Software Process Improvement

V. Paúl Pauca

Department of Computer Science
Wake Forest University

CSC 331-631
Fall 2013
Management of the software process identified as important economic concern

Growing pains: avoid project failures, stay within budget

1980s: The USAF funds the Software Engineering Institute (SEI) at Carnegie Mellon to address these issues

SEI develops a process maturity framework – used by DoD to evaluate software contractors
Key contribution of SEI: development of the capability maturity model (CMM) initiative.

Insight: organizations mature processes in stages, staged evolution to software practices

Related efforts include the ISO 9000-series standards of the International Organization for Standardization, and ISO/IEC 15504, an international software development improvement initiative.
Capability Maturity Model (CMM)

Overview

- A set of strategies for improving the software process.
- Not a life-cycle model.

CMM developed for various different aspects:

- SW-CMM for software
- P-CMM for human resources
- SE-CMM for systems engineering
- IPD-CMM for integrated product development
- SA-CMM for software acquisition

- These strategies are unified into CMMI (capability maturity model integration).
A strategy for improving the software process, developed in 1986 by W. Humphrey (SEI).

Fundamental premise

- Use of new software techniques \(\neq\) increased productivity and profitability.
- **Management** of the software process is the key underlying problem.
Fundamental strategy

- Induce change *incrementally* from one level of *maturity* to another.

- *Maturity* is a *measure* of the goodness of the process itself.

- Five levels of maturity are defined:
  - Level 1. Initial level
  - Level 2. Repeatable level
  - Level 3. Defined level
  - Level 4. Managed level
  - Level 5. Optimizing level

- An organization advances from level to level over time.
Level 1. Initial Level

- Ad hoc approach to software engineering management
- Time and cost overruns
- Unpredictability in the entire software process
- Crisis oriented development rather than planned development
- Lack of measurements

- Most organizations world-wide are at level 1
Level 2. Repeatable Level

- Use of basic software management
- Planning and management based on experience with similar products
- Use of various measurements to aid cost and duration estimation
- Identification and correction of problems
- Use of measurement data from previous projects
Level 3. Defined Level

- Fully documented software process
- Clearly defined managerial and technical aspects
- Continuous effort to improve quality and productivity
- Improve/focus on software quality
- Usage of computer-aided software engineering (CASE) tools, e.g. configuration control, data modeling, refactoring, source code generation, UML, etc.
Level 4. Managed Level

- Set quality and productivity goals for each project
- Continual monitoring of quality and productivity
- Measurement and correction of process
- Use of statistical quality controls
Level 5. Optimizing Level

- Continuous process improvement
- Statistical quality and process controls used for guidance
- Feedback of knowledge from each project to the next
Reaching the next maturity level

- SEI highlights a series of **key process areas (KPA)** to reach the next maturity level:

1. **Initial level:**
   - Ad hoc process
   - Not applicable
2. **Repeatable level:**
   - Basic project management
   - Requirements management
   - Software project planning
   - Software project tracking and oversight
   - Software subcontract management
   - Software quality assurance
   - Software configuration management
3. **Defined level:**
   - Process definition
   - Organization process focus
   - Organization process definition
   - Training program
   - Integrated software management
   - Software project engineering
   - Intergroup coordination
   - Peer reviews
4. **Managed level:**
   - Process measurement
   - Quantitative process management
   - Software quality management
5. **Optimizing level:**
   - Process control
   - Defect prevention
   - Technology change management
   - Process change management

---

WFU

Object Oriented Software Engineering
Remarks

- 3 to 5 years to go from level 1 to level 2.
- 1.5 to 3 years to go from level 2 to level 3.


DOD followed with similar directives.

Many companies worldwide not associated with the military have committed to SW-CMM compliance.
CMMI History I

- CMM developed from 1987 to 1997
- CMMI v.1.1 released in 2002
- CMMI v.1.2 released in 2006
- CMMI v.1.3 released in 2010 (support of agile software development)
Five related standards applicable to a variety of industrial activities, i.e. design, development, production, installation, and servicing

ISO 9001 is most applicable to software

Features

- Documentation of the process in words and pictures
- Adherence to standards does not guarantee high-quality product, only reduces risk of poor-quality product
- Management commitment to quality, intensive worker training, goals for continual quality improvement
Remarks

- Adopted by over 60 countries, including US, EU, Japan, Canada, etc.
- Must be ISO 9000 compliant to do business with international clients
Some examples

- Hughes Aircraft (Fullerton, CA) spent $500K (1987-90) moving from level 2 to 3.
  - Resulting savings estimated at $2M / year

- Equipment Division at Raytheon moved from level 1 in 1988 to level 3 in 1993
  - Productivity doubled
  - Return of $7.70 per dollar invested in process improvement
Tata Consultancy Services (India) used ISO 9000 and CMM (1996-2000)
- Errors in estimation decreased from 50% to 15%
- Effectiveness of reviews increased from 40% to 80%
- Effort devoted to reworking projects dropped from 12% to <6%

Motorola GED has used CMM since 1992 with CMM level from 1 to 5, resulting in
- Decrease in relative duration of software projects
- Higher quality of software
- Higher productivity
Software Quality

- Definition?
Software Quality

**Definition?**

- **Functional**: How well it conforms to a given design and specifications
- **Structural**: How it meets non-functional requirements
Software Quality

- **Definition?**
  - **Functional**: How well it conforms to a given design and specifications
  - **Structural**: How it meets non-functional requirements

- **How to measure?**
  - Measurable attributes
  - Desirable characteristics
Software Quality Measurement

**Application Architecture Standards**
- Multilayer design compliance (UI vs App Domain vs Infrastructure/Data)
- Data access performance
- Coupling Ratios
- Component (or pattern) reuse ratios

**Coding Practices**
- Error/exception handling (all layers UI/Logic/data)
- If applicable - compliance with OO and structured programming practices
- Secure controls (access to system functions, access controls to programs)

**Complexity**
- Transaction
- Algorithms
- Programming practices (e.g., use of polymorphism, dynamic instantiation)
- Dirty programming (dead code, empty code...)

**Documentation**
- Code readability and structuredness
- Architecture, program, and code-level documentation ratios
- Source code file organization

**Portability**
- Hardware, OS, and Software component and DB dependency levels

**Technical and Functional Volumes**
- # LOC per technology, # of artifacts, files
- Function points - Adherence to specifications (IFPUG, Cosmic references...)

**Reliability**
- Security
- Efficiency
- Maintainability
- Size
Analysis of Quality Attributes

- **Number of Critical programming errors:**
  - Reliability:
    - Uninitialized variables, null pointers, etc
    - Error management in insert, update, delete, create, select functions
    - Thread safe applications
  - Efficiency:
    - Network traffic, non-index DB access
  - Security:
    - data access w/o error management
    - return codes and error handling mechanisms
    - Input validation – SQL injection flaws
  - Maintainability:
    - Deep inheritance trees and nesting
    - Tightly coupled components
    - Ad-hoc naming conventions
Software Size

- Non-trivial problem
Software Size

- Non-trivial problem

- Common approaches:
  - Number of lines of code (#LOC)
  - Function points (FP): identify and weight user inputs, outputs and data stores
  - Development cost / FP
  - Delivered defects / FP
  - FP / Staff month

- Manual and cost-intensive process → Automated FP