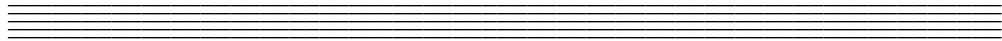


**Educational Materials**

CMU/SEI-93-EM-8

February 1993

# **Lecture Notes on Software Process Improvement**



**Laurie Honour Werth**

University of Texas at Austin

Approved for public release.  
Distribution unlimited.

**Software Engineering Institute**

Carnegie Mellon University  
Pittsburgh, Pennsylvania 15213

This document was prepared for the

SEI Joint Program Office  
ESC/AVS  
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

### **Review and Approval**

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

(Signature on file)

Thomas R. Miller, Lt Col, USAF  
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.  
This work was funded by the U.S. Department of Defense.

Copyright © 1993 Carnegie Mellon University

This document is available through the Defense Technical Information Center. DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center, Attn. FDRA, Cameron Station, Alexandria, VA 22304-6145.

Copies of this document are also available through the National Technical Information Services. For information on ordering, please contact NTIS directly: National Technical Information Services, U.S. Department of Commerce, Springfield, VA 22161.

Copies of this document are also available from Research Access, Inc., 3400 Forbes Avenue, Suite 302, Pittsburgh, PA 15213.

Use of any trademarks in this report is not intended in any way to infringe on the rights of the trademark holder.

# Table of Contents

<b>Preface</b>	<b>iii</b>
<b>1. Overview of Software Process and Quality Improvement</b>	<b>1</b>
1.1. Fundamental Process and Process Management Concepts	1
1.2. Historical Background	3
<b>2. The Software Engineering Institute Capability Maturity Model</b>	<b>5</b>
2.1. Uses of the CMM	5
<b>3. Capability Maturity Model Practices</b>	<b>8</b>
<b>4. Components of the CMM</b>	<b>9</b>
4.1. Maturity Level	10
4.2. Key Process Areas	11
4.3. Key Practices	12
4.4. Maturity Questionnaire	12
<b>5. Conclusions</b>	<b>13</b>
<b>Bibliography</b>	<b>15</b>
<b>Appendix A: Classroom Experiences with Software Process</b>	<b>19</b>
<b>Appendix B: Software Process Assessment Questionnaire</b>	<b>23</b>
<b>Attachments</b>	<b>31</b>



## List of Figures

<b>Figure 1. Five levels of software process maturity [Weber91]</b>	<b>2</b>
<b>Figure 2. Common steps in SPAs and SCEs</b>	<b>6</b>
<b>Figure 3. Capability maturity model (CMM) structure</b>	<b>9</b>
<b>Figure 4. Example of CMM structure</b>	<b>10</b>
<b>Figure 5. Results of GAO survey of software contracts</b>	<b>13</b>

## List of Tables

<b>Table 1. Shewart improvement cycle [Deming86]</b>	<b>3</b>
<b>Table 2. Comparison between SPA and SCE</b>	<b>7</b>
<b>Table 3. Key process areas (KPAs) by maturity level [Paulk91]</b>	<b>11</b>



# Preface

Software process improvement is not usually covered in standard software engineering textbooks. However, because it is a topic of great interest to the software industry, both faculty and students should be familiar with it. The goal of this package is to provide the basis for an introductory 30 to 60 minute lecture on the software process and its improvement.

The material in the main body of the document is intended primarily for instructors. In conjunction with the student-oriented document described below, it provides technical information from which an instructor can prepare a lecture. Two appendices provide additional information for instructors. Appendix A is a brief description of the use of the software process material in a university software engineering class at the University of Texas at Austin. This classroom example might be used as an illustration for students to relate the process concepts to their own software development work. Appendix B presents the Software Engineering Institute's (SEI) process maturity questionnaire from which instructors and students can gain some insight into how a software process is assessed.

A document titled "Introduction to Software Process Improvement" is attached; it is intended for students. It describes the SEI *capability maturity model* (CMM), the maturity questionnaire, and SEI procedures that are based on the questionnaire: *software process assessment* and *software capability evaluation*. Instructors may photocopy this document and distribute it to students to augment their textbook. (This document is also available electronically in PostScript format via the Internet, from which students can print their own copies. For details, send a request to Internet address [education@sei.cmu.edu](mailto:education@sei.cmu.edu).)

Finally, the package contains overhead transparency masters that instructors may find useful in delivery of their lectures.

## How to Use the Materials

These materials are introductory in nature, and they assume that the student audience is a beginning software engineering class. However, these materials may also be useful to prepare a talk for computer professionals and managers, or to prepare a general professionalism talk for computer science and engineering students.

Although the main body of this package is intended for instructors, it may be distributed to students if the instructor desires. Industry audiences, for example, may want to see the additional CMM material or even the questionnaire from Appendix B. However,

many of the maturity questionnaire concepts will not be familiar to college students unless they have had work experience or they have already had this material in class, so students may become overloaded by the terminology contained in the questionnaire and other supplementary material.

In the future we hope to add advice on applying the process improvement concepts to a software engineering class project, as well as additional materials on software process metrics.

## References

The student document does not contain a bibliography or citations because we have assumed that undergraduate students are unlikely to seek additional reading unless specifically assigned by the instructor. The bibliography in the instructor's document does contain all the references. References in the student document to an author's name can usually be found under that author's name in the instructor's bibliography. The surveys of software process maturity levels mentioned on page 4 may be found in [Humphrey89b]. References for the process improvements at Hughes Aircraft, Raytheon, and NASA are in [Humphrey91], [Dion92], and [Humphrey92], respectively.

# Lecture Notes on Software Process Improvement

## 1. Overview of Software Process and Quality Improvement

Today, concern for quality has become an international movement. England requires quality programs to be certified and audited. Europe will soon offer certification for software development companies that meet the standards described in ISO 9000 (International Standards Organization number 9000) [Arter92]. Japan has awarded the Deming Prize for years [Masaaki86].

In the United States, the Department of Commerce and NASA give major awards, such as the highly coveted Malcolm Baldrige Quality Award [Garvin91], for quality improvement. Many companies have begun to implement quality improvement or total quality management (TQM) programs throughout the company, not just in software development.

### 1.1. Fundamental Process and Process Management Concepts

*Process* is a term used to describe the people, methods, and tools used to produce software products. Improving the *quality* of the product is believed to be based on improving the *process* used to develop the product. Because software is intangible and not subject to the same physical constraints as hardware and many manufacturing products, defining the software process can be difficult.

*Software engineering process* is defined as the system of all tasks and the supporting tools, standards, methods, and practices involved in the production and evolution of a software product throughout the software life cycle. Process-driven software development implies that organizational process is adapted to meet project and product quality goals. Software development should be guided by an explicit process, with environment and tools integrated to support this process. Process definition is a prerequisite to process improvement. Defined processes promote collaboration and teamwork by making activities, roles, and dependencies visible. Process management supports improvement of the defined process through measurement and feedback.

Current implementations of process management combine three steps: definition, control, and improvement [Card89]. Process *definition* provides an exact description for the work to be performed. The current process is used as a *baseline* against which changes will be compared. Process *control* works to keep significant quality parameters within some predefined limits. Process *improvement* involves analyzing problems for root

causes and working to correct them. *Product* quality is seen as a result of continuously improving *process* quality. A process is said to be under *statistical control* if its future performance can be predicted within established statistical limits [Deming86].

Watts Humphrey incorporates these concepts into a software-oriented model in his book *Managing the Software Process* [Humphrey87]. Based on work begun at IBM, this model adapts the ideas to software development, providing five *maturity levels*, shown in Figure 1, which define an effective, staged progression toward a statistically controlled software process. Humphrey's work became the foundation of SEI software process improvement and the SEI capability maturity model (CMM) [Paulk91, Paulk93].

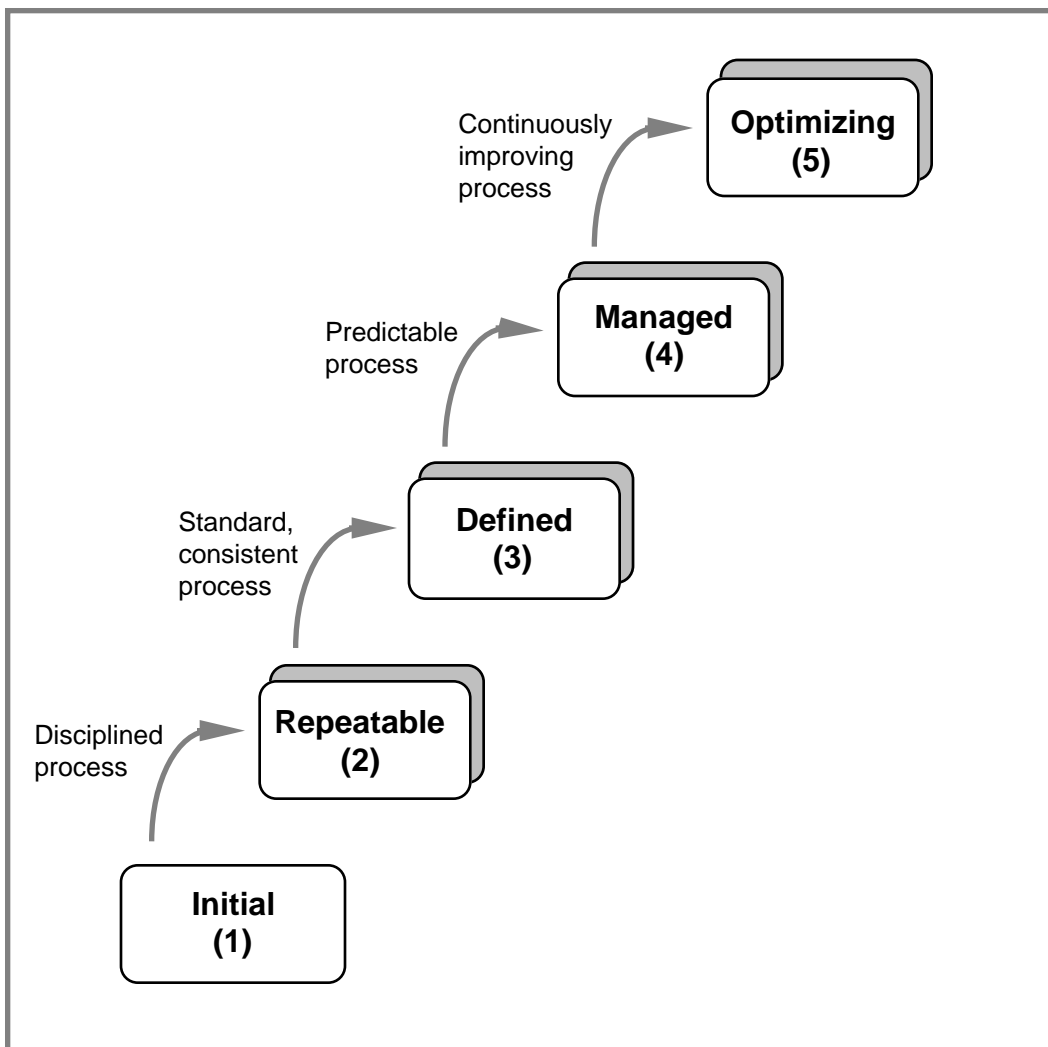


Figure 1. Five levels of software process maturity [Weber91]

The basic ideas of statistical process control have been known and practiced in other areas for at least 60 years. However, these methods represent significant change for most businesses and institutions. Resources must be provided to allow each organiza-

tion to adapt the ideas to their own environment gradually, over time. This is not a quick fix or a band-aid approach to improving software quality.

The capability maturity model defines the five stages or levels through which a company must move in order to improve process and resulting product quality. Each level of the model is the foundation for the next, so it is important not to try to move too quickly or to skip levels. Care must be taken that measures not be misused, either to evaluate individuals or to compare dissimilar projects. Still, the benefits experienced by organizations that have applied these software improvement methods have outweighed the costs. The productivity and quality improvements can help us retain our lead in the global software marketplace.

## 1.2. Historical Background

Walter Shewart, a physicist, worked at AT&T Bell Labs in statistical process control in the 1930s. W. Edwards Deming based his work on the Shewart improvement cycle (a sequence of four steps that are repeated indefinitely; see Table 1), which he successfully adapted to Japanese industry after World War II. Current Japanese management strategy continues to focus on quality improvement because the Japanese believe that the productivity and profit improvements will follow naturally. Many companies are applying the ideas of quality or process improvement across their organization. Software process improvement is the application of these concepts to software development.

---

### 1. Plan

**Define the problem**

**State improvement objectives**

### 2. Do

**Identify possible problem causes**

**Establish baselines**

**Test changes**

### 3. Check

**Collect data**

**Evaluate data**

### 4. Act

**Implement system change**

**Determine effectiveness**

---

Table 1. Shewart improvement cycle [Deming86]

Shewart's *Plan-Act-Check-Do* paradigm is the basis for the SEI process improvement program. Shewart's paradigm, applied to both the software product and process, generally consists of the following activities:

**Plan** The SEI capability maturity model is a general framework or plan for developing five increasingly improved levels (initial, repeatable, defined, managed, and optimizing) of software *process maturity*. Because the CMM is designed to be generic, each organization must customize its process improvement plan for its own application(s), environment, and company organization. The five levels are designed as a logical progression, so each level must be achieved, in order, from one to five. It is not possible to skip levels.

**Act** Because software is not produced by a manufacturing process, software designers must both strive to meet the users' functional requirements for the *product* and design for correct implementation and easy maintainability.

There will usually be many examples of *process* improvements that are needed. Efforts should focus on high-leverage points, and action plans to correct the defects must be evaluated for effectiveness. Software tools to automate and standardize the process may aid in institutionalizing improvements, but tools are not a cure-all.

**Check** Software inspections and peer reviews are the major *product* control mechanism used. Quantifiable inspections results such as change requests provide the foundation for measurable process control and improvement.

Audits are the most usual *process* verification process. Auditors need to examine not only whether the standards, procedures, and tools are adequate, but they also to see how well the project is following the prescribed process plans.

**Do** Software quality control is often specified both by the customer acceptance criteria in the contract or requirements specification and by whether the software *product* meets written standards. Software measures are used to measure product quality in a quantifiable way. The SEI has already published a core set of measures [Florac92] which can be used as a basis and enhanced by the organization as needed, though these measures are still under active development.

The most common *process* quality control approach is tracking actual against expected performance. Causes for significant deviation from the plan are found and corrected. In the later stages of the maturity model, the organization strives to actively prevent problems and errors rather than to wait to detect them in the later phases of the software development project.

## 2. The Software Engineering Institute Capability Maturity Model

The basic concept of a *maturity framework* was inspired by Crosby's quality management maturity grid and its five evolutionary stages in adopting quality practices [Crosby79]. This maturity framework was adapted for software by Radice and others at IBM [Radice85, Radice88]. Humphrey brought the maturity framework from IBM to the SEI in 1986, adding the concept of *maturity levels*. Various aspects of the maturity model are described in SEI technical reports [Fowler90, Weber91, Florac92] and Humphrey's book [Humphrey87].

Process management fundamentals are firmly grounded in science and engineering principles. They have been strongly influenced by the work on statistical process control developed by leaders such as Deming and Juran [Deming86, Juran88]. The SEI method incorporates a growing body of experience with techniques for cost and size estimation, configuration management, and other software quality improvement approaches. The field of technology transfer has evolved to the point that it now provides operative methods for helping organizations adapt to technological changes. This work has been combined to provide a foundation for learning about and improving the software development process.

The software field is young, and more modern tools and methods will evolve over time. Software organizations and applications are far too diverse to fit easily into a single process model. The SEI maturity model provides a framework as well as a method for evaluating and improving the software engineering process within an organization. The guidelines do not mandate particular methodologies, tools, or organizational structure.

### 2.1. Uses of the CMM

The capability maturity model, developed by the Software Engineering Institute, is designed to help both development organizations and customers (government organizations or companies who acquire software). Software organizations need to understand the quality of their software process and how to improve it. Organizations contracting for software need ways to evaluate a potential contractor's capability to carry out the work.

The CMM has four intended uses [Weber91] to help organizations improve their software process capabilities:

1. Identify improvements
2. Identify risks in selecting contractors
3. Implement a process improvement program
4. Guide definition and development of the software process

Over the last few years, the SEI has expanded and refined the CMM with input from many professionals from both government and industry. The current version [Paulk93a, Paulk93b] is based on several years' experience applying the model to soft-

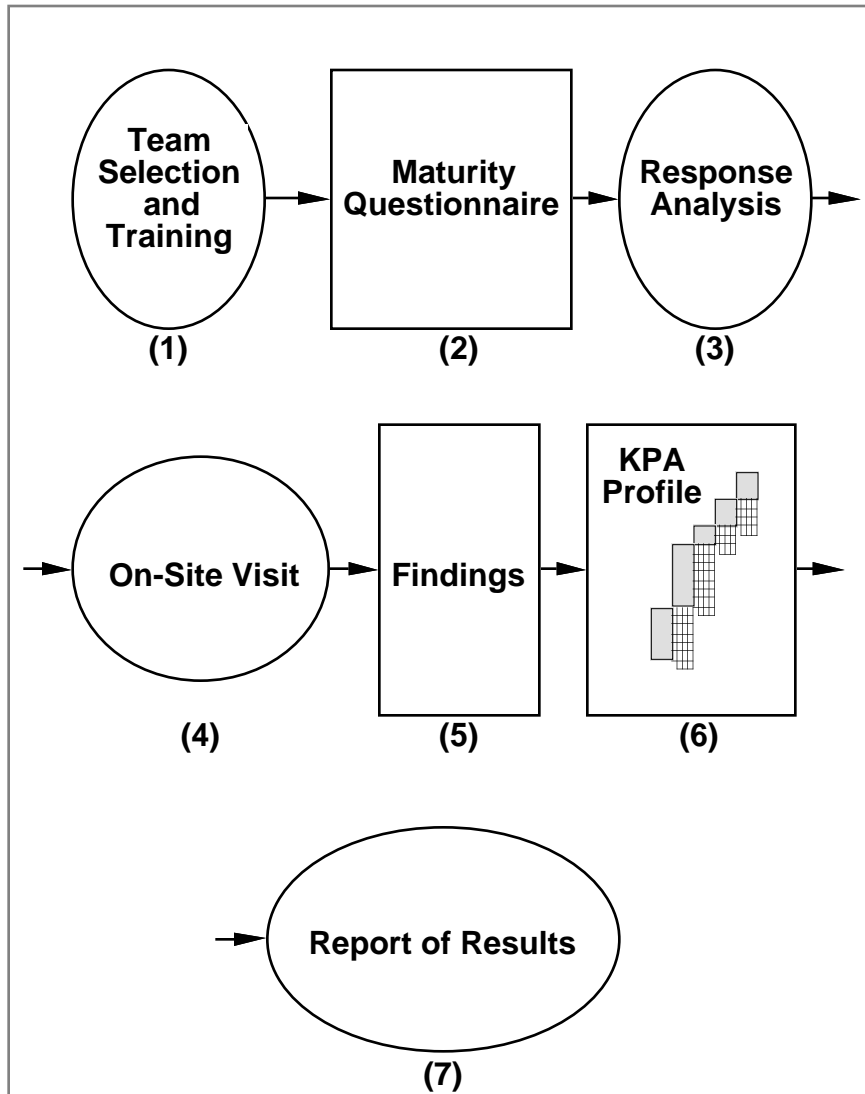


Figure 2. Common steps in SPAs and SCEs

ware process improvement [Humphrey89b]. The SEI has also developed two specific methods that apply the CMM: *software process assessment* (SPA) and *software capability evaluation* (SCE).

A software process assessment is an in-house determination, primarily of the weaknesses of the software process in an organization as a whole. It is an internal tool that an organization can choose as a part of an overall program for improving its ability to produce high-quality products on time and within budget. The objectives of the SPA method are to (1) identify strengths, weaknesses, and existing improvement activities on which to base an organization-wide improvement effort and (2) to get organizational buy-in to that effort. The method is used to help an organization identify key areas for improvement, begin to baseline its software process, and initiate improvements.

<b>SCE</b>	<b>SPA</b>
Used by acquisition organization for source selection and contract monitoring	Used by organization to improve software process
Results to the organization and the acquirer	Results to organization only
Substantiates current practice	Assesses current practice
Assesses commitment to improve	Acts as catalyst for process improvement
Analyzes contract performance potential	Provides input to improvement action plan
Independent evaluation—no organization members on team	Collaborative—organization members on team, with representative from licensed SPA associate or SEI
Applies to performance for a particular contract	Applies to organization overall, not individual projects

Table 2. Comparison between SPA and SCE

A software capability evaluation is an independent evaluation of an organization's software process as it relates to a particular acquisition. It is a tool that helps an external group (an "acquirer") determine the organization's ability to produce a particular product having high quality and to produce it on time and within budget. The objective of the SCE method is to identify strengths, weaknesses, and existing improvement activities in a supplier's software process that best indicate the risk associated with using that supplier for a particular software acquisition. The method is used to identify software risks, help in mitigating these risks, and motivate initiation of improvement programs.

The common steps in SPAs and SCEs are summarized in Figure 2. Additional similarities and differences between SPAs and SCEs are shown in Table 2 and are elaborated in [SEI92].

The SEI maturity questionnaire (see Appendix 2) is used for both SPA and SCE, but there are differences in how it is used in each method. These differences include the objectives, the make-up of the site visitation teams, the criteria for determining scope and for defining findings, and the ownership of the results. In view of these differences,

the outcomes of assessments and evaluations are not likely to be interchangeable or directly comparable. The SCE team may conclude that a particular weakness has low risk associated with it for the acquisition and therefore discount it (for example, subcontract management in an acquisition that may not involve subcontracts), while a SPA team might recommend corrective action as a high priority for the same weakness (since other projects involve subcontractors). Both views would be valid for their respective purposes.

There is considerable interest in the software community in using assessments and in the concept of continuous quality improvement that forms the basis of the methodology. The SEI has licensed independent training and consulting companies (SPA associates) to provide training and assistance in applying the SPA method to a particular environment. Software process conferences, software engineering process groups in companies, local software process improvement network (SPIN) groups, and tool support are beginning to emerge.

### 3. Capability Maturity Model Practices

The capability maturity model describes the characteristics of a mature software process. The model also shows how an immature software process can evolve into a well-managed mature one. The overall structure of the model is shown in Figure 3. Major components of the CMM, as shown in the figure, include:

- **Maturity level:** five levels or plateaus on the path to a mature software process.
- **Process capability:** capability refers to expected results, that is, what can we predict from this organization's *next* project based on their current process capability?
- **Key process areas:** a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability. These each contain common features.
- **Goals:** the high-level objectives to be achieved by the key practices for that specific key process area.
- **Key practices:** the policies, procedures, and activities that most significantly contribute to the institutionalization and implementation of the key process area.
- **Questions:** yes/no questions that sample the key practices.

Figure 4 shows an example, or a particular instantiation, of the parts of the CMM structure to illustrate the relationships among the parts. In this figure, one sees the relationship between the components (maturity levels, key process areas, key practices, and questions). At the repeatable maturity level, in the key process area of software project planning, one of the key practices is to estimate project size. Thus, a typical question from the maturity questionnaire might be "Do you use a documented procedure to estimate software size?"

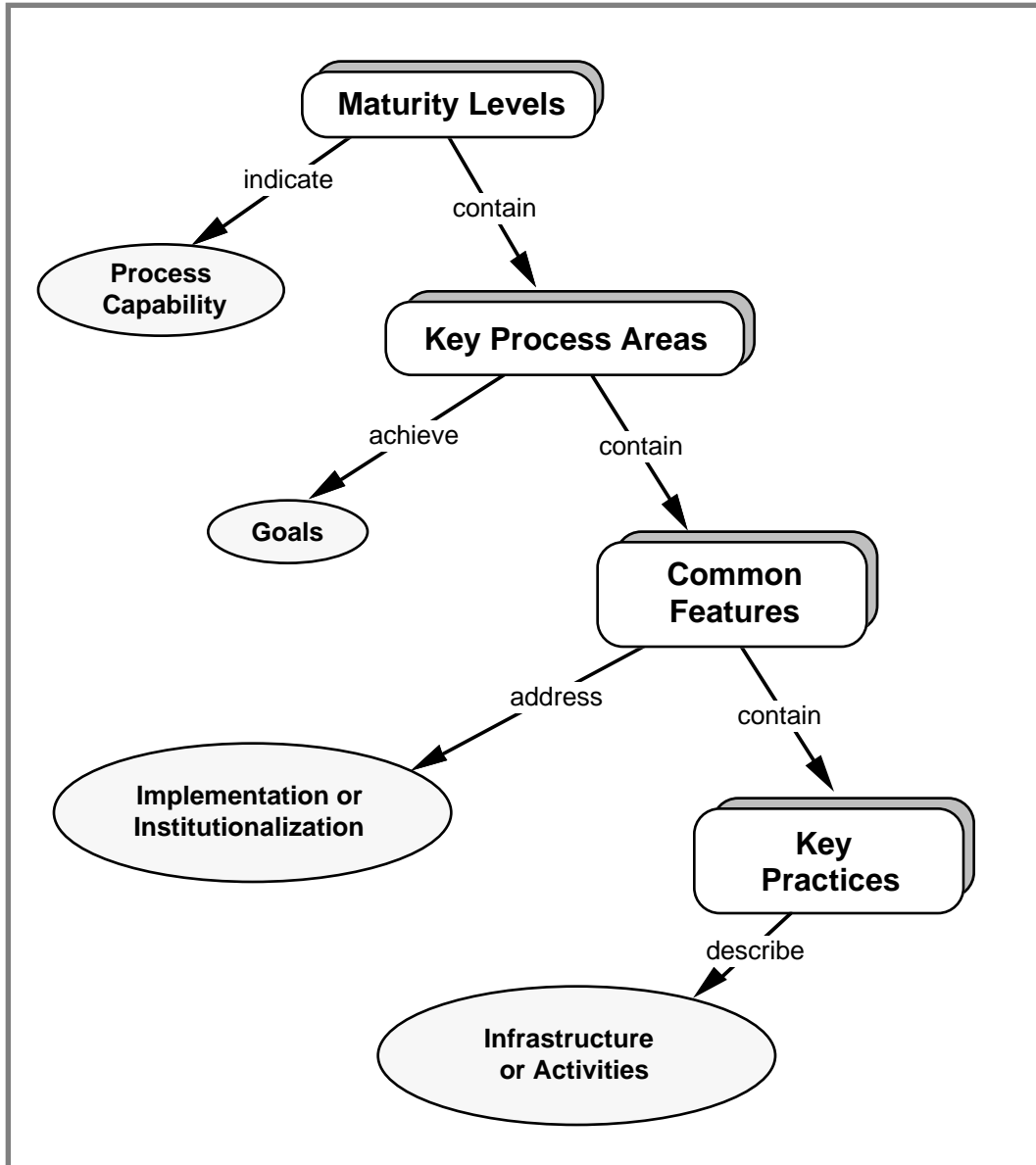


Figure 3. Capability maturity model (CMM) structure

#### 4. Components of the CMM

Each of the major components of the capability maturity model is described in more detail below. The SEI technical report, *Key Practices of the Capability Maturity Model* [Weber91], elaborates the key practices that correspond to each maturity level; these practices can be used to guide both software process improvements and capability evaluations.

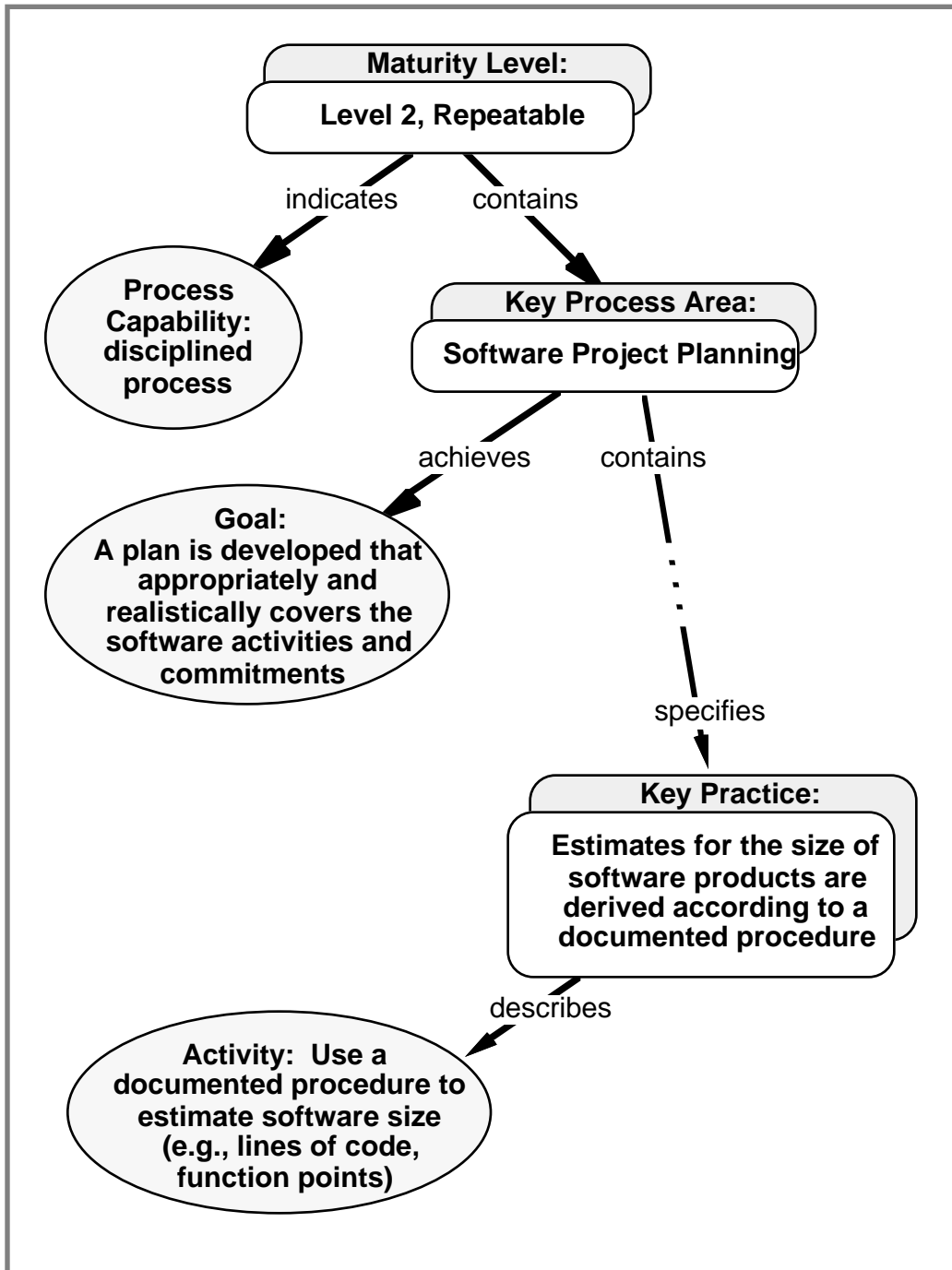


Figure 4. Example of CMM structure

#### 4.1. Maturity Level

Each *maturity level* in the CMM indicates a certain software *process capability*, describing how the software organization is expected to function: initial or ad hoc, repeatable, defined, managed, or optimizing. Each level represents an improvement in the software process. An organization's software capability can be improved by advancing through

---

<b>Level 2:</b>	<b>Repeatable</b>
Requirements Management	
Software Project Planning	
Software Project Tracking and Oversight	
Software Subcontract Management, if applicable	
Software Quality Assurance	
Software Configuration Management	
<b>Level 3:</b>	<b>Defined</b>
Organization Process Focus	
Organization Process Definition	
Training Program	
Integrated Software Management	
Software Product Engineering	
Intergroup Coordination	
Peer Reviews	
<b>Level 4:</b>	<b>Managed</b>
Process Measurement and Analysis	
Quality Management	
<b>Level 5:</b>	<b>Optimizing</b>
Defect Prevention	
Technology Innovation	
Process Change Management	

---

Table 3. Key process areas (KPAs) by maturity level [Paulk91]

these five stages or levels. Each maturity level allows management to gain a better understanding of the software process. Each level provides the foundation necessary to meet the goals of the next highest maturity level. Each maturity level has been decomposed into parts or key process areas as shown in Figures 3 and 4.

#### 4.2. Key Process Areas

*Key process areas* (KPAs) identify areas on which an organization should focus in order to improve its software development processes. Each key process area is made up of key practices that contribute to achieving the goals of the KPA. *Goals* can be used to resolve whether an organization or project has adequately implemented a key process area. Goals signify the scope, boundaries, and intent of each key process area.

Key process areas are building blocks—fundamental activities for organizations trying to improve their software process. Other process areas exist, but these were selected as particularly effective in improving process capability. Each key process area is unique

to a single maturity level. Table 3 shows the key process areas required for each maturity level. Note that there are no KPAs for the first or “initial” level.

### 4.3. Key Practices

*Key practices* are the lowest level, specific details of the CMM. Key practices define each key process area in Table 3 by specifying policies, procedures, and activities that contribute to satisfying its goal. They are a working definition of the key process area.

Key practices provide a link between the CMM and the maturity questionnaire. Specific questions relate to specific key practices. Industry experience and empirical studies were used to identify the key practices chosen by the SEI. Each key practice describes, but does not mandate, how that practice should be performed. The SEI technical report previously cited [Weber91] provides extensive definitions and guidance on the interpretation of key practices.

### 4.4. Maturity Questionnaire

The *maturity questionnaire* consists of questions about the software process that sample the practices in each key process area. (See the example question in Figure 4). All the questions used in the maturity questionnaire can be found in Appendix B.

The maturity questionnaire is a springboard for an assessment or evaluation team’s visit. The CMM provides a hierarchical structure that guides the team in investigating an organization’s software process. Answers to the questions identify process strengths and weaknesses in terms of key process areas. Questions in the maturity questionnaire are designed to determine the presence or absence of the various key practices. Questions are not open-ended, but are intended to obtain a quantified result from the following answers: yes, no, don’t know, and not applicable. A more detailed and open-ended questioning process begins after the responses to the maturity questionnaire have been analyzed.

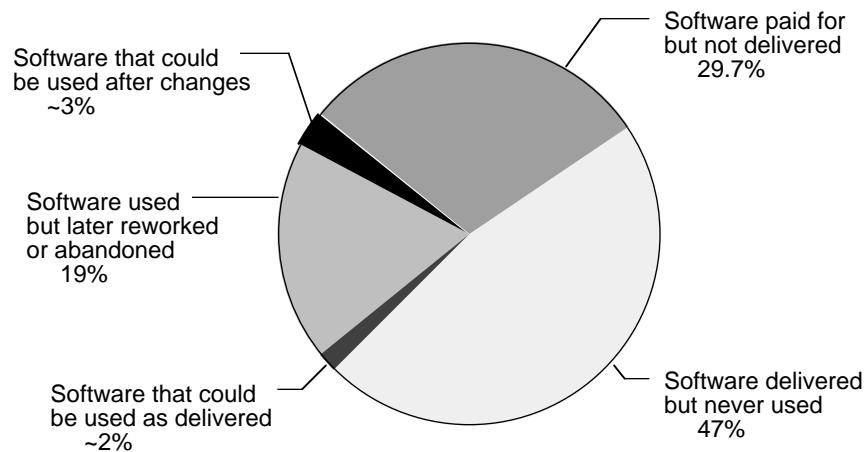
The SCE team identifies strengths, weaknesses, and improvement activities that they consider to be most relevant to performance on the acquisition contract. A group in the acquisition agency then transforms the findings into acquisition risks and/or technical ratings, which, along with other criteria, the agency can use to select a source or monitor a contract.

The SPA team also analyzes the questionnaire data to determine the current software process maturity level, identify key findings (that is, determine what will impede capability to produce quality software), and note strengths the organization can build upon. The team presents the results to senior management and, often, to the entire organization that was assessed. The team often enlists the aid of others within the organization to make recommendations for process improvement actions. An action planning group (often a software engineering process group, under the guidance of a management steering committee) develops the strategies for accomplishing long-term process improvement and determines what improvements are achievable within a specific time

frame. They work with many others in the organization to create an action plan and implement it.

## 5. Conclusions

While great strides have been made in developing software engineering methodologies and techniques, companies have been unable to consistently produce high-quality software. Stories of software problems appear on a regular basis, for example [Neumann92]. Large amounts of money have been spent on projects that have produced little usable software, as illustrated graphically in the results of a General Accounting Office (GAO) survey shown in Figure 5.



Year 1982: Nine Contracts Totalling \$6.8 Million

Figure 5. Results of GAO survey of software contracts

Successful projects have been largely based on individual or dedicated team effort rather than on software development methods [Humphrey89a]. We have come to understand that benefits of better methods and tools cannot be realized in undisciplined projects. In the typical “firefighting” mode in which immature organizations function, software quality is compromised to meet unrealistic schedules.

Process improvement ideas for software are similar to current business practices based on total quality management, popularized, beginning ten years ago, by books such as *Quality is Free* [Crosby79] and *In Search of Excellence* [Peters82]. Many of the process management and quality improvement concepts have been adapted from the work statistical process control done by W. Edwards Deming and Joseph Juran [Deming86, Juran88, Juran89]. The SEI developed the capability maturity model, based on this earlier work by quality experts, as a framework for evaluating and guiding software process improvement. As an organization increases in maturity, the difference between targeted and actual results decreases across the project. Development time and cost decrease, while productivity and quality increase. With an objective basis for measuring

quality and setting improvement priorities, time and costs become more predictable as rework and errors are removed from the system [Humphrey89a].

Companies report that improvements in work environment and motivation have turned out to be an even greater benefit than the cost saving that resulted from using the CMM [Henry92, Mays90]. (See also Table 2 in the attached student document *Introduction to Software Process*.) In a mature organization, everyone knows the processes and their own responsibilities. Workers become empowered by their involvement in developing the process descriptions and by the ability to update processes as needed. Internal processes of projects become more visible. Managers know current project status and can monitor quality and customer satisfaction.

Software process issues become even more important in the classroom, where the students are inexperienced. Using the CMM role definitions and job descriptions in classes can ease students' transition to working as a cohesive software engineering team in a professional environment. They have a better appreciation of why software development can be so difficult, and they have a meta-model for reducing this complexity. Even without applying the ideas on the class project, students can gain a clearer idea of the complications inherent in developing large software products and how they can be managed. Knowledge of software process concepts has proved helpful to students in talking to software company recruiters as well.

Knowledge transferred from the university to industry should begin to include process material in software engineering classes. The importance of software process is vital information needed to prepare computer science students for the challenges of modern software technology.

# Bibliography

- Arter92 Arter, D. "Demystifying the ISO 9000/Q90 Series Standards." *Quality Progress* (Nov. 1992).
- Brooks87 Brooks, F. P. "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer* 20, 4 (Apr. 1987): 10-19.
- Crosby79 Crosby, P. B. *Quality is Free*. New York: McGraw-Hill, 1979.
- Curtis88 Curtis, B.; Krasner, H.; & Iscoe, N. "A Field Study of the Software Design Process for Large Systems." *Comm. ACM* 31, 11 (Nov. 1988): 1268-1287.
- DeMarco82 DeMarco, T. *Controlling Software Projects: Management, Measurement and Estimation*. Yourdon Press, 1982.
- Deming86 Deming, W. E. *Out of the Crisis*. Cambridge, Mass.: MIT Center for Advanced Engineering Study, 1986.
- Dion92 Dion, R. "Elements of a Process-Improvement Program." *IEEE Software* (July 1992): 83-85.
- Florac92 Florac, W. A. *Software Quality Measurement: A Framework for Counting Problems and Defects* (Tech. Rep. CMU/SEI-92-TR-22). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1992.
- Fowler90 Fowler, P.; & Rifkin, S. *Software Engineering Process Group Guide* (Tech. Rep. CMU/SEI-90-TR-24, ADA235639). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1990.
- Garvin91 Garvin, D. "How the Baldrige Award Really Works." *Harvard Business Review* (Nov.-Dec. 1991).
- Henry92 Henry, J.; & Blasewitz, B. "Process Definition: Theory and Reality." *IEEE Software* 9, 6 (Nov. 1992): 103-105.
- Humphrey87 Humphrey, W.; & Sweet, W. *A Method for Assessing the Software Engineering Capability of Contractors* (Tech. Rep. CMU/SEI-87-TR-23, ADA187320). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1987.
- Humphrey88 Humphrey, W. "Characterizing the Software Process." *IEEE Software* 5, 2 (Mar. 1988): 73-79.
- Humphrey89a Humphrey, W. S. *Managing the Software Process*. Reading, Mass.: Addison-Wesley, 1989.

- Humphrey89b Humphrey, W.; Kitson, D.; & Kasse, T. *State of Software Engineering Practice* (Tech. Rep. CMU/SEI-89-TR-1, ADA206537). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1989. Also in *Proc. 13th Intl. Conf. Software Eng.*, Austin, Texas, 1991.
- Humphrey91a Humphrey, W.; Snyder, T.; & Willis, R. "Software Process Improvement at Hughes Aircraft." *IEEE Software* 8, 4 (July 1991): 11-23.
- Humphrey91b Humphrey, W. S. *Executive Leadership for Software*. Videotape, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., 1991.
- Humphrey92 Humphrey, W. S. *Introduction to Software Process Improvement* (Tech. Rep. CMU/SEI-92-TR-7, ADA253326). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1992.
- ISO87 ISO. *Quality Management and Quality Assurance Standards—Guidelines for Selection and Use*. 1987. Available from Global Engineering Document, Irvine, Calif.
- Juran88 *Juran's Quality Control Handbook, Fourth Ed.* Juran, J. M.; & Gryna, F. M., eds. New York: McGraw-Hill, 1988.
- Juran89 Juran, J. M. *Juran on Leadership for Quality*. New York: The Free Press, 1989.
- Masaaki86 Masaaki, I. *Kaizen: The Key to Japan's Competitive Success*. New York: McGraw-Hill, 1986.
- Mays88 Mays, R.; Jones, E.; Holloway, G.; & Studinski, D. "Experiences With Defect Prevention." *IBM Systems Journal* 29, 1 (1988): 4-32.
- Metzger83 Metzger, P. W. *Managing a Programming Project, Second Ed.* Englewood Cliffs, N. J.: Prentice-Hall, 1983.
- Neumann92 Neumann, P. G. "Risks to the Public." *Software Engineering Notes* (1992). Column in each issue.
- Paulk91 Paulk, M. C.; Curtis, B.; Chrissis, M. B.; Averill, E. L.; Bamberger, J.; Kasse, T. C.; Konrad, M.; Perdue, J. R.; Weber, C. V.; & Withey, J. V. *Capability Maturity Model for Software* (Tech. Rep. CMU/SEI-91-TR-24, ADA240603). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1991.
- Paulk92 Paulk, M.; Curtis, B.; Chrissis, M. B.; Averill, E.; Bamberger, J.; Kasse, T.; Konrad, M.; Perdue, J.; Weber, C.; & Withey, J. "The Capability Maturity Model for Software," 1-24. *Software Engineering Institute Technical Review '92*. Pittsburgh, Pa.: Software Engineering Institute, 1992.
- Paulk93a Paulk, M.; Curtis, B.; & Chrissis, M. B. *Capability Maturity Model for Software, Version 1.1* (Tech. Rep. CMU/SEI-93-TR-24). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1993.

- Paulk93b Paulk, M.; Weber, C.; et al. *Key Practices of the Capability Maturity Model, Version 1.1* (Tech. Rep. CMU/SEI-93-TR-25). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1993.
- Peters82 Peters, T. J.; & Waterman, R. H. *In Search of Excellence*. New York: Harper and Row, 1982.
- Pressman89 Pressman, R. *Getting Started in Software Engineering: A Guide to Implementing the Technology*. New York: McGraw-Hill, 1989.
- Pressman91 Pressman, R. *Software Engineering: A Practitioner's Approach*. New York: McGraw-Hill, 1991.
- Radice85a Radice, R. A.; Harding, J. T.; Munnis, P. E.; & Phillips, R. W. "A Programming Process Study." *IBM Systems Journal* 24, 2 (1985).
- Radice85b Radice, R. A.; Roth, N. K.; O'Hara, A. C., Jr.; & Ciarfella, W. A. "A Programming Process Architecture." *IBM Systems Journal* 24, 2 (1985).
- Scholtes88 Scholtes, P. *The Team Handbook: How to Use Teams to Improve Quality*. Madison, Wis.: Joiner Associates, 1988.
- SEI92 Software Engineering Institute. "SCE, SPA—Sorting It Out." *Bridge* (Dec. 1992): 1-6.
- Weber91 Weber, C. V.; Paulk, M. C.; Wise, C. J.; & Withey, J. V. *Key Practices of the Capability Maturity Model* (Tech. Rep. CMU/SEI-91-TR-25, ADA240604). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1991.



# Appendix A: Classroom Experiences with Software Process

## Introduction

Various earlier efforts by the author to incorporate software engineering techniques into a course at the University of Texas at Austin have been previously described [Werth88, Werth89, Werth90, Werth91]. Over the years, using higher level software such as MacApp, HyperCard, and Oracle in a Macintosh II laboratory, we developed tools for the software engineering class itself to provide support for testing, costing, version control, analysis and design, software process assessment, and others. A successful collaboration with a local company provided valuable experience for students using industry-strength CASE tools. In 1992 we explored the area of software process improvement, both by teaching the foundations and by applying it directly to the class project.

Our reasoning was that if software process improves the commercial software development environment, then the application of software process techniques should also strengthen the classroom environment. Two positive effects could be expected. First, successful experience with the techniques on the class project would result in students even better prepared to meet the challenges of modern software technology. Second, the use of quality improvement techniques, applied to the software engineering project as currently taught, would be a step in the direction of improving academic education as suggested, for example, by Peter Denning [Denning92].

## Course Description

Starting with Tomayko's model [Tomayko87] in the spring 1992 semester, we began to develop process plans and tools for the class project. This project provided an early design and prototype for a metrics tool to collect and analyze defect reports, as described by the Software Engineering Institute (SEI) [Florac91]. In the fall semester, students employed these plans and tools to complete the design and implementation of Dante's Defect Tracker.

The spring effort was based on disjoint subteams for design, implementation, testing and evaluation, documentation, configuration management, and quality assurance. In the fall, we integrated the process teams within the technical teams, providing a matrix management scheme. Each of the four technical teams (design, implementation, testing, and documentation) elected one person to each of the following process teams: project administration, system administration, configuration management, quality assurance, and documentation specialists. This new organization seemed to match more closely the roles that evolved during the spring semester's effort, as well as incorporate

natural liaison and communication of process procedures within the technical teams. Because of lack of experience, undergraduates find it easier to monitor and impose control when they are members of both the technical and the process teams.

Technical teams met and developed documents appropriate to their role: functional and design specifications by the design team; software product with programmer's manual by the implementation team; test plans, test cases, results, and reports by test and evaluation; and the user's manual(s) by the documentation team. Process teams developed or enhanced existing plans to describe their process function and wrote process legacies to pass along their increased understanding of their process role(s) for future teams.

Each Friday, at the status meeting held during class, one technical team presented their work, while process teams made announcements and reported progress. Everyone in the class acted as either the status meeting moderator or recorder at some point during the course. Agendas and minutes were sent by e-mail to class members. Status meetings worked very well, greatly improving communication and student process learning, as well as reducing the instructor workload. Our industry "user," Herb Krasner, attended these meetings as his schedule permitted.

Walkthroughs or reviews were held during the week, often after class on Monday or Wednesday, in preparation for the Friday presentation. Attendees included the presenting team and appropriate representatives from related technical and process teams.

The configuration control board (CCB) consisted of each team's configuration management representative, along with the head of quality assurance, the external auditor (a teaching assistant), and CEO (the instructor). Overall project management and coordination gravitated to the CCB meetings, held in the lab before class. Since other teams' members were often working in the lab during this time and all teams were represented, many questions and issues were resolved quickly and easily. Results were announced or further discussion took place immediately after the CCB meeting, during class.

## **Lessons Learned**

Student learning included the usual lessons such as the discovery of the complexity of software development, the need for communication and teamwork, and the importance of configuration management. Understanding of these issues was considerably deeper with the new course organization, however.

While this semester's class was relatively low in work experience and leadership skill, and relatively high in weak egos, significant learning took place due to the improved process structure. In fact, process improvement worked well enough to be instrumental in allowing us to bypass some of the battles between certain members of the design and implementation teams. More centralized project management may help here also.

Reducing the process learning startup time is of major importance, and several improvements are under development. We used an excellent book, *The Team Handbook*

[Scholtes89], as a supplementary process text, but additional class time needs to be used to practice the techniques. Students' skills are weak enough that teamwork cannot be learned solely from outside homework assignments. The students themselves suggested assignments, quizzes, or other means to ensure that everyone learn roles and processes early. Informal liaisons between the technical teams may need to be made explicit and enforced. Further work on developing written job descriptions and process plans will remedy some remaining deficiencies.

Technical writing and presentation skills are critical to process improvement and need to be strengthened. Some science college and departmental efforts in this direction may help. Our department's *Contemporary Issues in Computer Science* elective would be an ideal prerequisite, as it is designed to incorporate writing and presentation skills within a discussion of social, ethical, and professional issues. Relegating advanced methodology and CASE tool training to additional, probably separate, courses will allow the course to concentrate more on the project and ensure that all have the necessary technical skills, as well as making the course workload more equitable.

These techniques lead naturally to an analysis of the educational environment itself. When processes work smoothly, the underlying environment and infrastructure weaknesses become more apparent. Process improvement applied to the students' working environment identifies bottlenecks. In a time of limited resources, this is especially helpful for guiding instructors in directing their course organization efforts.

## Conclusions

As usual, the instructor learned more than the students during the semester. Having explicit process roles and duties greatly improved students' learning on the software engineering project. While it is difficult to quantify, individual process skills and experiences seem to affect the quality of the end-products in a substantive way. The technical analysis, design, and testing techniques employed are important; but the empowerment, shared learning, and more stable environment provided by quality improvement efforts seem instrumental in increasing the learning benefits of the software engineering project course. As the students observed, the whole is indeed greater the sum of the parts. Learning and applying software process can improve software engineering education by giving students a meta-model to understand and help them manage the complexities of a large software development project. Teaching and applying software process is a vital part of increasing the amount of technology transferred as students move from the classroom to industry.

## References

- Denning92      Denning, P. J. "Educating a New Engineer." *Comm. ACM* 35, 12 (Dec. 1992): 83-97.
- Florac92      Florac, W. A. *Software Quality Measurement: A Framework for Counting Problems and Defects* (Tech. Rep. CMU/SEI-92-TR-22). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1992.

- Scholtes88 Scholtes, P. *The Team Handbook: How to Use Teams to Improve Quality*. Madison, Wis.: Joiner Associates, 1988.
- Tomayko87 Tomayko, J. E. *Teaching a Project-Intensive Introduction to Software Engineering* (Tech. Rep. CMU/SEI-87-TR-20, ADA200603). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1987.
- Werth88 Werth, L. H. "Software Tools at the University: Why, What and How," 169-186. *Software Engineering Education*, Ford, G., ed. New York: Springer-Verlag, Apr. 1988.
- Werth89 Werth, L. H. "Preparing Students for Programming-in-the-Large," 37-41. *Proc. 20th SIGCSE Tech. Symp. Computer Science Education*, Barrett, R. A., Mansfield, M. J., eds. New York: ACM, Feb. 1989.
- Werth90 Werth, L. H. "Object Oriented Programming and Design Class Projects." *J. Object Oriented Programming* (Nov./Dec. 1990).
- Werth91 Werth, L. H. "Industrial-Strength CASE Tools for Software Engineering Classes," 245-256. *Software Engineering Education*, Tomayko, J. E., ed. New York: Springer-Verlag, Oct. 1991.

# Appendix B: Software Process Assessment Questionnaire

Five levels of process maturity have been defined for the assessment of software engineering organizations:

Level 1	Initial
Level 2	Repeatable
Level 3	Defined
Level 4	Managed
Level 5	Optimized

**Level 1 - Initial Process** - The initial environment has ill-defined procedures and controls. While positive responses to some of the organizational questions are likely, the organization does not consistently apply software engineering management to the process, nor does it use modern tools and technology.

**Level 2 - Repeatable Process** - At Maturity Level 2, the organization uses standard methods and practices for managing software development activities such as cost estimating, scheduling, requirements changes, code changes, and status reviews. The organization will provide positive responses to most of the following questions (\* indicates a question of greater importance in determining the CMM level).

- 1.1.1 For each project involving software development, is there a designated software manager?
- 1.1.2 Does the project software manager report directly to the project (or project development) manager?
- \*1.1.3 Does the Software Quality Assurance (SQA) function have a management reporting channel separate from the software development project management?
- \*1.1.6 Is there a software configuration control function for each project that involves software development?
- 1.2.2 Is there a required training program for all newly appointed development managers designed to familiarize them with software project management?
- 1.3.1 Is a *mechanism* used for maintaining awareness of the state-of-the-art in software engineering technology?

- \*2.1.3 Is a *formal procedure* used in the management review of each software development prior to making contractual commitments?
- 2.1.4 Is a *formal procedure* used to assure periodic management review of the status of each software development project?
- 2.1.5 Is there a *mechanism* for assuring that software subcontractors, if any, follow a disciplined software development process?
- 2.1.7 For each project, are independent audits conducted for each step of the software development *process*?
- 2.1.9 Are coding *standards* applied to each software development project?
- \*2.1.14 Is a *formal procedure* used to make estimates of software size?
- \*2.1.15 Is a *formal procedure* used to produce software development schedules?
- \*2.1.16 Are *formal procedures* applied to estimating software development cost?
- 2.1.17 Is a *mechanism used* for ensuring that the software design teams understand each software requirement?
- 2.2.1 Are software staffing profiles maintained of actual staffing versus planned staffing?
- \*2.2.2 Are profiles of software size maintained for each software configuration item, over time?
- \*2.2.4 Are statistics on software code and test errors gathered?
- 2.2.7 Are profiles maintained of actual versus planned software units designed, over time?
- 2.2.8 Are profiles maintained of actual versus planned software units completing unit testing, over time?
- 2.2.9 Are profiles maintained of actual versus planned software units integrated, over time?
- 2.2.10 Are target computer memory utilization estimates and actuals tracked?
- 2.2.11 Are target computer throughput utilization estimates and actuals tracked?
- 2.2.12 Is target computer I/O channel utilization tracked?
- 2.2.16 Are software trouble reports resulting from testing tracked to closure?

- 2.2.18 Is test progress tracked by deliverable software component and compared to the plan?
- 2.2.19 Are profiles maintained of software build/release content versus time?
- \*2.4.1 Does senior management have a *mechanism* for the regular review of the status of software development projects?
- 2.4.5 Is a *mechanism* used for regular technical interchanges with the customer?
- \*2.4.7 Do software development first-line managers sign off on their schedules and cost estimates?
- \*2.4.9 Is a *mechanism* used for controlling changes to the software requirements?
- \*2.4.17 Is a *mechanism* used for controlling changes to the code? (Who can make changes and under which circumstances?)
- 2.4.20 Is there a *mechanism* for assuring that regression testing is routinely performed?

**Level 3 - Defined Process** - At Maturity Level 3, the organization not only defines its process in terms of software engineering standards and methods, it also has made a series of organizational and methodological improvements. These specifically include design and code review, training programs for programmers and review leaders, and increased organizational focus on software engineering. A major improvement in this phase is the establishment and staffing of a software engineering process group that focuses on the software engineering process and the adequacy with which it is implemented. In addition to the questions for Level 2, organizations at Level 3 will respond “yes” to most of the following questions.

- 1.1.4 Is there a designated individual or team responsible for the control of software interfaces?
- 1.1.5 Is software system engineering represented on the system design team?
- 1.1.7 Is there a software engineering *process group function*?
- 1.2.1 Does each software developer have a private computer-supported work station/terminal?
- \*1.2.3 Is there a required software engineering training program for software developers?
- 1.2.4 Is there a required software engineering training program for first-line supervisors of software development?
- \*1.2.5 Is a formal training program required for design and code *review leaders*?

- 1.3.2 Is a *mechanism* used for evaluating technologies used by the organization versus those externally available?
- \*2.1.1 Does the software organization use a standardized and documented software development *process* on each project?
- 2.1.2 Does the standard software development *process* documentation describe the use of tools and techniques?
- 2.1.6 Are *standards* used for the content of software development files/folders?
- 2.1.8 Is a *mechanism* used for assessing existing designs and code for reuse in new applications?
- 2.1.10 Are *standards* applied to the preparation of unit test cases?
- 2.1.11 Are code maintainability *standards* applied?
- 2.1.18 Are man-machine interface *standards* applied to each appropriate software development project?
- \*2.2.3 Are statistics on software design errors gathered?
- \*2.2.15 Are the action items resulting from design reviews tracked to closure?
- \*2.2.17 Are the action items resulting from code reviews tracked to closure?
- 2.4.3 Is a *mechanism* used for identifying and resolving system engineering issues that affect software?
- 2.4.4 Is a *mechanism* used for independently calling integration and test issues to the attention of the project manager?
- \*2.4.6 Is a *mechanism* used for ensuring compliance with the software engineering *standards*?
- 2.4.8 Is a *mechanism* used for ensuring traceability between the software requirements and top-level design?
- 2.4.11 Is a *mechanism* used for ensuring traceability between the software top-level and detailed designs?
- \*2.4.12 Are internal software design reviews conducted?
- \*2.4.13 Is a *mechanism* used for controlling changes to the software design?
- 2.4.14 Is a *mechanism* used for ensuring traceability between the software detailed design and the code?

- 2.4.15 Are formal records maintained of unit (module) development progress?
- \*2.4.16 Are software code reviews conducted?
- 2.4.18 Is a *mechanism* used for configuration management of the software tools used in the development *process*?
- \*2.4.19 Is a *mechanism* used for verifying that the samples examined by Software Quality Assurance are truly representative of the work performed?
- \*2.4.21 Is there a *mechanism* for assuring the adequacy of regression testing?
- 2.4.22 Are formal test case reviews conducted?

**Level 4 - Managed Process** - At Maturity Level 4, the organization typically bases its operating decisions on quantitative process data, and conducts extensive analyses of the data gathered during software engineering reviews and tests. Tools are used increasingly to control and manage the design process as well as to support data gathering and analysis. The organization is learning to project expected errors with reasonable accuracy. In addition to questions for Levels 2 and 3, organizations at Level 4 will respond “yes” to most of the following questions.

- 1.3.3 Is a *mechanism* used for deciding when to insert new technology into the development *process*?
- \*1.3.4 Is a *mechanism* used for managing and supporting the introduction of new technologies?
- 2.1.12 Are internal design review *standards* applied?
- \*2.1.13 Are code review *standards* applied?
- \*2.2.5 Are design errors projected and compared to actuals?
- \*2.2.6 Are code and test errors projected and compared to actuals?
- \*2.2.13 Are design and code *review coverages* measured and recorded?
- \*2.2.14 Is *test coverage* measured and recorded for each phase of functional testing?
- \*2.3.1 Has a managed and controlled *process database* been established for *process metrics* data across all projects?
- \*2.3.2 Are the *review data* gathered during design reviews analyzed?
- \*2.3.3 Is the effort data from code reviews and tests analyzed to determine the likely distribution and characteristics of the errors remaining in the product?

- \*2.3.4 Are analyses of errors conducted to determine their *process* related causes?
- \*2.3.8 Is *review efficiency* analyzed for each project?
- 2.3.9 Is software productivity analyzed for *major process* steps?
- \*2.4.2 Is a *mechanism* used for periodically assessing the software engineering *process* and implementing indicated improvements?
- 2.4.10 Is there a formal management *process* for determining if the prototyping of software functions is an appropriate part of the design *process*?

**Level 5 - Optimized Process** - At Maturity Level 5, organizations have not only achieved a high degree of control over their process, they have a major focus on improving and optimizing its operation. This includes more sophisticated analyses of the error and cost data gathered during the process as well as the introducing of comprehensive error cause analysis and prevention studies.

- \*1.3.5 Is a *mechanism* used for identifying and replacing obsolete technologies?
- \*2.3.5 Is a *mechanism* used for error cause analysis?
- \*2.3.6 Are the error causes reviewed to determine the *process* changes required to prevent them?
- \*2.3.7 Is a *mechanism* used for initiating error prevention actions?

### Technology Addendum

- \*3.1 Is automated configuration control used to control and track change activity throughout the software development process?
- 3.2 Are computer tools used to assist in tracing software requirements to software design?
- 3.3 Are formal design notations such as PDL used in program design?
- 3.4 Are computer tools used to assist in tracing the software design to the code?
- \*3.5 Is the majority of product development implemented in a high-order language?
- 3.6 Are automated test input data generators used for testing?
- 3.7 Are computer tools used to measure *test coverage*?
- 3.8 Are computer tools used to track every required function and assure that it is tested/verified?

- 3.9 Are automated tools used to analyze the size and change activity in software components?
- 3.10 Are automated tools used to analyze software complexity?
- 3.11 Are automated tools used to analyze cross references between modules?
- \*3.12 Are interactive source-level debuggers used?
- \*3.13 Are the software development and maintenance personnel provided with interactive documentation facilities?
- \*3.14 Are computer tools used for tracking and reporting the status of the software in the software development library?
- 3.15 Are prototyping methods used in designing the critical performance elements of the software?
- 3.16 Are prototyping methods used in designing the critical elements of the man-machine interface?



# Attachments

Two attachments follow. The first, titled “Introduction to Software Process Improvement,” is intended as supplementary reading for students. Its pages are numbered separately from the body of this educational materials package.

The second attachment consists of nine overhead transparency masters, the contents of which are taken from the body of this document and from the student document. They include:

- Software Process Maturity Levels (Figure 1; student document Figure 1)
- Shewart Improvement Cycle (Table 1)
- Common Steps in SPAs and SCEs (Figure 2; student document Figure 2)
- Comparison Between SCE and SPA (Table 2; student document Table 3)
- Capability Maturity Model Structure (Figure 3)
- Example of CMM Structure (Figure 4)
- Key Process Areas by Maturity Level (Table 4; student document Table 1)
- Software Systems Development is Prone to Waste (Figure 5)
- On-Board Shuttle Software Improvement (student document Table 2)