

Chapter 10

Developing Software Maturity

Contents

10.1 Process Maturity: an Essential for Success	10-3
10.2 Benefits of Moving Up the Maturity Scale	10-5
10.3 How to Measure ROI	10-9
10.3.1 Maturity Models	10-11
10.3.2 Capability Maturity Model (CMM)	10-14
10.3.3 People — Capability Maturity Model (P-CMM)	10-16
10.3.3.1 P-CMM Structure	10-17
10.3.4 Software Acquisition — Capability Maturity Model (SA-CMM)....	10-18
10.3.5 Systems Engineering — Capability Maturity Model (SE-CMM)	10-20
10.3.6 ISO/IEC Maturity Standard: SPICE.....	10-22
10.3.6.1 SPICE Product Suite	10-22
10.3.6.2 Baseline Practices Guide	10-23
10.3.6.3 BPG Capability Levels	10-23
10.3.7 Common Features and Generic Practices	10-24
10.4 Lessons Learned in Implementing the Software Development	
CMM	10-26
10.5 Software Development Capability Assessment Methods	10-26
10.5.1 Software Development Capability Evaluation (SDCE)	10-28
10.5.2 Software Engineering Institute (SEI) Software Capability	
Evaluation (SCE)	10-28
10.5.3 Addressing Maturity in the Request for Proposal (RFP).....	10-29
10.6 References	10-31

10.1 Process Maturity: an Essential for Success

“The development of a weapon system requires integrating technical, administrative, and management disciplines into a cohesive, well-planned, and rigorously controlled process. As a critical component of a weapon system, software must be developed under a similarly disciplined engineering process.” [DSMC90]

The above statement identifies the need for capability and maturity in software acquisition and development — we must have a disciplined, engineering approach to software acquisition and development, while maintaining a grasp on the process and product. If we cannot achieve either of these, we have introduced unnecessary risk in meeting end customer needs within cost and schedule constraints.

What constitutes capability and maturity? Capability is the ability to do something. Maturity is the state of being fully developed. Thus, one may conclude that to have “capability maturity,” an organization would be capable of acting in a fully developed manner.

Mark C. Paulk describes the differences between immature and mature organizations in this manner:

“The immature ... organization is reactionary, and managers are usually focused on solving immediate crises (better known as fire fighting). Schedules and budgets are routinely exceeded because they are not based on realistic estimates. When hard deadlines are imposed, product functionality and quality are often compromised to meet the schedule.

In an immature organization, there is no objective basis for judging product quality or for solving product or process problems. Therefore, product quality is difficult to predict. Activities intended to enhance quality such as reviews and testing are often curtailed or eliminated when projects fall behind schedule.”

On the other hand, a mature ... organization possesses an organization-wide ability for managing ... processes. The ... process is accurately communicated to both existing staff and new employees, and work activities are carried out according to the planned process. The processes mandated are fit for use and consistent with the way the work actually gets done. These defined processes are updated when necessary, and improvements are developed through controlled pilot-tests and/or cost benefit analysis. Roles and responsibilities within the defined process are clear throughout the project and across the organization.

In a mature organization, managers monitor the quality of the ... products and customer satisfaction. There is an objective, quantitative basis for judging product quality and analyzing problems with the product and process. Schedules and budgets are based on historical performance and are realistic; the expected results for cost, schedule, functionality, and quality of the product are usually achieved. In general, a disciplined process is consistently followed because all of the participants understand the value of doing so, and the necessary infrastructure exists to support the process.” — Mark C. Paulk [PAULK93]

Notice the use of the term “process” throughout the description above. Having and using a defined process, while continuously improving that process, is directly related to capability maturity. The following are quotations from two of many authors who have commented on the importance of processes:

“Not having a system [process] is like doing every job for the first time, every time – including the accompanying errors and false starts. Without the regularity and repetition of work that characterize a system [process], it becomes much more difficult to learn because we are deprived of time-ordered process data. Without a system [process], workers, teams, and departments would act independently of one another, without synchronization or division of labor.

If we have no system [process] we have no opportunities for progress and productivity improvements.” — Kenneth T. Delavigne and J. Danie Robertson [DELAVIGNE94]

“Since I first coined the term [“reengineering”] in the late 1980s, I have consistently used the same definition for it: Reengineering is the radical redesign of business processes for dramatic improvement. Originally, I felt that the most important word in the definition was ‘radical.’ The clean sheet of paper, the breaking of assumptions, the throw-it-all-out-and-start-again flavor of reengineering – this is what I felt distinguished it from other business improvement programs.

I have now come to realize that I was wrong, that the radical character of reengineering, however important and exciting, is not its most significant aspect. The key word in the definition of reengineering is “process”: a complete end-to-end set of activities that together create value for a customer.”

[American managers] were getting nowhere because they were applying task solutions to process problems.

The difference between a task and process is the difference between part and whole. A task is a unit of work, a business activity normally performed by one person. A process, in contrast, is a related group of tasks that together create a result of value to a customer.

The problems that afflict modern organizations are not task problems. They are process problems.”
— Michael Hammer [HAMMER96]

Webster defines process as, “A series of operations performed in the making or treatment of a product, e.g., a manufacturing process.” A software process is the series of operations performed in acquiring, developing, or maintaining of a software product. A software process definition is the description of that process. The process definition guides teams of software acquisition personnel or development engineers in the performance of their work. Thus, a defined, disciplined process is one that is documented, taught, applied, measured, used by everyone in the organization, and accessible to all team members (e.g., an organization’s procedures manual). A defined process does the following:

- It provides the basis for examining and improving the process;
- It aids in establishing predictability;
- It improves understanding of roles and dependencies;
- It guides software personnel through orderly decisions,
- It provides a smooth working framework; and
- It helps staff members to readily transition from one program to another. [CLOUGH92]

Without a defined, disciplined software process, each team member's work rests on intuition, and the quality of the product on blind faith. Team members are left to arrive at their own operational processes, methods, procedures, and standards without the direction and support professionals in other disciplines consider essential (e.g., in sports, the arts, or science).

Watts Humphrey explains that a software process is the technical and management framework for applying engineering methods, tools, procedures, and people to software development, while the process definition identifies roles and specifies tasks. The definition also establishes measures and provides entry and exit criteria for every major step in the process. [HUMPHREY95]

Software acquisition and development organizations are more successful when they have processes they can effectively communicate, manage, and evolve. A well-defined process is also easier to improve. For instance, if some steps in the process are skipped, or if the process is inefficient, problems may occur. Steps, or the process itself, may not be used if the definition is poor, communication is unclear, or team members are not motivated. Improvements can be made once these problems are identified. The process, its definition, and the supporting infrastructure all evolve and mature with use and experience. [HUMPHREY95]

10.2 Benefits of Moving Up the Maturity Scale

Published studies of software engineering improvements measured by the Software Engineering Institute's Capability Maturity Model (CMM) indicate significant cost savings and return on investment (ROI). Thus, software testing and maintenance costs are decreased, because quality requirements are more readily met. Of the companies studied, a distinction is made between the one time CMM—compliance costs of achieving a higher maturity level and the cost of continuing to perform software engineering at that higher level. The latter has been found to actually represent a cost reduction when compared to software production costs at the former lower level. Some studies show that the onetime cost of achieving a higher level are quickly recouped by significant savings in producing higher-quality software that requires less rework and is easier to maintain.

All companies studied report that process improvement works best when employees and employer agree to accept the required extra effort and expense. One such arrangement is to have some meetings and training sessions conducted during the lunch hour, with the company providing lunch. Other variations on employer/employee compromises include *shared time*, when training is conducted on 50% company time and 50% employee time.

It is generally easy to quantify the benefits of increased maturity at the company level. Production costs go down—quality goes up—time to market is shortened. How employees benefit is subtler. The higher level in which an employee works, the more valuable he is to the software industry—i.e., the techniques learned are very marketable, useful professional skills. In addition,

employee pride and management respect cannot be overlooked as an employee benefit, reward, and motivating force. Those companies having achieved higher maturity levels agree that a good reputation with their customers is primarily based on product quality and agreeable customer interrelations. Higher maturity levels lead to higher quality software, and therefore, increased company reputation. It also tends to change the manner in which companies interact with their customers. For example, the formality of a higher maturity level lessens *ad hoc* contractor tendencies to give into volatile government requirements; it also contributes to more reliable, mutually-beneficial contractor-government relationships. Above all, the most compelling benefit is also the most basic one: higher quality software, at lower cost, with improved company reputation, is a powerful formula for competing, winning, and keeping contracts. [SAIEDIAN95]

In the August 1994 report, *Benefits of CMM-Based Software Process Improvement: Initial Results* (CMU/SEI-94-TR-13), the SEI collected and analyzed data from 13 organizations (both industry and Government) to obtain process improvement results of efforts associated with the CMM. Table 10-1 summarizes these results. *A 35% median productivity gain, a 19% schedule reduction, a 39% post-release defect reduction, and a 5:1 return on investment ratio bear convincing testimony of the value of process improvement.* The SEI stated that if these CMM process improvements had been combined with more robust software engineering environments, the use of automated process control tools, or the implementation of methodology improvements, the results would have been even more dramatic.

CATEGORY	RANGE	MEDIAN
Total yearly cost of SPI activities	\$49,000 to \$1,202,000	\$245,000
Years engaged in SPI	1 to 9	3.5
Cost of SPI per software engineer	\$490 to \$2,004	\$1,375
Productivity gain per year	9% to 67%	35%
Early detection gain per year (pre-test defects discovered)	6% to 25%	22%
Yearly reduction in time to market	15% to 23%	19%
Yearly reduction in post-release defect reports	10% to 94%	39%
Business value of investment in SPI (value returned on each dollar invested)	4.0:1 to 8.8:1	5.0

Table 10-1. Summary of SEI CMM Software Process Improvement (SPI) Study

In this report, quality was defined as the state of software when released or delivered to customers. The most common measure of quality among the data submitted was the number of post-release field defect reports. Figure 10-1 illustrates yearly reductions in that number. The letter values on the Y-axis are arbitrary designations for organization anonymity. The number values in parentheses on the Y-axis indicate the number of years the organization invested in software process improvement (SPI) programs. Organization P sustained a remarkable defect report reduction rate of 39% per year over a 9-year period. That rate represents successive releases with substantial amounts of new and modified code — all of which completed its entire life cycle throughout that period. Organization P's last release had no defects reported in new and modified code. Organizations S and T also experienced substantial reductions for a significant period.

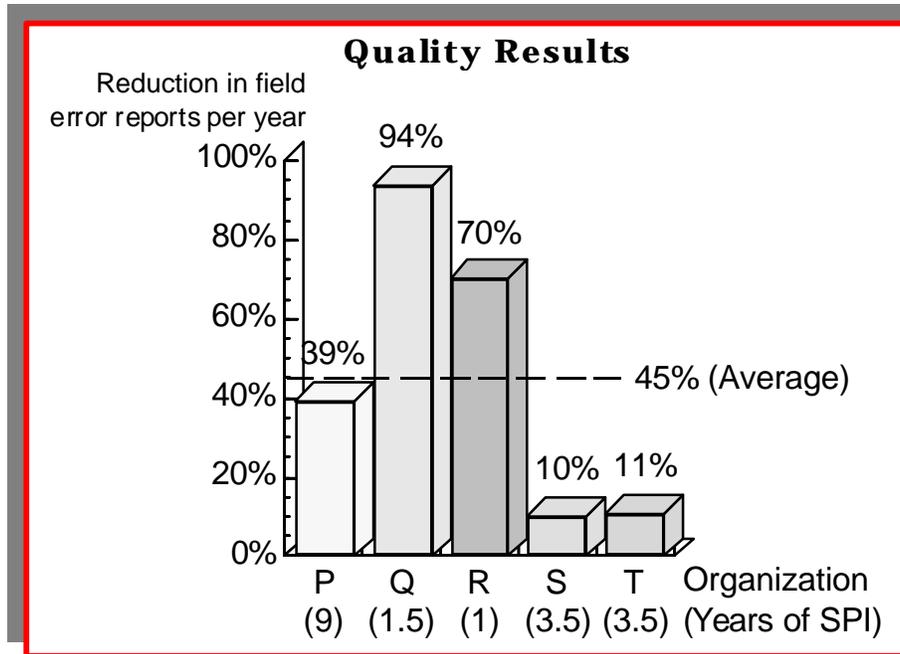


Figure 10-1. Reduction per Year in Post-release Defect Reports [SEI94]

Productivity data were gathered on lines-of-code (LOC) produced per unit of time. As illustrated in Figure 10-2, the largest gain, organization G, was based on a comparison of two programs, only one of which adopted the SPI. The superior productivity of the second program was due to clear requirements definition and management. Organization H had a large productivity gain due to a *reuse* program supported by tools and an environment adapted to promote reuse.

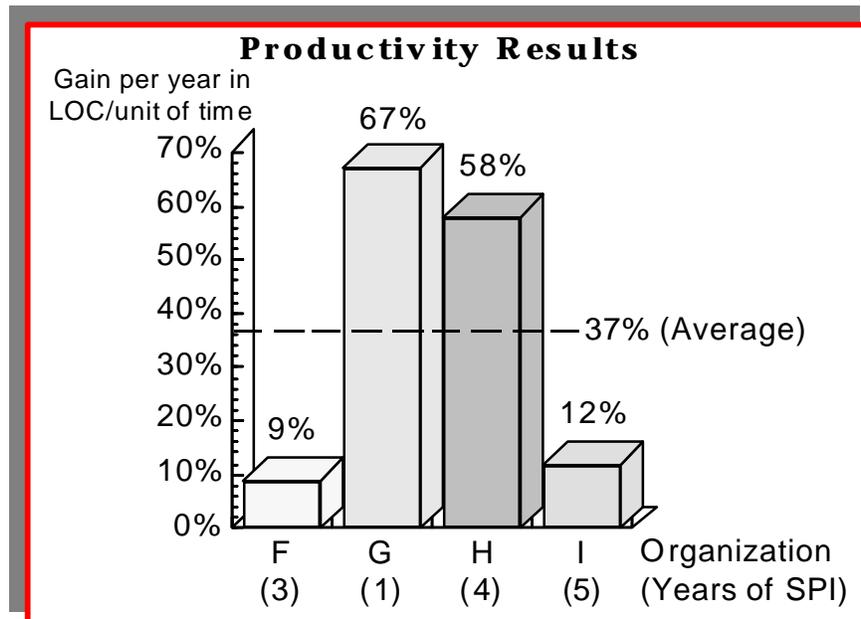


Figure 10-2. Gain per Year in Productivity [SEI94]

ROI data were reported in terms of the ratio of measured benefits to measured costs, as illustrated in Figure 10-3. Benefits included savings from productivity gains and fewer defects. The benefits did not, however, include the value of enhanced competitive position from increased quality and shorter time to market. The cost of SPI included the cost of the Software Engineering Process Group (SEPG), assessments, and training, but did not include indirect costs such as incidental staff time to put new procedures in place.

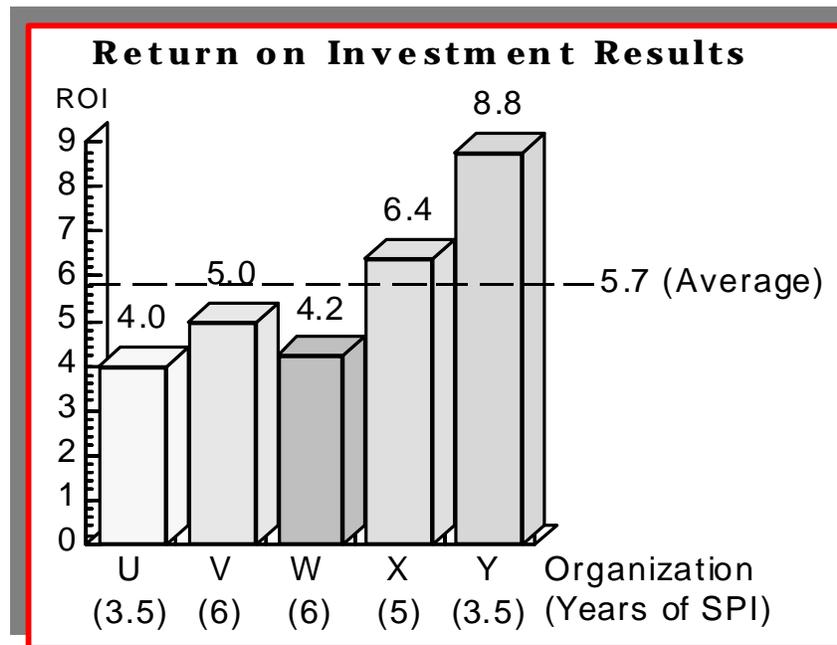


Figure 10-3. Return on Investment Ratio of SPI Efforts [SEI94]

In 1997, Karl Williams reported the following experience at Motorola based on 176 groups in 11 different countries:

- Overtime down 20X
- Released defects down 20X
- Cycle time improved 2X
- Productivity up 3.4X
- Development cost down 3X
- Schedule and cost overruns down more than 100X
- Return on investment ranging from 4X to 16X with an average of 8X. [WILLIAMS97]

John Vu of Boeing Space Transportation Systems reported the following:

- Later phase defects reduced from 31% to 4%
- Improved productivity by 62%
- Improved cycle time by 36%
- Improved customer satisfaction by more than 10%. [VU97]

Some individuals are probably thinking, “Why spend all the effort on process improvement when I can buy a few application development tools, save a lot of bucks, and be done with it.”

Researchers at IBM spent four years studying the impact of application development tools. They concluded that for teams with well-structured processes, the use of such tools enhanced the process and improved performance. However, for teams with more informal or ad hoc processes, tool use abetted chaos. [GUINAN97] Or to quote an often heard phrase, “A fool with a tool is still a fool.”

NOTE: Refer to past issues of CrossTalk (published by the Software Technology Support Center (STSC)) for timely, pertinent articles on the subject of process improvement and moving up the maturity scale.

Other sources with information regarding the benefits of software process improvement are:

- [University of Texas](#)
- [European Software Institute](#)
- [Standish Group Report](#)
- [Software Engineering Institute](#)

10.3 How to Measure ROI

Most managers will want to know what financial benefit their organization or company is reaping from their investment in process improvement. In order to obtain that information the organization must have baseline data from which to measure change. As a minimum, the cost per unit of production, such as the cost per line of code, is needed. A very mature organization using quantitative methods for process management will probably have all the data needed down to the development process phase. Most, however, will be fortunate to know their bottom line costs and quantity of software produced.

Since most organizations haven't matured to the point where quantitative methods are being used to manage their processes, an example of a relatively simple method of computing return on investment is included here. In adopting a simplified approach, several assumptions must be made to proceed. These assumptions are:

1. Any change to processes will require time to be implemented and mature to the point of effectiveness.
2. Changes made at different times, say a year apart, may each be improving productivity over the same period of time. This will make attribution of any improvement to a given change or investment very difficult.
3. Simple, end to end measures, which are easy to implement and which are needed for CMM level 2 estimating & feedback, are the best for bottom-line evaluations of ROI.
4. Since no evidence exists to the contrary, the organizational capability will be assumed to be steady state and any changes to productivity will be attributed to the investment in process improvement.
5. The benefit to the organization due to improvements in productivity can take the form of cost avoidance (reduced developmental staff) or increased capability (more software requirements met for the same cost).

6. All cost benefits can be calculated from the baseline cost per unit of production, the cost per unit of production at measurement points and the volume of units produced or expected to be produced at each measurement point.
7. Most organizations deal with more than one product line (eg. Operational Flight Programs and Automatic Test Equipment (ATE) Test Program Set development as well as bug fixes in ATE programs.) The savings realized for each product line must be calculated separately. The total savings will be the summation of the savings for each product line.
8. It is a common business practice to use a five year period to calculate the payback for investment in new equipment or processes. We will use the same period in evaluating the return on investment in process improvement.

Using the these assumptions, the savings obtained by productivity improvements in the development of a given product line for a given period would be equal to the fractional reduction in cost per unit times the volume produced for that product line in dollars adjusted for inflation. The total savings will include actual savings shown for past performance increases and projected savings five years into the future. However, the projected savings should be based on the per unit cost reduction to the present, multiplied by the volume of projected production for the next five years

In this example, the calculated ROI will be based on a sliding window of time no more than 11 years in length. The calculations are based on a period measured back no more than five years from the present and projected for 5 years into the future. For those organizations that do not have a five-year history of data from which to work, the baseline year should be the first year from which data is available. For some, that may be the current year.

In order to be consistent, all calculations of cost and savings should be based on direct labor costs, burdened only by organizational overhead. All calculations should also be done in constant year dollars calculated against the current baseline year.

The method of calculating ROI using the above logic is as follows:

$$\text{ROI} = \text{Total Savings} \div \text{Total Investment}$$

Total Savings: $S_{Total} = \sum S_{PL}$ (The sum of all product line savings) where:

Product line savings: $S_{PL} = S_{TD} + S_{Projected}$ (The sum of savings to date and projected savings)

Savings to date: $S_{TD} = \sum_{x=B+1}^P ((\frac{C_B}{C_X} - 1) * E_X)$ (The sum of savings for the product line from the first year after process improvement investments began to the present)

P = Present year

B = Baseline year

C_B = Baseline Cost per Unit of Production. i.e. The cost per unit at the beginning of process improvement

C_X = Cost per Unit of Production during year X.

E_X = Direct and organizational overhead expense of production for year X.

$$\text{Projected Savings: } S_{\text{Projected}} = \sum_{X=\text{Present}+1}^{\text{Present}+5} \left[\left(\frac{C_B}{C_P} - 1 \right) * E_X \right]$$

C_P = Present Cost per unit of Production

E_X = Projected direct and organizational overhead expense of production for each year.

Total Investment includes the sum of all costs for manpower, training and tools expended for the process improvement effort from the beginning of the calculation window to date.

10.3.1 Maturity Models

Many organizations have embarked on organizational improvement efforts that focus on software process improvement. The Capability Maturity Models (CMM) provides a means for assessing current practice and guiding process improvement efforts. *[The CMM is a registered Service Mark of Carnegie Mellon University.]* The International Standards Organization/International Electrotechnical Commission (ISO/IEC) has developed a version of the CMM framework, the Software Process Improvement Capability determination (SPICE) model. The models discussed here include:

- Capability Maturity Model (CMM) for Software ([CMU/SEI-TR-93-24](#) and [CMU/SEI-TR-93-25](#)),
- People — Capability Maturity Model (P-CMM) ([CMU/SEI 95-MM-001](#) and [CMU/SEI-95-MM-002](#)),
- Software Acquisition — Capability Maturity Model (SA-CMM) ([CMU/SEI-96-TR-20](#)),
- Systems Engineering — Capability Maturity Model (SE-CMM) ([CMU/SEI-95-MM-003](#)),
- Software Process Improvement Capability determination (SPICE) ([The ISO/IEC 15504](#)).

Other maturity models have been developed for other domains. The proliferation of models has led to the effort to integrate several domain models into a single model. The Federal Aviation Administration integrated CMM (iCMM) model combines the CMMs for Software, Software Acquisition, and Systems Engineering into a single model (available at www.faa.gov/ait/ait5/FAA-iCMM.htm). There is an ongoing effort by the Office of the Secretary of Defense, industry, and the Software Engineering Institute to create a model that combines the CMMs for Software, Systems Engineering and Integrated Product Development into a single model. This is known as the CMMi project. The models are based on one of two architectures, either a staged architecture or continuous architecture. For example the CMM for Software, People — Capability Maturity Model (P-CMM) , and Software Acquisition — Capability Maturity Model (SA-CMM) all use a

staged architecture while the Software Engineering — Capability Maturity Model (SE-CMM) and SPICE use continuous architectures. To date, a model combining the CMMs for software and system engineering has been developed in both a staged and continuous representation.

Figure 10-4 describes the architecture for the staged models. The maturity levels or stages are composed of several key process areas. Each key process area is organized into five sections called common features. The common features specify the key practices that, when collectively addressed, accomplish the goals of the key process area. The five common features are:

- Commitment to perform,
- Ability to perform,
- Activities performed,
- Measurement and Analysis, and
- Verifying implementation.

The practices in the common feature Activities Performed describe what must be implemented to establish a process capability. The other practices, taken as a whole, form the basis by which an organization institutionalize the practices described in the Activities Performed common feature. [PAULK93]

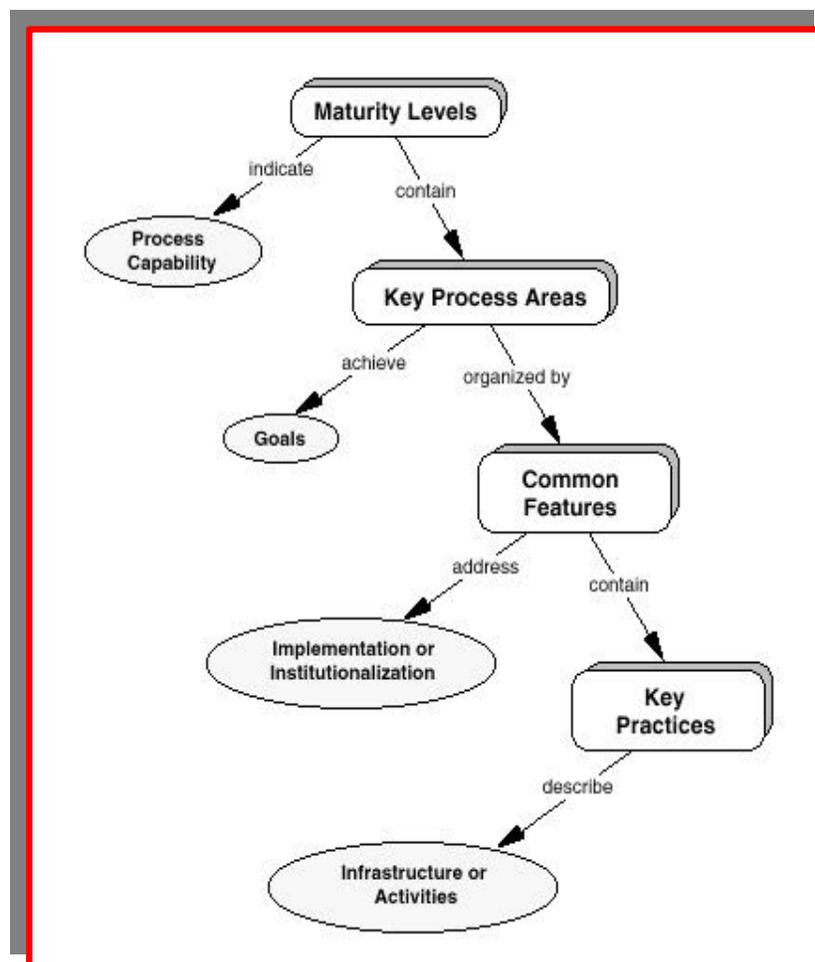


Figure 10-4. Staged Model Architecture [PAULK93]

The architecture for the Continuous Model was developed by the SPICE committee. It is based on the philosophy that improvement is a continuum, not separate distinct stages. It doesn't dictate the order in which organizational activities or processes should be addressed. It leaves that up to the organization to decide which is most important to their situation. Figure 10-5 is an example of a continuous model. The architecture has domain specific process areas with capability levels and associated common features or generic practices that are common to all process areas. The capability levels provide a recommended improvement path within a process area.

Examples of domains are engineering, project and organizational domains. Examples of process areas within the project domain are Project Planning, Project Monitoring and Control and Coordination. Each process area is characterized by a set of base or process specific practices.

Examples of capability levels are Performed, Managed, Quantitatively Managed and Optimizing. Each is characterized by a set of "generic" goals and practices which define a common level of sophistication or capability for the implementation of process areas. By applying the capability level ladder with associated generic goals and practices to a given process area, the organization has a road map for the improvement of one segment of its business practices

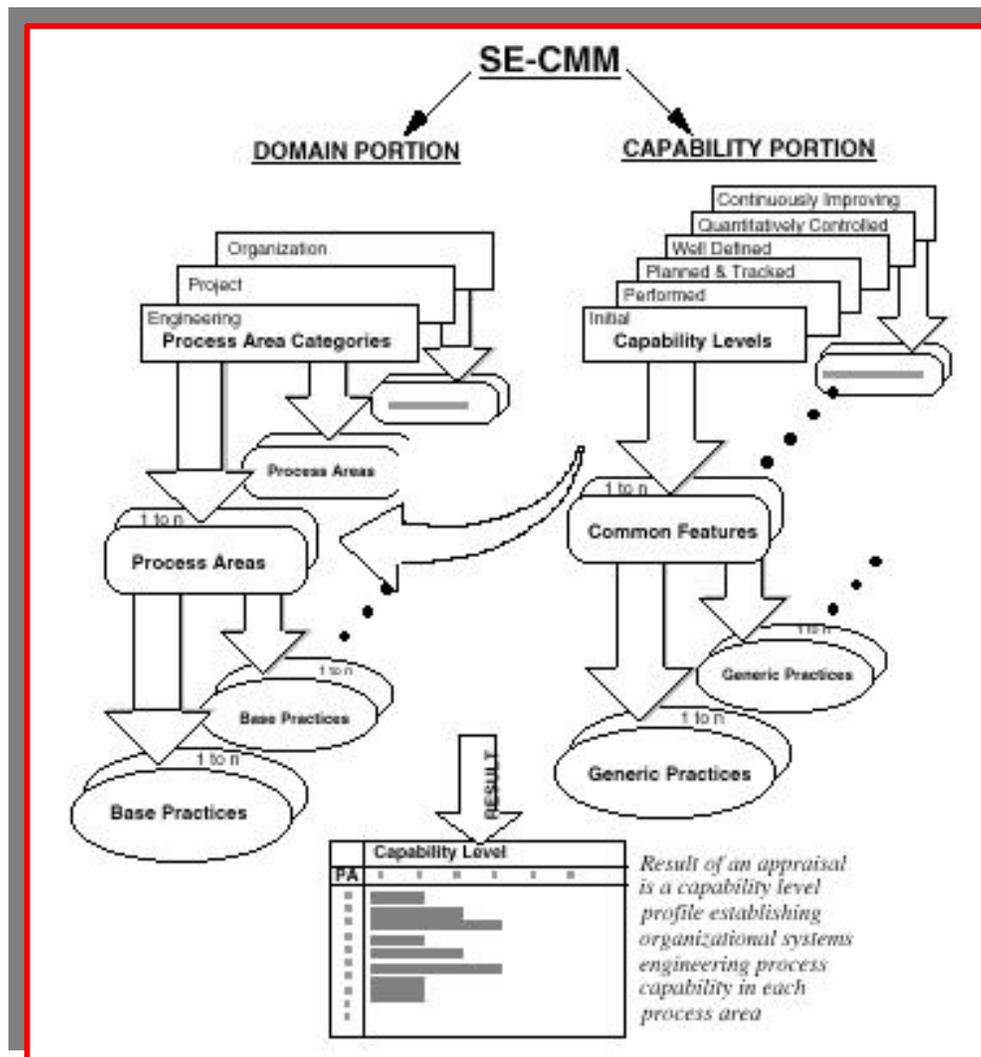


Figure 10-5. Continuous Model Architecture [SEI95]

10.3.2 Capability Maturity Model (CMM)

Like habitual and subconscious actions, software development processes are difficult to establish and even more difficult to break. Improvement seldom occurs by simply defining a more efficient process. Software engineers must understand the need to change, be convinced the new process will, indeed, improve performance, and be supported while they learn and implement it. The development processes for major software-intensive systems are often large and extremely complex. Therefore, they are difficult to define, comprehend, and especially, to implement. To aid organizations in determining the capabilities of their current process and to establish priorities for improvement, the SEI developed the software process maturity framework, as illustrated in Figure 10-6.

MATURITY LEVEL	CHARACTERISTICS	KEY CHALLENGES	RESULTS
5 OPTIMIZING	<ul style="list-style-type: none"> - Improvement fed back into the process - Automated tools used to identify weakest process elements - Numerical evidence used to apply technology to critical tasks - Rigorous defect-causal analysis and defect prevention 	<ul style="list-style-type: none"> - Still human-intensive process - Maintain organization at optimizing level 	Q P R O D U C T I V I T Y R I S K
4 MANAGED	(Quantitative) <ul style="list-style-type: none"> - Measured process - Minimum set of quality and productivity measurements - Process data stored, analyzed, and maintained 	<ul style="list-style-type: none"> - Changing technology - Problem analysis - Problem prevention 	
3 DEFINED	(Qualitative) <ul style="list-style-type: none"> - Process defined and institutionalized - Software Engineering Process Group leads process improvement 	<ul style="list-style-type: none"> - Process measurement - Process analysis - Quantitative quality plans 	
2 REPEATABLE	(Intuitive) <ul style="list-style-type: none"> - Process dependent on individuals - Basic project controls established - Strength in doing similar work, but new challenges present major risk - Orderly framework for improvement lacking 	<ul style="list-style-type: none"> - Training - Technical practices (reviews, testing) - Process focus (standards, process groups) 	
1 INITIAL	(Ad hoc/chaotic process) <ul style="list-style-type: none"> - No formal procedures, cost estimates, project plans - No management mechanism to ensure procedures are followed - Tools not well integrated; change control is lax - Senior management does not understand key issues 	<ul style="list-style-type: none"> - Project management - Project planning - Configuration management - Software quality assurance 	

Figure 10-6. Software Process Maturity Framework

The framework provides a benchmark of sound, proven principles for quality, recognized by both engineering and manufacturing disciplines to be effective for software. The purpose of the model is to help organizations determine their current capabilities and identify their most critical issues. The model characterizes the level of an organization's maturity based on the extent to which *measurable* and *repeatable* software engineering and management practices are institutionalized. This method can also be used to identify areas for improvement. Software managers usually know their problems in excruciating detail, but lack clear improvement priorities that can be understood and agreed upon by the team. By establishing a limited set of priorities and working aggressively to achieve them, more rapid progress can be made than with an unfocused effort. The CMM is organized into five maturity levels:

- **Level 1 — Initial.** The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined and success depends on individual effort and heroics.
- **Level 2 — Repeatable.** Basic program management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on programs with similar applications.
- **Level 3 — Defined.** The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All programs use an approved, tailored version of the organization's standard software process for developing and maintaining software.
- **Level 4 — Managed.** Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
- **Level 5 — Optimizing.** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies. [PAULK93]

Figure 10-7 shows the key process areas for each maturity level for the CMM for Software.

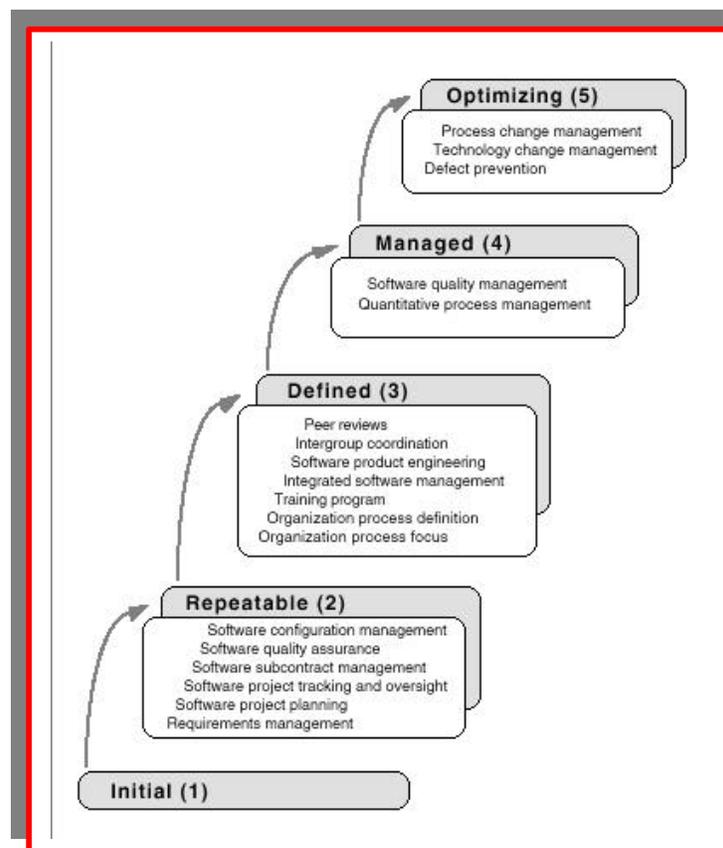


Figure 10-7. Key Process Areas and Maturity Levels for the CMM for Software [PAULK93]

Except for Level 1, each maturity level is decomposed into several key process areas (KPAs) that indicate the areas on which an organization should focus to improve its software process. The KPAs at Level 2 focus on establishing basic program management controls. The KPAs at Level 3 address both program and organizational issues, as the organization establishes an infrastructure that institutionalizes effective software engineering and management across all programs. The

KPAs at Level 4 focus on establishing a quantitative understanding of software process and work products under development. The KPAs at Level 5 cover issues that the organization and programs must address to implement continuous and measurable process improvement. Each KPA is described in terms of the key practices that contribute to satisfying its goals, and the infrastructure and activities contributing most to their effective implementation and institutionalization as the organization moves toward higher maturity levels.

10.3.3 People — Capability Maturity Model (P-CMM)

Organizations trying to improve their capability often discover a number of interrelated components must be addressed. Three key components for improvement are: people, process, and technology, as illustrated in Figure 10-8.

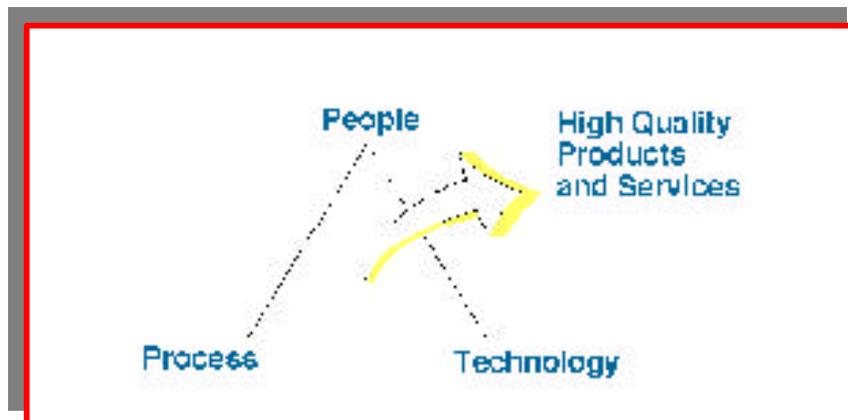


Figure 10-8. Three Key Components for Improvement [HEFLEY95]

Despite the importance of a talented staff, human resource practices are often *ad hoc* and inconsistent, and managers are insufficiently trained in performing them. Consequently, software managers often rely on their human resource departments for human resource practices administration (such as training, professional development, mentoring). Thus, these practices are applied with regard to how they impact performance. In many cases, even when software organizations are aware of the problem and want to improve these practices, they do not know where or how to begin.

The SEI's People — Capability Maturity Model (P-CMM) provides guidance on how to improve human resource management. The P-CMM is an adaptation of the CMM that focuses on developing organizational talent. It can be used to radically improve an organization's ability to attract, develop, motivate, organize, and retain the talent needed to increase software development maturity. The P-CMM helps software organizations to:

- Characterize people management maturity,
- Set priorities for improving the level of talent,
- Integrate talent growth with process improvement, and
- Establish a culture of software engineering excellence that attracts and retains the best and the brightest.

10.3.3.1 P-CMM Structure

The P-CMM is fashioned after the CMM in structure and format. The P-CMM will evolve to stay synchronized with architectural changes made in the CMM and other maturity standards, such as SPICE. It provides the same type guidance as the CMM, but in a different dimension. People management maturity describes an organization's ability to consistently improve the knowledge and skills of its staff and align their performance with organizational objectives. The P-CMM addresses a broad range of people management issues, including:

- Recruiting (attracting talent),
- Selection (choosing talent),
- Performance management (coaching talent),
- Training (enhancing talent),
- Compensation and reward (rewarding talent),
- Career development (developing talent),
- Organization and work design (organizing talent), and
- Team and culture development (integrating talent).

As illustrated in Figure 10-9, the P-CMM consists of five maturity levels. Each maturity level is a well-defined evolutionary plateau that institutionalizes a level of capability within the organization. Each level contains numerous KPAs designed to satisfy a set of goals set in the context of how people management practices are defined.

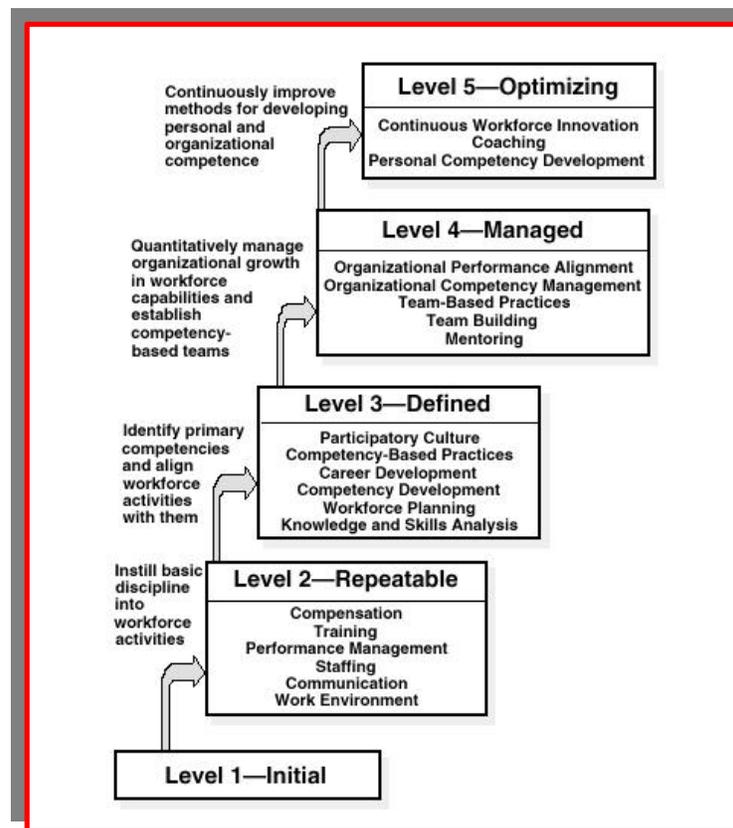


Figure 10-9. P-CMM Key Process Areas by Maturity Level [HEFLEY95]

For instance, the KPAs at Level 2 focus on instilling basic discipline into people management activities. The KPAs at Level 3 address the issues of identifying primary competencies and aligning people management activities with them. The KPAs at Level 4 focus on quantitatively managing organizational growth in people management capabilities and in establishing competency-based teams. The KPAs at Level 5 cover continuous improvement methods for developing competency at the organizational and individual level. The KPAs are internally organized by common features (i.e., those attributes indicating whether KPA implementation and institutionalization is effective, repeatable, and lasting). The five common features are: commitment to perform, ability to perform, activities performed, measurement and analysis, and verifying implementation. [HEFLEY95]

10.3.4 Software Acquisition — Capability Maturity Model (SA-CMM)

The SEI's Software Acquisition — Capability Maturity Model (SA-CMM) was developed to assess the government's internal software acquisition management process maturity. It reflects a collaborative team effort by acquisition experts from DoD, federal agencies, the SEI, and industry, and provides a framework for benchmarking and improving the software acquisition process. Its users are those organizations with responsibility for acquiring and supporting software-intensive products, e.g., government Project Managers/Program Executive Officers (PMs/PEOs), government Software Support Activities, industry PM/PEO equivalents, and senior executives. The purpose of the SA-CMM is to:

- Support senior management goal setting (i.e., each level of maturity represents an increased software acquisition process capability); and
- Support prediction of potential performance (includes accounting for factors significantly contributing to process capability).

The SA-CMM is based on the premise that, *as we mature and improve our capabilities, our probability of success increases, and we are able to make better predictions*. The purpose of assessing an acquisition organization's maturity level is to identify areas for process improvement. To make improvements, an organization must have an ultimate goal, know what is required to achieve that goal, and be able to measure progress towards achieving it. The SA-CMM provides the information and guidance needed to facilitate those activities.

The SA-CMM defines KPAs for four of five maturity levels. While the SA-CMM describes the acquirer's role (in contrast to the CMM which focuses on the developer's process), it includes certain pre-contract award activities, such as software Statement of Work preparation and documentation requirements, and source selection participation. The SA-CMM has the same architecture as the CMM, as illustrated in Table 10-2. SA-CMM maturity levels are described as:

- **Level 1 — Initial.** The software acquisition process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined and success depends on individual effort. For an organization to mature beyond the initial level, it must install basic management controls to instill self-discipline.
- **Level 2 — Repeatable.** Basic software acquisition project management processes are established to plan all aspects of the acquisition, manage software requirements, track project team and contractor performance, manage the project's cost and schedule baselines, evaluate the products and services, and successfully transition the software to its support organization. The project team is basically reacting to circumstances of the acquisition as they arise. The necessary process discipline is in place to repeat earlier successes on projects in similar domains. For an organization to mature beyond the level of self-discipline, it must use well-defined processes as a foundation for improvement.
- **Level 3 — Defined.** The acquisition organization's software acquisition process is documented and standardized. All projects use an approved, tailored version of the organization's standard software acquisition process for acquiring their software products and services. Project and contract management activities are proactive, attempting to anticipate and deal with acquisition circumstances before they arise. Risk management is integrated into all aspects of the project, and the organization provides the training required by personnel involved in the acquisition. For an organization to mature beyond the level of defined processes, it must base decisions on quantitative measures of its processes and products so that objectivity can be attained and rational decisions made.
- **Level 4 — Quantitative.** Detailed measures of the software acquisition processes, products, and services are collected. The software processes, products, and services are quantitatively and qualitatively understood and controlled.
- **Level 5 — Optimizing.** Continuous process improvement is empowered by quantitative feedback from the process and from piloting innovative ideas and technologies. Ultimately an organization recognizes that continual improvement (and continual change) is necessary to survive. [FERGUSON96]

Level	Focus	Key Process Areas
5 Optimizing	<i>Continuous Process Improvement</i>	<ul style="list-style-type: none"> • Continuous Process Improvement • Acquisition Innovation Management
4 Quantitative	<i>Quantitative Management</i>	<ul style="list-style-type: none"> • Quantitative Process Management • Quantitative Acquisition Management
3 Defined	<i>Process Standardization</i>	<ul style="list-style-type: none"> • Process Definition and Maintenance • Project Performance Management • Contract Performance Management • Acquisition Risk Management • Training Program
2 Repeatable	<i>Basic Project Management</i>	<ul style="list-style-type: none"> • Software Acquisition Planning • Solicitation • Requirements Development and Management • Project Management • Contract Tracking and Oversight • Evaluation • Transition to Support
1 Initial	<i>Competent People and Heroics</i>	

Table 10-2. Synopsis of the SA-CMM [FERGUSON96]

10.3.5 Systems Engineering — Capability Maturity Model (SE-CMM)

The Systems Engineering — Capability Maturity Model (SE-CMM), developed by the Enterprise Process Improvement Collaboration (EPIC), expresses essential characteristics of the basic technical, management, and support processes for systems engineering, and provides guidance in applying process management and institutionalization principles to the systems engineering process. The SE-CMM architecture (Table 10-3) adopts that of the SPICE program's Baseline Practices Guide (BPG).

Capability Level	Common Features
Continually Improving	<ul style="list-style-type: none"> Improving organizational capability Improving process effectiveness
Quantitatively Controlled	<ul style="list-style-type: none"> Establishing measurable quality goals Objectively managing performance
Well Defined	<ul style="list-style-type: none"> Defining a standard process Perform the standard process
Planned and Tracked	<ul style="list-style-type: none"> Planning performance Disciplined performance Verifying performance Tracking performance
Performed Informally	<ul style="list-style-type: none"> Best practices performed

Table 10-3. SE-CMM Architecture [SEI95]

Similar to the BPG, the SE-CMM architecture separates actual domain process characteristics — systems engineering — from the practices related to managing those processes. It provides generic and domain specific-guidance for process management. A base practice is defined as an engineering or management practice that addresses the purpose of a particular process area. Base practices are contained in 18 process areas (see Table 10-4), divided into three process area categories: program, engineering, and organization. For example, the SE-CMM contains the engineering process area “*integrate system*,” the purpose of which is to ensure all system elements work together. One base practice in this process area is to develop detailed interface descriptions implied by the systems architecture. Base practices provide state-of-the-practice type guidance. Systems engineering functions are described in the base practices exhibited in the process.

Engineering Process Areas	Project Process Areas	Organizational Process Areas
Analyze Candidate Solutions	Ensure Quality	Coordinate with Suppliers
Derive and Allocate Requirements	Manage Configurations	Define Organization’s Systems Engineering Process
Evolve System Architecture	Manage Risk	Improve Organization’s Systems Engineering Processes
Integrate Disciplines	Monitor and Control Technical Effort	Manage Product Line Evolution
Integrate System	Plan Technical Effort	Manage Systems Engineering Support Environment
Understand Customer Needs and Expectations		Provide Ongoing Knowledge and Skills
Verify and Validate Systems		

Table 10-4. Base Practices of the SE-CMM [SEI95]

Generic practices [*defined above*] are divided into common features [*also defined above*] which are contained in process capability levels. For example, in Level 2 (Planned and Tracked), the common feature “*planning performance*” contains practices to allocate adequate process resources, assign responsibilities for product development, and provide adequate tools to support the process. As illustrated in Figure 10-5, the advantage of the SE-CMM architecture is that principles upon which the CMM is based are abstracted and expressed in such a way that they can be used to assess any organization’s processes — i.e., its generic practices. On the other hand, essential process characteristics from a particular domain are also clearly expressed — i.e., its base practices. This architecture, isolates both types of practices and looks at them separately. They are then merged back together to build and design processes. In this way, enterprise domain and process management needs are addressed and supported. [KUHN95]

10.3.6 ISO/IEC Maturity Standard: SPICE

The International Standards Organization/International Electrotechnical Commission (ISO/IEC) is creating a set of international standards under the Software Process Improvement Capability dEtermination (SPICE) Program. One objective of the ISO/IEC effort is to create a framework for assessment approaches, while avoiding any specific approach to improvement, such as CMM maturity levels. Organizations will be able to use this standard for:

- **Self-assessment** (to help determine an organization’s ability to implement a new software program);
- **Process improvement** (to help an organization improve its own software development and maintenance processes); and
- **Capability determination** (to help a purchasing organization determine the capability of a potential software supplier).

10.3.6.1 SPICE Product Suite

The core set of SPICE products comprising the software process assessment standard include:

- ISO/IEC TR 15504-1:1998 Part 1: Concepts and introductory guide.
- ISO/IEC TR 15504-2:1998 Part 2: A reference model for processes and process capability
- ISO/IEC TR 15504-3:1998 Part 3: Performing an assessment
- ISO/IEC TR 15504-4:1998 Part 4: Guide to performing assessments
- ISO/IEC DTR 15504-5:1998 Part 5: An assessment model and indicator guidance (informative)
- ISO/IEC TR 15504-6:1998 Part 6: Guide to competency of assessors
- ISO/IEC TR 15504-7:1998 Part 7: Guide for us in process improvement
- ISO/IEC TR 15504-8:1998 Part 8: Guide for use in determining supplier process capability
- ISO/IEC TR 15504-9:1998 Part 9: Vocabulary

Field trials of SPICE-based assessments began in January 1995 and will continue until ISO/IEC 15504 is published as a full international standard, scheduled by 2001.

10.3.6.2 Baseline Practices Guide

Early in the SPICE effort, a Baseline Practices Guide (BPG) was developed. The BPG defined, at a high level, the goals and fundamental activities essential to good software engineering practices. The BPG described what activities are required — not how to implement them. BPG practices may be extended through Practice Guides that address a specific industry, sector, or other requirements. [*The CMM is an example of a sector-specific Practice Guide for large, software-intensive programs and organizations.*] The BPG defined five process categories:

- **Customer-supplier.** This category consists of processes that directly impact the customer, support development, support transition of the software to the customer, and provide for its correct operation and use.
- **Engineering.** This category consists of processes to directly specify, implement, or maintain a system, a software product, and its user documentation.
- **Program.** This category consists of processes to establish the program and coordinate and manage its resources to produce customer satisfactory products or services.
- **Support.** This category consists of processes enabling and supporting performance of other program processes.
- **Organization.** This category consists of processes establishing organizational business goals and developing process, product, and resource assets to achieve business goals.

Each process in the BPG can be described in terms of base practices unique to software engineering or management activities. Process categories, processes, and base practices provide a grouping by type of activity. These processes and activities characterize performance of a process, even if it is not systematic. Performance of base practices may be *ad hoc*, unpredictable, inconsistent, poorly planned, and/or result in poor quality products, but those work products are, at least marginally, usable in achieving process purpose. Implementing only process base practices of a process may be of minimal value and represent only the first step in building a process capability. However, the base practices represent unique, functional process activities when implemented in a particular environment.

10.3.6.3 BPG Capability Levels

The BPG expressed evolving process maturity in terms of capability levels, common features, and generic practices. A capability level is a set of common features (sets of activities) that, when applied together, increase a developer's ability to perform a process. Each level represents a major process capability improvement and process performance growth. They constitute a rational way for practice progression and harmonize different software process rating approaches (i.e., the CMM). Capability levels provide two benefits: (1) they acknowledge dependencies among process practices; and (2) they help identify which improvements might be performed first, based on a plausible process implementation sequence. The BPG lists six capability levels:

- **Level 0 — Not performed.** This level has no common features and there is a general failure to perform base practices. There are no easily-identifiable process work products or outputs.
- **Level 1 — Performed informally.** Base practices are generally performed and process work products testify to performance.

- **Level 2 — Planned and tracked.** Process base practice performance is planned, tracked, and verified. Work products conform to specified standards and requirements. The primary distinction from the previous level is that process performance is planned, managed, and progressing towards being well-defined.
- **Level 3 — Well-defined.** Base practices are performed according to a well-defined process using approved, tailored versions of standard, documented processes. The primary distinction from the previous level is that the process is planned, managed, and standardized throughout the organization.
- **Level 4 — Quantitatively controlled.** Detailed measures of performance are collected and analyzed. This leads to a quantitative understanding of process capability and an improved ability to predict and manage performance. The quality of work products is quantitatively known. The primary distinction from the previous level is that the defined process is quantitatively understood and controlled.
- **Level 5 — Continuously improving.** Quantitative process effectiveness and performance efficiency goals (targets) are established based on organizational business goals. Continuous process improvement against these goals is enabled by quantitative feedback from defined process performance and the piloting of innovative ideas and technologies. The primary distinction from the previous level is that the defined, standardized process undergoes continuous refinement and improvement based on a quantitative understanding of the impact of process changes.

10.3.7 Common Features and Generic Practices

A common feature in the BPG is a set of practices (called generic practices) that address the aspects of process implementation and institutionalization. The words “*common*” and “*generic*” convey the idea that these features and practices are applicable to any process, with the goal of enhancing the capability to perform that process. For example, “*planning*” is a feature common to improved management of any process. Table 10-5 lists BPG common features and generic practices by capability level.

CAPABILITY LEVEL	COMMON FEATURE	GENERIC PRACTICE
<u>LEVEL 5</u> Continuously Improving	Improving Organizational Capability	<ul style="list-style-type: none"> Establish process effectiveness goals Continuously improve the standard process
	Improving Process Effectiveness	<ul style="list-style-type: none"> Perform casual analysis Eliminate defect causes Continuously improve the defined process
<u>LEVEL 4</u> Quantitatively Controlled	Establishing Measurable Quality Goals	<ul style="list-style-type: none"> Establish quality goals
	Objectively Managing Performance	<ul style="list-style-type: none"> Determine process capability Use process capability
<u>LEVEL 3</u> Well-Defined	Defining a Standard Process	<ul style="list-style-type: none"> Standardize the process Tailor the standard process
	Performing the Defined Process	<ul style="list-style-type: none"> Use a well-defined process Perform peer reviews Use well-defined data
<u>LEVEL 2</u> Planned and Tracked	Planning Performance	<ul style="list-style-type: none"> Allocate resources Assign responsibilities Document the process Provide tools Ensure training Plan the process
	Disciplined Performance	<ul style="list-style-type: none"> Use plans, standards, and procedures Do configuration management
	Verifying Performance	<ul style="list-style-type: none"> Verify process compliance Audit work products
	Tracking Performance	<ul style="list-style-type: none"> Track with measurement Take corrective action
<u>LEVEL 1</u> Performed Informally	Performing Base Practices	<ul style="list-style-type: none"> Perform the process

Table 10-5. BPG Capability Levels, Common Features, and Generic Practices [KONRAD95]

BPG capability levels and CMM maturity levels are similar, yet distinctly different. BPG capability levels are applied on a per process basis, while CMM organizational maturity levels are a set of process profiles. Also, the BPG architecture does not prescribe any specific organizational improvement path. Improvement priorities are left completely up to the software organization, as determined by its business objectives. Individual processes, at either organization or program level, can be measured and rated using the BPG continuous improvement architecture. [KONRAD95]

10.4 Lessons Learned in Implementing the Software Development CMM

The Software Engineering Division of the Technology and Industrial Support Directorate at Hill AFB, UT was assessed at CMM Level 5 in July 1998. Patrick W. Cosgriff, a member of their Software Engineering Process Group (SEPG), offers the following as lessons learned [COSGRIFF99¹], [COSGRIFF99²]:

- Understand the practices one level above the implementation level. Give some thought to how the practices interrelate and build off each other. This may save some rework in the long run.
- Enforcement and implementation are basically the same thing, especially in large organizations. Or perhaps a softer way to state this would be that enforcement is the most effective implementation strategy. For my money, objective audits done by capable, well trained people, with a clear set of audit requirements is the most effective enforcement/implementation strategy.
- If you are a large diverse organization you may want to coordinate the developing of project processes with the development of your organizational processes right from the start. Again, this may reduce rework associated with fundamental style and design differences between different product lines.
- Emphasize performance of project planning activities, not creation of documents that gather dust.
- Don't ignore intergroup coordination. It is kind of hard to get your hands on but it is very important. Think of it as a characteristic you want integrated into all your activities, not a discrete set of activities unto themselves.
- Consider the data requirements of the Software Quality Management KPA when implementing peer reviews. It is not that difficult to gather the extra data needed to support Software Quality Management and Defect Prevention. Having historical data on defects will give you a big jump on implementing these higher maturity practices.
- Level four and five KPAs can be implemented together, in fact defect prevention is the logical extension of software quality management, and Process Change Management is the logical extension of Quantitative Process Management.
- Don't limit your solutions to the activities and sub-practices listed in the CMM.
- Package plans, processes, and procedures in a usable way.
- Make the CMM fit your organization, not the other way around.
- Integrate common activities from different KPAs as much as possible.
- Data complements common sense, it doesn't replace it.

10.5 Software Development Capability Assessment Methods

Now that software acquisition and development capability maturity have been defined and the relationship of capability maturity to processes has been explained, how is this information to be used in improving the acquisition and development of software intensive systems? Until recently, this information has been applied exclusively to software development organizations. If applied

by an acquisition organization, it was usually in the form of a Software Development Capability Evaluation or Software Capability Evaluation. If applied by a development organization for internal process improvement, it was usually applied as a CMM Based Appraisal for Internal Process Improvement, SPICE audit, or ISO 9001 audit.

Software development capability assessments are an effective method for determining the maturity of an organization's process. They provide a performance rating system that was established to be fair, accurate, and enforce uniform procedures, clear definitions, consistent measurements, and reliable information to keep vendors from challenging negative ratings. These assessments involve visits to bidders' facilities to determine their readiness to perform on a contract and their software development maturity. They are used to ascertain whether the developer has a mature software process in place that is predictable, repeatable, and manageable in terms of cost and schedule. The purpose of these assessments is risk mitigation. They are used to determine what risks are associated with contracting a given organization to perform your development task. An award to a contractor with a mature, well-defined, standardized process can translate into substantially lower program risk and cost savings for the Government through reduced documentation, oversight, review, and auditing requirements.

REMEMBER: In addition to having a mature software development process, the developer should also have experience in the application domain being developed. Although a developer might have a high-level process maturity, the lack of domain expertise could have a drastic impact on product development (i.e., performance, cost and schedule). You need both process maturity and domain expertise to minimize software development risk. This chapter addresses development maturity, and assumes the software developer has the necessary domain experience.

Two software development capability assessment methods are available for source selection evaluations. The Air Force Materiel Command's Aeronautical Systems Center Software Development Capability Evaluation (SDCE) is best used for developers of weapon systems with embedded software or any application requiring substantial systems engineering. The SEI Software Capability Evaluation (SCE) is appropriate for management information systems (MIS) developers, and has been used for command, control, communications, computers and intelligence (C4I) developers. However, with the substantial systems engineering required by C4I programs, you should consider performing a SDCE to ensure the developer has a mature systems engineering capability. The objective of these methods is to provide structured, consistent, and comprehensive approaches for evaluating the software process. A high rating on the review does not guarantee software development success, but the evaluation does isolate areas needing closer consideration during source selection (and if selected, after contract award).

NOTE: For questions about command, control, and communications (C3) and ground electronics systems acquisitions capability assessments, contact Electronic Systems Center [see Volume 2, Appendix A]. For MIS acquisitions contact the Standard Systems Group (SSG) [see Volume 2, Appendix A]. For avionics and embedded systems contact Aeronautical Systems Center. Air Force in-house software development organizations with questions on Software Process Improvement and Software Maturity Assessments should contact the Software Technology Support Center.

Once your program is assessed, it is of little or no use if you are not committed to *improvement*. No matter how often the assessment is performed, it is only a starting point. Each time an assessment is performed, it identifies your current level of capability — but more importantly — it identifies a point from which to begin your next round of improvement. Those few organizations who have achieved a CMM Level 3 or above claim they got there, and stay there, because they have an organization-wide quality attitude. Always looking for ways to improve, they develop an extensive set of measures that they perpetually re-evaluate.

10.5.1 Software Development Capability Evaluation (SDCE)

The SDCE evaluates a contractor's ability to develop software for a specific weapon system program, as defined in the RFP. It also helps to decide whether the contractor has the capacity and sufficient qualified personnel available to complete the proposed software development. Assessing capability during source selection accomplishes three related objectives:

- The offeror's capability and capacity to develop the required software within the program baseline is determined;
- The review process elicits a contractual commitment by the offeror, if selected, to implement the methods, tools, practices, and procedures making up their software development process; and
- Insight is gained into each offeror's systems and software engineering development methods and tools to be applied to your program.

The SDCE concentrates on five areas: management approach, management tools, development practices, personnel resources, and programming language technology. The review is normally performed during the Engineering and Manufacturing Development (EMD) request for proposal (RFP) preparation and source selection acquisition phase. However, when software developed during Demonstration/Validation (Dem/Val) is planned to be carried through to EMD, an SDCE should be performed during the Dem/Val source selection phase. Your RFP must state that offerors provide specific information describing their software development methods, including examples of how their methods have been applied on past or on-going programs. If an open discussion is conducted, an in-plant review of the offeror's team is performed by the Government. *The evaluation can also be based solely on the material submitted with the proposal, with the in-plant portion of the SDCE conducted after contract award. [Aeronautical Systems Center policy requires the use of SDCE results in all weapons systems software source selections.]*

10.5.2 Software Engineering Institute (SEI) Software Capability Evaluation (SCE)

The SCE is described in *SCE Version 3.0 Method Description* (CMU/SEI-96-TR-002, April 1996) and the *SCE Verion 3.0 Implementation Guide for Supplier Selection*, (CMU/SEI-95-TR-012, April 1996) by the SEI.

SCE is a method for evaluating the software process of an organization to gain insight into its software development capability. This insight can be a valuable input to process improvement activities. Hence, the SCE Method helps evaluate the software process capability of a software

development organization (an organization that develops and/or maintains software products). Software process capability refers to the range of expected results that can be achieved by following a process. The processes evaluated by SCE include decision-making processes (such as project planning), communication processes (such as intergroup communication), and technical support processes (such as peer reviews and product engineering)—but not technical production processes (i.e. processes required by a particular methodology, such as object oriented design). The SCE Method does not evaluate technical production processes such as requirements analysis, specification, and design, but instead focuses on the management of the technical production processes and on other key processes. [BARBOUR95]

A WORD OF CAUTION! An SCE investigates areas generally limited to the processes used. For example, this includes the process of selecting appropriate tools and methods, and training personnel to use them. However, an SCE does not evaluate whether the processes themselves are effective or efficient, nor does it address the appropriateness of the tools and methods used by the developer. Therefore, a proposal by a mature software development organization to use new, state-of-the-art tools and methods could be a significant risk if the developer does not have an experience base to handle them. [HOROWITZ95]

SEI transition partners train source selection teams on conducting Software Capability Evaluations (SCEs). SEI instructor personnel do not lead or formally participate in SCEs. However, they may observe SCE teams while they conduct evaluations on site. These observation trips, lessons-learned reports, and experiences have been major contributors to the SCE method's evolution into its current form. [BARBOUR93]

NOTE: ESC can provide SCE evaluation teams upon request for Air Force procurements. Contact ESC for more information [see Volume 2, Appendix A]. For more information on the SEI Transition Partners, contact the SEI [see Volume 2, Appendix A].

10.5.3 Addressing Maturity in the Request for Proposal (RFP)

To ensure the software process enacted for your program is predictable, repeatable, and manageable in terms of quality, cost, schedule, and performance, you should evaluate the offeror's software development capabilities prior to (or during) source selection. Remember, *you are buying the process as well as the product!* Performing a software development capability assessment will help you identify risks associated with the offeror's approach. Risk identification is possible, since you will have:

1. An understanding of how the organization managed software development efforts in the past; and
2. The opportunity to compare past performance with the proposed software development process.

Therefore, you must pay due attention to the offeror's software development processes, starting with overall assessments like the SCE or SDCE, which focus on the details of tools, metrics, personnel facilities, management control, and language experience. Based on the maturity level of the selected contractor, you should consider customizing your contract to adapt that offeror's

strengths and weaknesses. For example, if the contractor has achieved a high level of maturity (3 or above), you may decide that online access to the contractor's development environment and management status reports (e.g., cost, schedule, risk management and metrics data) is an effective alternative to the traditional oversight mechanisms of formal reviews and submission/approval of data items. Alternatively, if an offeror's process for coordinating the efforts of different engineering disciplines and stake holders is relatively weak, you may add a requirement for an on-site liaison to support coordination with users and the contractors developing interfacing systems.

10.6 References

- [BARBOUR93] Barbour, Rick, Software Capability Evaluation Version 1.0 Implementation Guide, CMU/SEI-93-TR-18, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, 1993
- [BARBOUR95] Barbour, Rick, Software Capability Evaluation Version 3.0 Implementation Guide for Supplier Selection, CMU/SEI-95-TR-12, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, 1995
- [CLOUGH92] Clough, Anne J., "Software Process Technology," *CrossTalk*, June/July 1992
- [COSGRIFF99¹] Cosgriff, Patrick W., "The Journey to CMM Level 5: How Long Does It Take?", draft article submitted to *CrossTalk*, March 1999
- [COSGRIFF99²] Cosgriff, Patrick W., and David Haakenson, "The Right Things for the Right Reasons," briefing given to USPIN, 1999
- [DELAVIGNE94] Delevagne, Kenneth T. and J. Daniel Robertson, Deming's Profound Changes, PTR Prentice Hall, Englewood Cliffs, New Jersey, 1994
- [DSMC90] Defense Systems Management College, Systems Engineering Management Guide, US Government Printing Office, Washington, RUN-TIME, 1990
- [FERGUSON96] Ferguson, Jack R., et al, "Software Acquisition Capability Maturity Model (SA-CMM) Version 1.01, CMU/SEI-96-TR-020, Software Engineering Institute, Carnegie Mellon University, Pittsburgh PA, December 1996
- [GUINAN97] Guinan, P. J., J. G. Coopriider, and S. Sawyer, "The Effective Use of Automated Application Development Tools," *IBM Systems Journal*, Vol 36, No. 1, 1997
- [HAMMER96] Hammer, Michael, Beyond Reengineering, HarperCollins, New York, New York, 1996
- [HEFLEY95] Hefley, William E., et al., "People Capability Maturity Model (P-CMM) Incorporating Human Resources into Process Improvement Programs," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1995
- [HOROWITZ95] Horowitz, Barry M., personal communication to Lloyd K. Mosemann, II, December 1995
- [HUMPHREY95] Humphrey, Watts S., A Discipline for Software Engineering, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995
- [KONRAD95] Konrad, Michael D., and Mark C. Paulk, "An Overview of SPICE's Model for Process Management," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1995
- [KUHN95] Kuhn, Dorothy A., and Suzanne M. Garcia, "Developing a Capability Maturity Model for Systems Engineering," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1995
- [PAULK93] Paulk, Mark C., Bill Curtis, Mary Beth Chrissis, Charles V. Weber, Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-24), Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1003
- [SAIEDIAN95] Saiedian, Hussein, and Richard Kitzara, "SEI Capability Maturity Model's Impact on Contractors," *IEEE*, January 1995
- [SEI94] *Benefits of CMM-Based Software Process Improvement: Initial Results* (CMU/SEI-94-TR-13), Software Engineering Institute, Carnegie-Mellon University, August 1994
- [SEI95] *A Systems Engineering Capability Maturity Model, Version 1.1* (CMU/SEI 95-MM-003), Software Engineering Institute, Carnegie-Mellon University, November 1995
- [VU97] Vu, John D., Presentation at 1997 National SEPG Conference
- [WILLIAMS97] Williams, Karl, "The Value of Software Improvement," SPIRE97, 1997