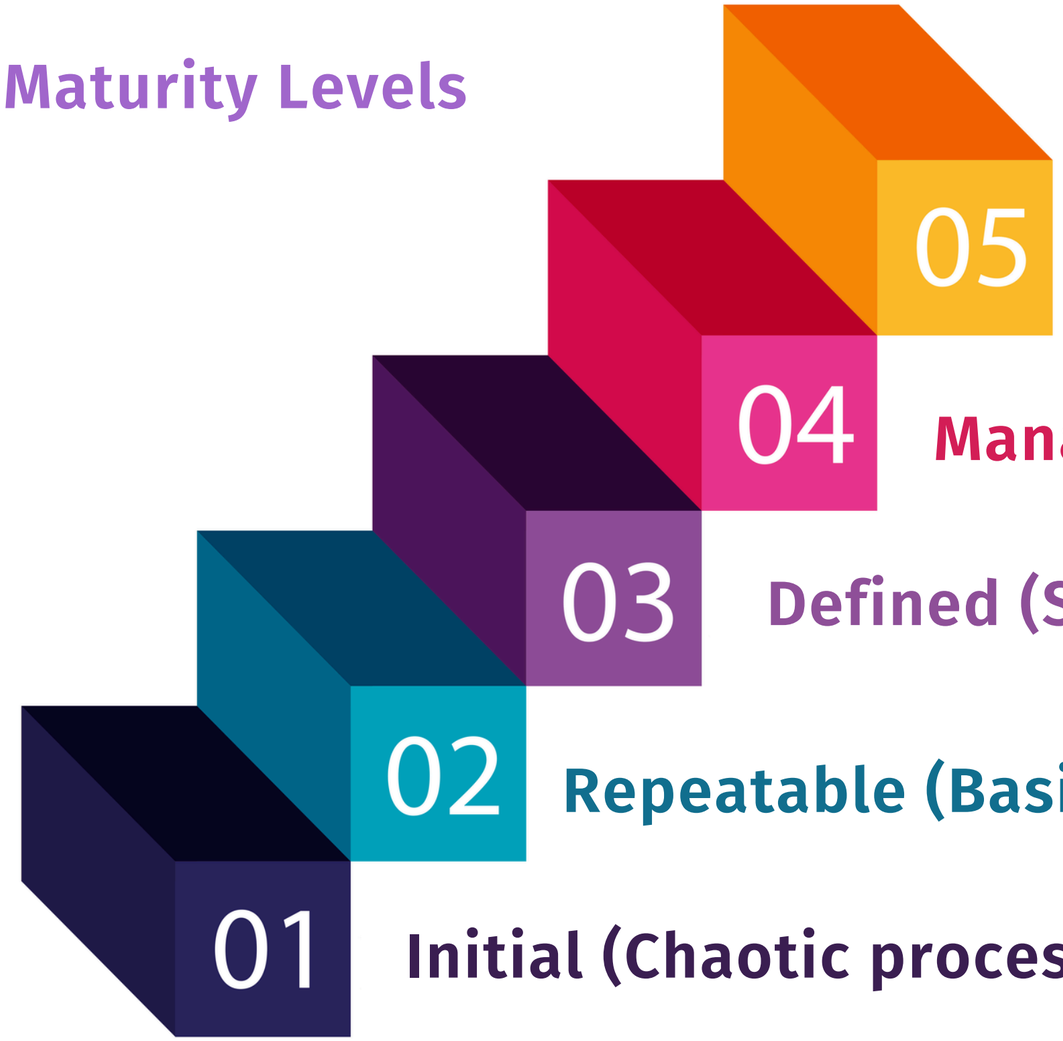
- Unpredictable schedules - Projects frequently run late
- Budget overruns - Costs spiral out of control
- Quality issues - Bugs discovered late in development
- Dependency on heroes - Success relies on key individuals
- Repeated mistakes - Same problems occur across projects

The Solution

CMM provides a **roadmap** for organizations to systematically improve their software development processes

CMM Structure Overview

Five Maturity Levels



05 Optimizing (Continuous improvement)

04 Managed (Measured and controlled)

03 Defined (Standardized processes)

02 Repeatable (Basic project management)

01 Initial (Chaotic processes)

Level 1 - Initial (Chaotic)

Characteristics

- Ad-hoc processes
- Unpredictable results
- Reactive approach
- Hero-dependent

Typical Problems

- Frequent schedule slips
- Budget overruns
- Poor quality software
- High stress environment
- Customer dissatisfaction

Example Scenario

The team starts coding immediately after getting requirements. No planning, no documentation. When bugs appear, everyone works overtime to fix them. Success depends on the star programmer.

Level 2 - Repeatable

Key Achievement:
Basic project management processes are established

Process Areas

- Requirements Management
- Project Planning
- Project Monitoring
- Configuration Management
- Quality Assurance

Characteristics

- Can repeat past successes
- Basic project discipline
- Established policies and procedures
- Management oversight

Example Scenario

The team now creates project plans, tracks requirements, and monitors progress. They can deliver similar projects successfully because they follow basic management practices.

Level 3 - Defined

Key Achievement:
Standard processes are defined organization-wide

Process Areas

- Organization Process Focus
- Organization Process Definition
- Training Program
- Integrated Software Management
- Software Product Engineering
- Peer Reviews

Characteristics

- Well-understood processes
- Consistent implementation across projects
- Training programs in place
- Process documentation

Example Scenario

All teams follow the same development methodology. New team members receive proper training. Code reviews are standard practice. Everyone knows what to do and how to do it.

Level 4 - Managed

Key Achievement:
Processes are controlled using measurement and analysis

Process Areas

- Quantitative Process Management
- Software Quality Management

Key Metrics Examples
- Defect density
- Productivity rates
- Review efficiency

Characteristics

- Detailed metrics collection
- Statistical process control
- Predictable performance
- Data-driven decisions
- Process performance baselines

Example Scenario

The organization collects detailed metrics on all projects. They can predict with 90% accuracy how long a project will take and how many defects to expect. Decisions are based on data, not gut feelings.

Level 5 - Optimizing (Continuously Improving)

Key Achievement:
Continuous process improvement through innovation

Process Areas

- Defect Prevention
- Technology Change Management
- Process Change Management

Characteristics

- Proactive improvement culture
- Innovation encouraged
- Pilot programs for new technologies
- Organization learns from experience
- Feedback loops for improvement

Example Scenario

The organization constantly looks for ways to improve. They pilot new development tools, experiment with different methodologies, and share lessons learned across all teams. Improvement is everyone's responsibility.

Benefits of CMM Implementation

Quality Improvements	• 50-95% reduction in defect density • Earlier defect detection • More reliable software
Schedule Improvements	• 15-20% reduction in development time • More predictable delivery dates • Better project planning
Cost Benefits	• 20-25% reduction in development costs • Lower maintenance costs • Reduced rework
Business Benefits	• Higher customer satisfaction • Competitive advantage • Better risk management • Improved employee morale

Activity

Below are five software team scenarios (A–E). Each scenario describes a company's software development behavior.
Your task is to match each scenario to the correct Capability Maturity Model (CMM) level.

A. The team collects and analyzes metrics on defect rates and productivity. Project decisions are made based on measurable data, and performance is tracked closely.

B. The company does not follow any formal development processes. Projects are chaotic, deadlines are missed, and success often depends on individual effort and heroics.

C. Teams follow a well-documented software development process that is standardized across the entire organization. Training is given to ensure consistency.

D. The organization regularly reviews and improves its processes using feedback, metrics, and industry best practices. Continuous improvement is a key focus.

E. Basic project management practices are in place. Project timelines and costs are tracked. Previous successes are repeatable, but processes are not yet standardized organization-wide.

CMM vs Modern Approaches

CMM Limitations

- Heavy documentation requirements
- Long implementation timeline
- Rigid structure
- Focus on process over people

Modern Alternatives

- **CMMI (Capability Maturity Model Integration)** - Evolution of CMM
- **Agile methodologies** - Emphasis on individuals and interactions
- **DevOps practices** - Integration of development and operations
- **Lean software development** - Eliminate waste, amplify learning

What is CMMI?

Capability Maturity Model Integration (CMMI)

CMMI is the improved and modern version of CMM.

It was developed because organizations needed:

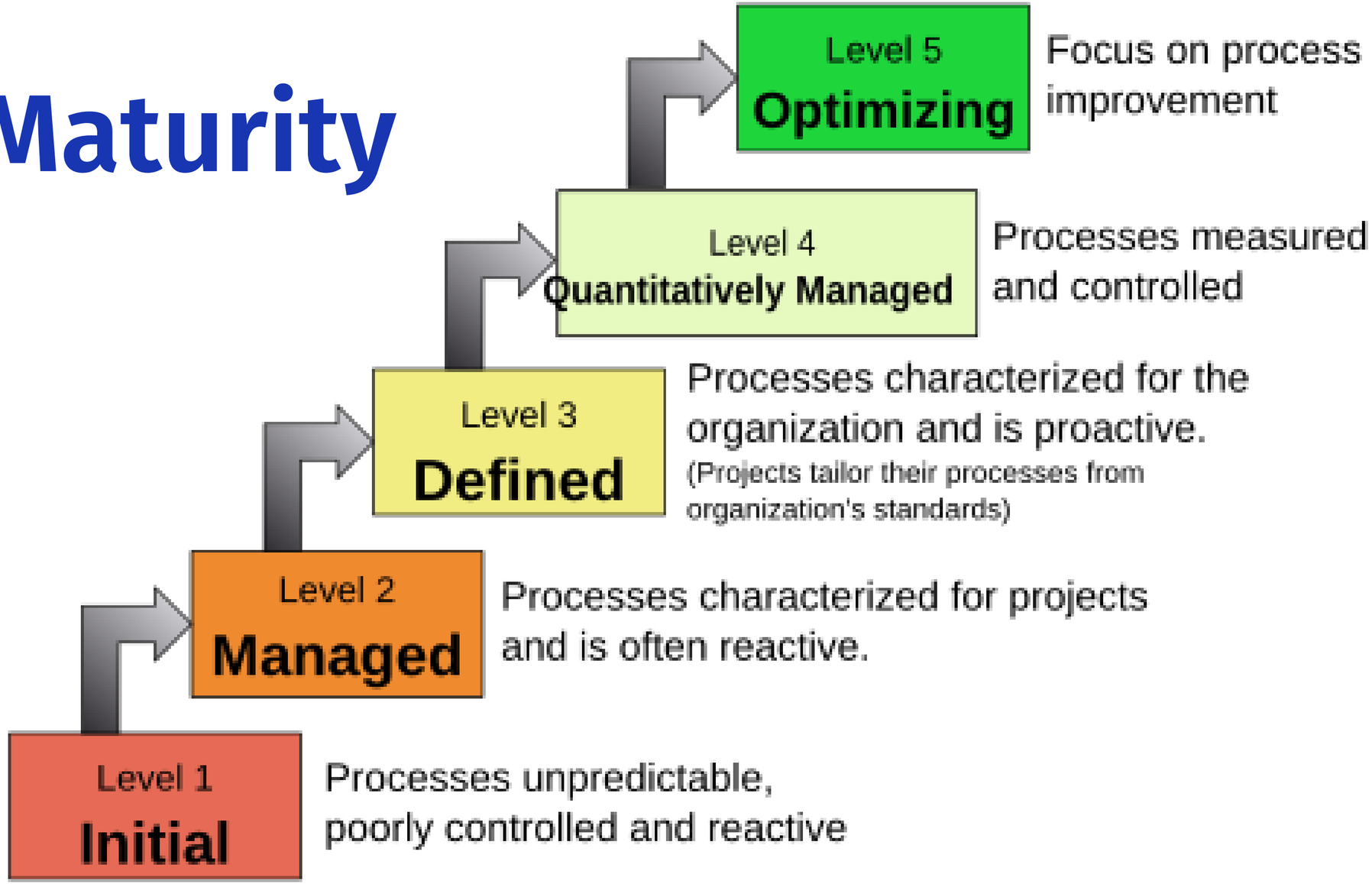
- integration of multiple process models,
- broader coverage beyond software,
- better performance management.

What Does CMMI Cover?

CMMI is not limited to software only.
It can be used in:

- software engineering,
- systems engineering,
- product development,
- service management,
- supplier management.

CMMI Maturity Levels



Difference Between CMM and CMMI

Aspect	CMM	CMMI
Developed For	Mainly software development	Multiple business/process areas
Scope	Narrower	Broader
Approach	Separate models	Integrated framework
Flexibility	Less flexible	More flexible
Modern Usage	Mostly outdated	Widely used today
Focus	Process maturity	Process improvement + integration

Personal Software Process (PSP)

What is Personal Software Process (PSP)?

PSP is a disciplined, data-driven methodology that helps individual software engineers improve their personal development practices

- **Developer:** Watts Humphrey at Software Engineering Institute (SEI)

Core Philosophy

"To consistently produce quality software, you must first understand and improve your own work habits"

Why Do We Need PSP?

Common Developer Problems

- Poor estimation
- Unpredictable quality
- Repeated mistakes
- No learning from experience
- Reactive approach

PSP Solution

- Self-awareness through measurement
- Systematic approach to development
- Data-driven improvement
- Preventive quality practices

PSP Framework Activities Overview

The Five Framework Activities

1. **Planning** - Know what you're going to do
2. **High-Level Design** - Think before you code
3. **Design Review** - Catch problems early
4. **Development** - Build with discipline
5. **Postmortem** - Learn from experience

Key Principle

Each activity builds on the previous one and provides input to the next

Activity 1 - Planning

Create a realistic plan based on your historical data and current requirements

What You Do in Planning

- Estimate software size (lines of code, function points)
- Predict development time using your productivity data
- Identify major tasks and milestones
- Create a schedule with realistic deadlines
- Plan quality activities (reviews, testing)

"Failing to plan is planning to fail"

Good planning prevents most project problems

Activity 2 - High-Level Design

Think through the solution architecture before starting to code

What You Do in High-Level Design

- Define major components and their relationships
- Identify key algorithms and data structures
- Plan user interfaces and system interfaces
- Make architectural decisions and document rationale
- Create design documentation to guide implementation

Activity 3 - High-Level Design Review

Systematically examine your design to find and fix problems before coding

What You Do in Design Review

- Check completeness - Does design address all requirements?
- Look for logical flaws - Are there gaps or inconsistencies?
- Verify component interactions - Do interfaces make sense?
- Consider alternatives - Could design be improved or simplified?
- Update design based on issues found

Activity 4 - Development

Implement the design using disciplined coding and quality practices

Development Sub-Activities

- Detailed Design - Flesh out component specifics
- Coding - Implement design in programming language
- Code Review - Examine code for defects and improvements
- Compilation - Build software and fix compile errors
- Unit Testing - Test individual components thoroughly

Quality Practices During Development

- Follow coding standards consistently
- Conduct personal code reviews before testing
- Use systematic testing approaches
- Track defects and fix them immediately
- Measure progress against your plan

Activity 5 - Postmortem

Analyze what happened and learn lessons for continuous improvement

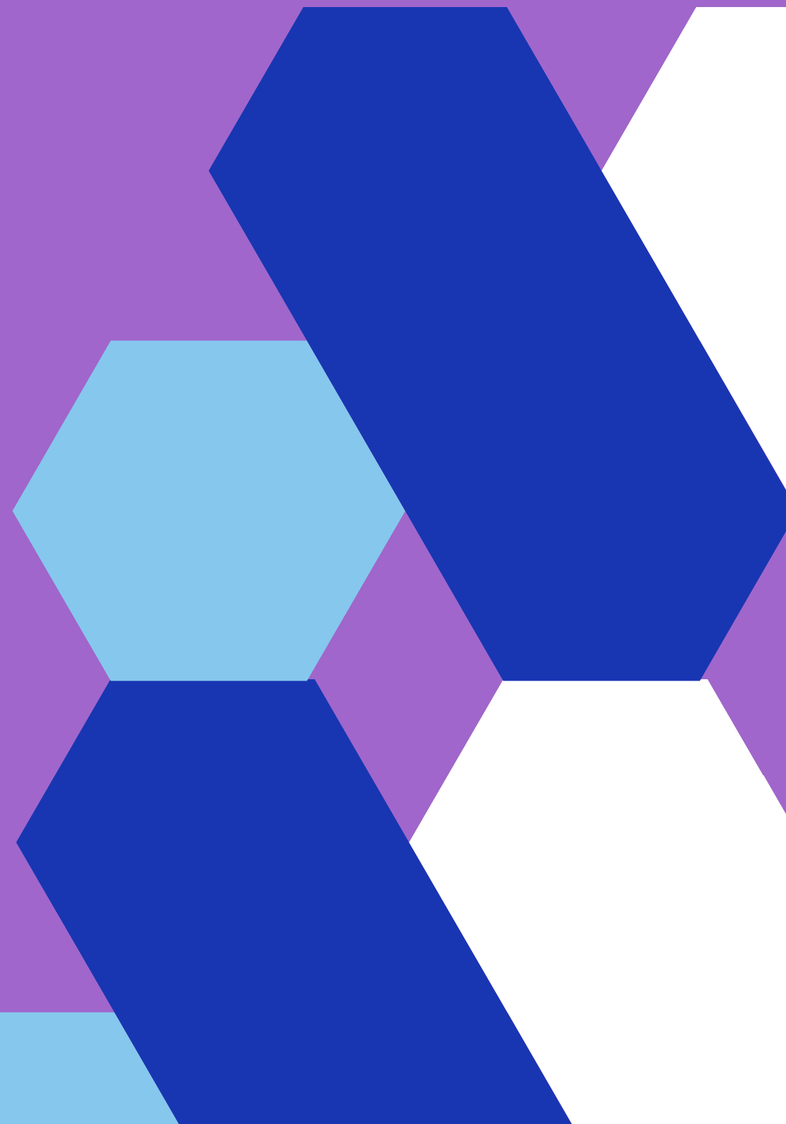
What You Do in Postmortem

- Compare actual vs. planned time, size, and schedule
- Analyze defect data - Where did defects come from?
- Evaluate process performance - What worked well/poorly?
- Identify root causes of problems
- Plan improvements for next project

Postmortem Benefits

- Continuous learning from each project
- Improved estimation accuracy over time
- Better understanding of personal work patterns
- Systematic improvement rather than random changes

Team Software Process (TSP)



What is Team Software Process (TSP)?

TSP is a structured methodology that extends Personal Software Process (PSP) principles to team-based software development.

Purpose:

- Transform groups of individual developers into high-performing teams
- Provide disciplined processes for planning, tracking, and managing software projects
- Address common causes of software project failure

Target Teams

- Pure software teams or integrated product teams
- 3 to 20 engineers
- Teams that want to be self-directed

Why Do We Need TSP?

Most software projects fail due to management issues, not technical problems

Common Software Project Problems

- Poor Planning
- Unclear Responsibilities
- Quality Issues
- Lack of Tracking
- Communication Gaps

TSP Solution Approach

- Self-managing teams instead of micromanagement
- Data-driven decisions based on actual project metrics
- Quality focus - prevent defects rather than just find them
- Disciplined processes with defined standards and practices

Core TSP Principles

Self-Managing Teams

- Teams take ownership of their commitments
- Managers provide support rather than detailed direction
- Decisions made by those closest to the work

Data-Driven Decisions

- Collect detailed metrics (time, defects, size estimates)
- Use historical data to improve future planning
- Track actual vs. planned performance continuously

Quality First

- Focus on defect prevention, not just detection
- Regular reviews and inspections
- Quality gates at each development phase

Continuous Improvement

- Regular retrospectives and process adjustments
- Learn from each project cycle
- Adapt practices based on team experience

TSP Team Launch Process

Launch Activities Overview

- Goals and Roles - Understand objectives and assign responsibilities
- Requirements Analysis - Review and clarify what needs to be built
- Risk Assessment - Identify and plan for potential problems
- Estimation and Planning - Create detailed project plans
- Team Commitment - Negotiate and commit to deliverables

PSP & TSP Core Measures

1. **Effort**
   - Time on task tracking
   - Understanding how long activities actually take
2. **Size**
   - Based on defined standard unit (lines of code, function points, etc.)
   - Consistent counting standard across projects
3. **Defect**
   - Based on defined standard defect types
   - Systematic defect recording and analysis

Key Takeaways

PSP Core Message

Personal improvement and discipline are the foundation of professional software development

TSP Core Message

High-performance teams require structured approaches, not just good individual developers

Essential Principles

- Measure your work to understand and improve
- Plan systematically rather than just start coding
- Focus on quality throughout development
- Learn continuously from each project
- Work effectively in teams through defined processes

That's a wrap!

Thank you for participating.

May 25, 2026