2

Capability Maturity Model

Organization of the CMM

The CMM developed by the Software Engineering Institute (SEI) has been explained in great detail in different publications. Some of you have probably already read about it, gone through CMM training, or been trained on how to conduct a software capability evaluation (SCE).

CMM is a conceptual framework that represents process management of software development. CMM contains five maturity levels or stages [1].

1. *Level 1, initial:* The software process is characterized as ad hoc, chaotic, and heroic. Few processes are defined or followed, and project success depends on individual effort. There is no formal management control over software development.
2. *Level 2, repeatable:* This level provides an introduction to the formal, documented process. Basic management processes are

established to control cost, scheduling, and functionality. The necessary process discipline is in place to repeat previous successes on projects with similar applications. Elevation from Level 1 to Level 2 means that the organization has established project management control, established a software engineering process group (SEPG), and formally introduced software engineering methods and techniques.

3. *Level 3, defined:* This level provides a foundation for continuous process improvement by establishing the necessary process management functions to control process parameters. The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use a tailored version of the organization's standard software process for developing and maintaining software.
4. *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.
5. *Level 5, optimized:* Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

These five maturity levels define an original scale for measuring the maturity of an organization's software process and for evaluating its software process capability (Figure 2.1). Each maturity level indicates the level of process capability.

Levels 2 through 5 are decomposed into 18 key process areas (KPAs) as shown in Figure 2.2. Each KPA is organized into five sections, called *common features*.

1. *Commitment to perform:* Describes the actions that must take place within the organization to ensure that the process is established and will function. The commitment to perform feature usually involves developing organizational policies and senior management sponsorship.

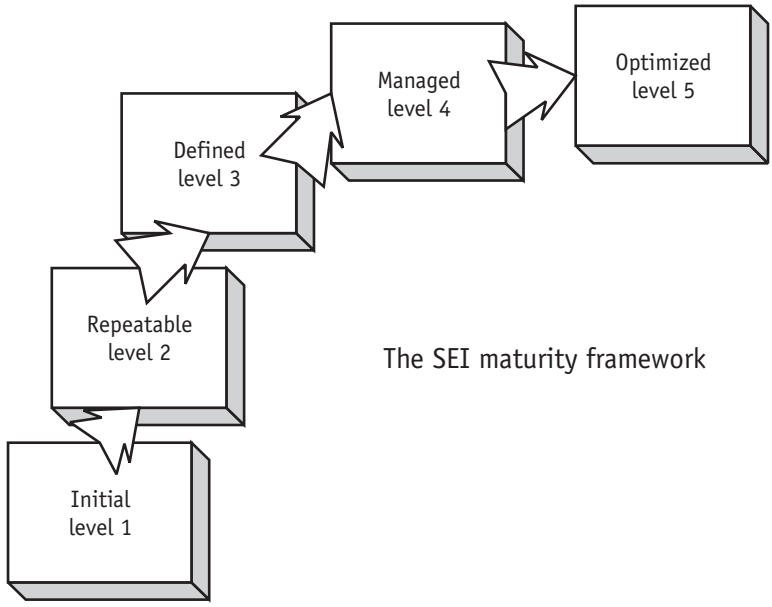

Figure 2.1 SEI maturity framework.

2. *Ability to perform:* Describes the preconditions that must exist in the project or organization to implement the software process competently. It usually includes resources, training, organizational structures, and tools.
3. *Activities performed:* Describes the roles and procedures necessary to implement a key process area. This feature typically involves establishing plans and procedures, performing the work, tracking it, and taking corrective actions as necessary.
4. *Measurement and analysis:* Describes the need to measure the process and analyze the measurements. The measurement and analysis feature typically includes examples of the measurements that could be taken to determine the status and effectiveness of the activities performed feature.
5. *Verifying implementation:* Describes the steps to ensure that the activities are performed in compliance with the process that has

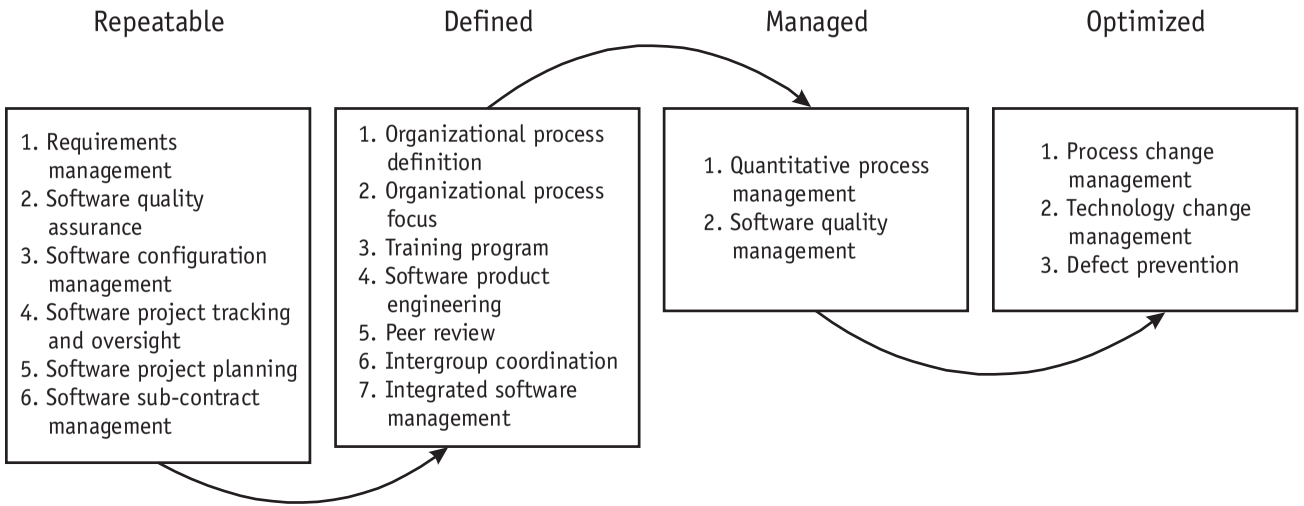

Figure 2.2 CMM levels and KPAs.

been established. Verifying typically means reviews and audits by management and software quality assurance.

The practices in the activity performed common feature (item 3 above) describe what must be implemented to establish a process capability.

Key practices (CMM has 316 suggested) describe the infrastructure and activities that contribute to the effective implementation and institutionalization of the key process area. The key practices describe "what" should be done, but do not mandate "how" it is to be done. They do not require a specific model of the software life cycle such as a waterfall or spiral. They do not demand a specific organizational structure. They do not judge a technical approach to product implementation or the development tools used. They merely suggest; that is, they do not mandate. Instead they leave the "how" up to each individual organization. The CMM is a management model. It gives you a guideline on how to manage a software process and does not judge the technical aspects of the product development or the performance of the developers.

The software process improvement and level achievements of the SEI CMM are very similar in nature and have the same objective: to improve how software is developed. Software process improvement is a general issue, the specifics of which are addressed in CMM. Three main issues are related to software process improvement and software project management: (1) cost, (2) scheduling, and (3) quality.

These issues often become big, pathological problems, which end up making the whole organization dysfunctional. These issues are not technical issues. These issues are issues of management—the issues of how to manage and identify the impact of present or potential problems. How do we prevent an issue from becoming a problem?

The *improvement* in the term *software process improvement* refers to an improvement in management techniques! That is what most companies had not realized earlier. This is where both the nature of conflict (us versus them) and inconsistency in organizational goals arise. "All too often, proactiveness is reactiveness in disguise. If we simply become more aggressive fighting the enemy out there, we are reacting—regardless of what we call it. True proactiveness comes from seeing how we contribute to our own problems. It is a product of our way of thinking, not our emotional state" [2].

I was told this story: Once upon a time, a big company invited a very famous consultant to its headquarters for a discussion about CMM and software process improvement. The company published a memorandum requiring the participation of all software managers in the meeting. The first thing the consultant said was that "the cause of poor quality is poor management." Right there, the management immediately lost any interest, and after the break, the room was almost empty. The company never invited the consultant back.

The five CMM levels

Now we take a look the five CMM levels in a little more detail.

Level 1: initial

The first question I ask new clients is this: "When was your last reorganization?" Why do I ask that question? Because I need a starting point.

Unfortunately, the initial level processes are the most practiced processes in the software business. Companies develop these processes over time and develop ownership. To some degree, they represent the reality of the software development organization: pressures, crises, and limitations. As the processes grow, more and more software developers get involved. A particular process can became so complex that nobody knows exactly how and why it works. The Level 1 processes constantly change, just as organizations change. Processes are not documented so people cannot really understand them. The success of Level 1 organizations in product development comes not from the project management function but from the efforts of individuals performing their specific tasks without a clear understanding how the process works. I like to compare the Level 1 organization with a typical John Wayne movie, in which the main character is a hero who always saves the new settlers from the bad guys.

One of the interesting characteristics of the Level 1 organization is that management's presence is very strong, but very inefficient due to the lack of communications. As a result, the quality and the delivery schedule

of the product is unpredictable. At this point, the management and management decisions become very unpopular among the practitioners [3].

At Level 1, management's remedy to improve a process is to reorganize. They believe the process is okay, but the way in which management executes the process is wrong. In the first few months after a reorganization the productivity of the employees usually increases; fear drives them, fear of management. The productivity of the department increases not because the process has been improved by the reorganization, but because extra hours are spent at the terminal.

What is the next step to address symptoms not problems? Another reorganization!

Level 2: repeatable

At this level, basic project management processes are established to track costs, schedules, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications [1].

A repeatable process is a documented process. The process should be communicated on all levels with the help of documented procedures. (By the way, if your organization does not have documented procedures, you failed the assessment.)

At Level 2 the project managers shift their attention from technical issues to organizational and managerial issues. Therefore, they should have two basic abilities: (1) the ability to determine and communicate the status of a project as accurately as possible and (2) the ability to estimate the impact of past decisions and document and review them in terms of scheduling, effort, and product quality.

You start your continuous process improvement at the repeatable level by listening to your software practitioners and by documenting the process with formal procedures. These procedures will be used across the organization and continuously improved. Everybody wants action—discussions take time. *Level 2 is about listening, communicating, and documenting* [3].

Level 2 is concerned primarily with how the organization estimates, defines, and determines the status of a project and with the impact of decisions in terms of costs, scheduling, and quality. The following issues are very important at the repeatable level:

- Impact of changed commitments to the customer;
- How the software requirements baseline has been modified, how extensive the changes are, and how changes are controlled and communicated;
- Impact of the work effort (how the resources are planned in order to implement the customer's requirements and allocated to accommodate the modifications);
- Impact of the schedule (how the schedule is modified based on changed commitments);
- Impact of costs (determining the cost implications of the modifications).

Because these project management issues are interdependent, CMM suggests ways to control the process and suggests the metrics approach for qualifying and quantifying the impact of modifications.

The repeatable process enables organizations to integrate the historical analysis of past performance into documented procedures to improve continually. With the use of measurement, a historical analysis of previous projects (procedures, interpretations, and knowledge base required) can take place, providing management with information about design, scheduling, quality, and costs. This information helps management make qualified decisions before contractual obligations are made. *Historical analysis is a technique that allows organizations to "adjust future conduct based on past performance."*

Historical analysis of previous projects is necessary to establish and understand the pattern of how the organization develops the product. Project history data fall into three major categories: software development, operational data, and maintenance data. *Software development data* and *operational data* describe the characteristics of the project (e.g., product, process, and resources) and form the foundation for developing estimation and prediction models. *Software maintenance data* describe the behavior of the product (e.g., its successes/failures and modifications history).

The approach is to determine systematically answers to the following questions when looking at historical analyses:

- What is known about the past project?
- What is unclear?
- What is assumed?
- In what ways are past project implementations similar?
- In what ways are past project implementations different?

Each question demands a factual answer, and these can only be obtained through analysis and investigation. If objective answers are obtained, then the decision is much less likely to be faulty.

Level 3: defined

The software process for both management and engineering activities is documented, standardized, and integrated into an organization-wide software process. All projects use a documented and approved version of the organization's process for developing and maintaining software [1].

Level 3 supports projects by institutionalizing and expanding the main principles of Level 2 across the entire organization. At Level 3 tasks are being formally defined and documented. Level 3 shifts the emphasis from the project itself to organizational support of the project. Level 3 is built on the project management foundation of Level 2. If the repeatable level defines what to do and who should do it, the defined level specifies when to do it and how to do it (Figure 2.3).

Level 3 establishes a common understanding of the process across the organization and also provides flexibility in terms of how to change

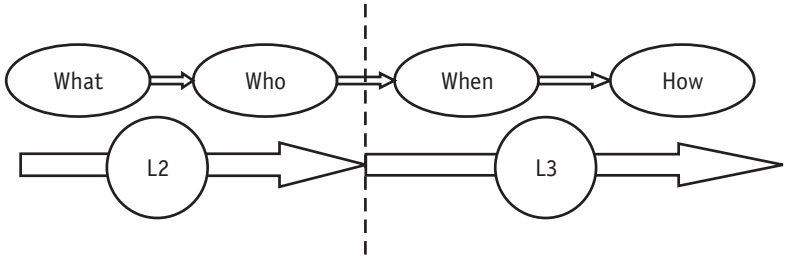

Figure 2.3 Transition from Level 2 to Level 3.

the process and relate it to ongoing activities. It provides the foundation for the quality of management decisions.

At maturity Level 3, the organization not only defines its process in terms of software engineering standards and methods, it also makes a series of organizational and methodological improvements. These improvements include design and code reviews, training programs for developers and review leaders, and increased organizational focus on software engineering. The establishment of a software engineering process group is considered to be one of the major improvement of this phase, and it focuses on the software engineering process and the adequacy with which it is implemented.

A process cannot be defined if we do not know *how* the process flows throughout the organization.

Level 4: managed

In Level 4, detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures [1]. The KPAs of Level 4 are probably the most misunderstood requirements in the entire structure of CMM because the directions about how to move from Level 3 to Level 4 are very fuzzy.

The managed level is not merely about measurements; if it were, it would be called the measured level. Instead, Level 4 is about seeing and understanding the process as a whole by identifying interrelationships between subprocesses and managing them as they change. If Levels 2 and 3 are just a foundation for successful process management building, Level 4 provides a vision for the organization to control and finish construction of the building. Level 4 is an integration stage for all policies, procedures, and measurement practices implemented in the previous levels. Implementation of Level 4 KPAs allows management to predict not only the quality of the product, but the quality of its decisions.

Consider the story of three masons. A pedestrian asked three masons working on a construction project what they were doing. The first mason said, “I am laying bricks”; the second mason said, “I am building a wall”; and the third mason said, “ I am building a temple to worship God.”

Level 4 involves a dynamic set of subprocesses that is organized and internally directed toward certain goals. What are your goals? How productive have you been in achieving these goals? The process parameters should be defined first and measured second. The key elements of such a process include the following:

- Setting of subgoals that are instrumental to the achievement of higher level goals;
- Implementation of planned actions designed to coordinate the behavior of the process and its subprocesses to move them toward established goals;
- Management and monitoring of process performance so corrective actions can be taken if necessary.

Moving to Level 4 is very difficult and very time consuming, unless the organization starts implementation of measurement and metrics at the very beginning of the software process improvement.

Level 5: optimized

Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies [1]. In Level 5, the organization moves from the process improvement stage into the process management stage. Changes are monitored closely, using the measurement system, and the process is refined as needed. The organization operates at peak performance.

The keyword for Level 5 is *feedback*. It is not only a keyword, it is a concept. Management and its actions are part of the feedback process. Everybody shares responsibility for the problems associated with the process. No one person can be blamed.

You have to be very careful, though, with the feedback process. Why? It can limit your growth or push you back to the chaotic stage. The feedback should be goal oriented and balanced, otherwise you will fall into the trap of managing the changes that are wrong for your organization from the very beginning, and the process will feed itself with faulty information.

In the Level 5 organization, cycles are as regular as a daily commute.

> Commuter—one who spends his life
> In riding to and from his wife;
> A man who shaves and takes a train
> And then rides back to shave again.
>
> *E. B. White*

Benefits of the CMM approach

Do you remember the story of the Knights of the Round Table? The circle of the Round Table united its members with a set of common goals, ideas, and ethics—a common bond. After knights spent time at the Round Table, each of them went his own way, killing dragons and saving women, but their paths met, crossed, and intertwined [2].

CMM unites the software development organization and provides the following navigation and leadership tools with which management can improve the way in which they manage software projects:

- A clear understanding by software developers of what is expected of them. ("I have no idea what the customer really wants." "No one asked me if it is possible to implement this.")
- Clarity of work procedures and sufficiency of resources, skills, and knowledge. ("I do not have the training, the tools, the proper resources, or the time to do this job right.")
- Cross-functional processes that affect customer quality are defined. ("All I do is fix bugs.")
- Opportunities for improving the organization, process, and product quality. ("Hey, I tested *my* change, and it worked just fine in *my* environment.")

Common features of key process areas

The CMM's concept of common features is not really a new concept. In 1931, Walter Shewhart recognized that the outputs of manufacturing

processes are subject to some types of variation, variations that cause excessive quality problems [4]. No two items going through a production line turned out exactly the same. The variations occurred as a result of imprecisions on the part of the machinery or the material or because of the skills of different machine operators. Shewhart called these *common causes* of variations and viewed them as a natural phenomenon in the production process. He developed methods for measuring variations and provided managers and workers the tools for determining whether their processes were operating smoothly or needed attention in order to improve quality.

On top of that, he recommended a course of action for each case. As a result, manufacturing personnel were able to identify the variations in a process and find ways to reduce them. This became known as the Shewhart cycle, and later reappeared in Japan under the name *kaisen* (continuous improvement). Don't you remember "Plan, Do, Check, and Act"?

Let's use the example of the Shewhart cycle as an analogy for our cause. CMM provides a structure for KPAs. This structure contains *common features;* is composed of goals, commitments, abilities, activities, measurements, and verification; and can serve as the main structure for software process improvement.

Goals

The goals stated in CMM are linked directly to the problems and opportunities of the software development organization.

For example, one CMM goal is the following: System requirements allocated to software are controlled to establish a baseline for software engineering and management use. Can you relate that goal to your business situation? Of course, you can. The requirements management problem is one of the most common problems in a software engineering community. In many companies the requirements are not controlled at all. They change all the way through the software development cycle, including the beta test.

Try to relate the KPA's goals to your business case. Very soon, you will realize that the problems you experience with your software development are the results of missing goals. Examples of typical problems include the following:

- Product delivery is late.
- The size and costs of the project keep increasing.
- The project is out of control.
- The product does not meet requirements.
- The performance of software subcontractors is poor.

This list, however, is endless.

Commitment to perform

This common feature usually involves establishment of organizational policies and requires senior management support. The key to this feature is to be able to establish a relationship between senior management and the actual people doing software process improvement using CMM as a guide.

Software process improvement is successful when the commitment of senior management is successfully tested. When senior management supports and encourages the members of a SEPG team, and gives them a good review from time to time, the effort of process improvement is going to succeed. If management's commitment is not truly strong, all improvement efforts are going to fall apart at the first sign of problems. The software improvement effort will die a slow death if management does not share commitment. As an example, I would like to quote one software development manager.

> We are not going to *mandate* the requirements management tool for this release. Instead, individual engineering managers can *choose* to use it if it helps them to meet their deliverables. If a manager believes the tool will aid in delivering on time with quality, he or she can use it and then demonstrate it to the rest of the organization for use on future products.

The degree of commitment of senior management is equal to the degree of risk the organization will take once it is committed to improving its processes and staying competitive in the marketplace. It is time for the top management to ask themselves a few questions.

- Does the organization have a vision, mission, goals, and objectives for the process improvement effort?
- How are they communicated?
- Are they written down?

Any attempt at software process improvement needs a stated goal. If there is no stated goal, the project will have no focus. In addition, a development organization with few policies and procedures will show inconsistent performance. This organization will go into the crisis mode at the first sign of a problem.

Policy statements generally refer to a written, organizational policy for the practices of that key process area. The policy should define responsibilities and authorities. The typical situation in most companies that are just beginning their software improvement efforts is that responsibilities are clear and authorities are not.

The policies and procedures handbook, if you choose to create one, should be a living document. This document is needed for short- and long-term planning. It will help to test the ideas, gain experience, and continuously improve the process.

Ability to perform

The ability to perform common feature describes the precondition that must exist in the project or organization to implement the software process competently. This simply means that you should have the abilities to do what you have planned and committed to do.

The level of detail of a documented procedure can vary significantly, from a handwritten individual desk procedure to a formal organizational standard operating procedure. The formality and level of detail depends on who will perform the task or activity (e.g., individual or team) and how often it is performed.

A documented procedure is usually needed so that the individuals responsible for a task or activity are able to perform it in a repeatable way and so that others with general knowledge of the area will be able to learn and perform the task or activity in the same way. This is an important aspect of institutionalizing a process.

Have you created the organizational structure to support project needs and software process improvement needs? One of the very important points that management needs to understand is that the organization should function and be managed as a system, not as a collection of independent subfunctions. Some organizations have unlimited funds for process improvement (I would love to see such a company). They can buy any tools for their "engineering sandbox." They can hire the best instructors in the world to teach their people new technology, but that organization will still be unable to improve anything without proper training of management in how to integrate individual performance goals into the overall organizational process of software development.

Activities performed

One hundred-and-fifty key practices fall under the activities performed common feature, representing almost half of the total number of key practices specified by CMM.

CMM is a descriptive not a prescriptive model. Activities performed vary from one organization to another. The implementation activities are different because the level of details, organizational focus, and need for planning and documentation are different. The focus of the CMM activities is on the software development process flow and requires procedures describing results to be accomplished.

Another important aspect of the CMM implementation is that you structure the activities around the tasks—not around the people. Your goal is to build a system that will allow continuous process improvement and be independent from any organizational changes involving people. Some companies that struggled through CMM implementation for some time found themselves in the situation of going down from Level 4 to Level 2 because the process improvement infrastructure was based on the effort and skills of individuals. The performed activities must be viewed and managed from a systems standpoint, not just management of an individual unit or function. What does it mean to manage activities from a system standpoint? It means that management should understand the relationship of their function in the software development life cycle to the total picture and define the entry and exit criteria for their part of the cycle.

Measurement and analysis

How do you know your development organization is doing things right? How are you measuring progress? How will you know how productive you are in achieving your goals or desired phase?

If your goals and objectives are specific and measurable, you can define measurements and metrics as a standard you can measure yourself.

Measurement and analysis is another common feature of KPA that describes the need to measure the process and analyze the measurements. The measurement and analysis section of CMM usually includes examples of the measurements that could be taken to determine the status and effectiveness of the activities performed.

The key practices in the measurement and analysis common feature describe basic measurement practices that are necessary to determine status related to the activities performed. The core suggested measures are in the areas of size, effort, schedule, and quality.

Verifying implementation

The verifying implementation common feature describes the steps to ensure that the activities are performed in compliance with the process that has been established. This common feature generally contains key practices that relate to oversight by senior management, project management, and software quality assurance.

The primary purpose of periodic reviews by senior management is to provide awareness of, and insight into, software process activities at an appropriate level of abstraction and in a timely manner. The time between reviews should meet the needs of the organization and may be lengthy, as long as adequate mechanisms for exception reporting are available.

The scope and content of senior management reviews depends in large part on which senior manager is involved in the review. Reviews by the senior manager responsible for all software project activities of an organization are expected to occur on a different schedule, and address different topics, in comparison with a review by the senior executive of the entire organization. Senior management reviews would also be expected to cover different topics, or similar topics at a higher level of abstraction, than project management oversight reviews.

Main process concepts of CMM

Software process definition

Software process definition is fundamental if higher levels of maturity are to be achieved. A fundamental concept of process definition in CMM is the organization's standard software process. An organization's standard software process is the operational definition of the basic process that guides the establishment of a common software process across the software projects in the organization. It describes the fundamental software process elements that each software project is expected to incorporate into its defined software process. It also describes the relationships (e.g., ordering and interfaces) between these software process elements. It establishes a consistent way of performing the software activities across the organization and is essential for long-term stability and improvement [1].

At the organizational level, the organization's standard software process needs to be described, managed, controlled, and improved in a formal manner. At the project level, emphasis is on the usability of the project's defined software process and the value it adds to the project. A project's defined software process is the operational definition of the software process used by the project. The project's defined software process is a well-characterized and understood software process, described in terms of software standards, procedures, tools, and methods. It is developed by tailoring the organization's standard software process to fit the specific characteristics of the project.

Process definition concepts

A fundamental concept that supports the approach taken by the SEI in its process definition work is that processes can be developed and maintained in a manner similar to the way in which products are developed and maintained. Requirements include a definition of the process to be described, an architecture and a design, implementation of the process design in a project or organizational situation,validation of the process description via measurement, and deployment of the process into widespread operation within the organization or project for which the process is intended.

Using the analogy of product development, a framework for software process development and maintenance has evolved that translates these concepts into ones that are more specific to the process development discipline (similar to the specificity of terminology used for developing real-time embedded systems versus management information systems) [1].

Organization's standard software process

An organization's standard software process is the operational definition of the basic process that guides the establishment of a common software process across the software projects in the organization. It describes the fundamental software process elements that each software project is expected to incorporate into its defined software process. It also describes the relationships (e.g., ordering and interfaces) between these software process elements. It guides the establishment of a common software process across the software development and maintenance projects in the organization.

The relationship between software process elements is sometimes referred to as a *software process architecture*.

The organization's standard software process forms the basis for a project's defined software processes. It provides continuity in the organization's process activities and is the reference for the measurements and long-term improvement of the software processes used in the organization [1].

Software process architecture

The software process architecture is a high-level (i.e., summary) description of the organization's standard software process. It describes the ordering, interfaces, interdependencies, and other relationships between the software process elements of the organization's standard software process. It also describes the interfaces, dependencies, and other relationships to other external processes (e.g., system engineering, hardware engineering, and contract management) [1].

Software process element

A software process element is a constituent element of a software process description. Each process element covers a well-defined, bounded,

closely related set of tasks (e.g., software estimating element, software design element, coding element, and peer review element). The descriptions of the process elements may be templates to be filled in, fragments to be completed, abstractions to be refined, or complete descriptions to be modified or used unmodified.

A software life cycle is the period of time that begins when a software product is conceived and ends when the software is no longer available for use. The software life cycle typically includes a concept stage, requirements stage, design stage, implementation stage, test stage, installation and checkout stage, operation and maintenance stage, and, sometimes, a retirement stage.

Concepts related to the project's defined software process

Description

The description of a project's defined software process is the operational definition of the software process used by the project. The project's defined software process is a well-characterized and understood process, described in terms of software standards, procedures, tools, and methods. It has to be tailored to a standard software process that is accepted within the organization to fit specific characteristics of the project. This tailoring includes selecting a software life cycle model from the models already approved by the organization and modifying the organization's standard software process to fit the specific characteristics of the project.

The project's defined software process provides the basis for planning, performing, and improving activities of the managers and technical staff assigned to the project. It is possible for one project to have multiple defined software processes (e.g., for the operational software and for the test support software) or to have one defined software process for two or more similar projects.

Within the context of process definition, a task is a well-defined component of a defined process. All tasks can be considered activities, but not all activities are defined well enough to be considered tasks (although an activity may include a task). Because of this, use of the word *task* in the

Level 2 key practices is avoided and the less rigorous term *activity* is used. An activity is any step taken or function performed, both mental and physical, toward achieving some objective. Activities include all of the work the managers and technical staff do to perform the tasks of the project and organization.

The results of activities and tasks consist primarily of software work products. A software work product is any artifact created as part of defining, maintaining, or using a software process, including process descriptions, plans, procedures, computer programs, and associated documentation, which may or may not be intended for delivery to a customer or end user. Work products become inputs to the next step in the process or provide archival information on the software project for use in future projects.

Examples of software work products include plans, estimates, data on actual efforts, corrective action documentation, and requirements documents. The subset of software deliverables is referred to as software products [1].

Software products

The software products are the complete set, or any of the individual items of the set, of computer programs, procedures, and associated documentation and data designated for delivery to a customer or end user. All software products are also software work products. A software work product that will not be delivered to a customer or end user is not, however, a software product.

The description of the project's defined software process will usually not be specific enough to be performed directly. Although the description typically identifies such things as roles (i.e., who performs a task) and types of software work products needed to perform a task, it does not specify the individual who will assume the roles, the specific software work products that will be created, nor the schedule for performing the tasks and activities.

The project's software development plan, either as a single document or a collection of plans collectively referred to as a software development plan, provides the bridge between the project's defined software process (what will be done and how it will be done) and the specifics of how the

project will be performed (e.g., which individuals will produce which software work products according to what schedule). The combination of the project's defined software process and its software development plan makes it possible to actually perform the process [1].

Key practices and the CMM

The key practices are not meant to limit the choice of a software life cycle. People who have extensively used one particular software life cycle may perceive elements of that life cycle in the organization and structure of the key practices. However, there is no intent either to encourage or preclude the use of any particular software life cycle.

The term *stage* is used to refer to a defined partition of the software project's effort, but the term should not be tied to any specific software life cycle. As it is used in the key practices, *stage* can mean rigidly sequential stages or overlapping and iterative stages.

The key practices neither require nor preclude specific software technologies, such as prototyping, object-oriented design, or reuse of software requirements, design, code, or other elements.

The key practices describe a number of process-related documents, each one covering specific areas of content. The key practices do not require a one-to-one relationship between the documents named in the key practices and the actual work products of an organization or project; nor is there an intended one-to-one relationship to documents specified by the U.S. Department of Defense or to standards such as DOD-STD-2167A or IEEE software standards. The key practices require only that the applicable contents of these documents be part of the organization's or project's written work products.

Collection and analysis of process data

The key practices for the collection and analysis of the process data evolve across the maturity levels.

At Level 2, the data are primarily related to the size of the project's work products, effort, and schedule, and are defined, collected, and

stored separately by each project. The data are shared between projects via informal procedures [1].

At Level 3, each project has a defined software process tailored from the organization's standard software process. Data related to each project's defined software process are collected and stored in the organization's software process database. The data collected and stored may be different for each project, but the data are well defined within the organization's software process database [1].

At Level 4, the organization defines a standard set of measurements based on the organization's standard software process. All projects collect this standard set of measurement data, as well as other project-specific data, and store them in the organization's software process database. The data are used by the projects to quantitatively understand and stabilize the process performance of the project's defined software processes. They are also used by the organization to establish a process capability baseline for the organization's standard software process [1].

At Level 5, data are used to select areas for technology and process improvements, to plan these improvements, and to evaluate the effects of these improvements on the organization's process capability [1].

Applying professional judgment

To provide a complete set of valid principles that apply to a wide range of situations, some of the key practices are intentionally stated to allow for flexibility. Throughout the key practices, nonspecific phrases like "affected groups," "as appropriate," and "as necessary" are used. The use of such nonspecific terms is generally minimized in the key practices, with examples provided in many cases, at least for the first use of the term. These phrases may have different meanings for two different organizations, for two projects in a single organization, or for one project at different points in its life cycle. Each project or organization must clarify these phrases for its specific situation.

Clarifying these phrases requires the organization to consider the overall context in which they are used. The pertinent question is whether the specific interpretation of one of these phrases meets the goals of the

KPA. Professional judgment must be used to determine whether the goals have been achieved.

Professional judgment must also be used when interpreting the key practices and how they contribute to the goals of a key process area. In general, the KPAs describe a fundamental set of behaviors that all software organizations should exhibit, regardless of their size or their products.

The key practices in the CMM, however, must be interpreted in light of a project or organization's business environment and specific circumstances. This interpretation should be based on an informed knowledge of both the CMM and the organization and its projects. The goals of the KPAs provide a means for structuring this interpretation. If an organization's implementation of a key process area satisfies the goals, but differs significantly from the key practices, the rationale for the interpretation should be documented. A documented rationale will help assessment and evaluation teams understand why certain practices are implemented the way they are.

Applying professional judgment leads to the issue of the "goodness" of the software process. The CMM does not place "goodness" requirements on the software process, although it does establish minimal criteria for a "reasonable" process in many software environments. The objective of process management is to establish processes that are used and can act as a foundation for systematic improvement based on the organization's business needs.

References

[1] Paulk, M. C., et al.,*The Capability Maturity Model: Guidelines for Improving the Software Process,* Reading, MA: Addison-Wesley, 1995.

[2] Senge, P., *The Fifth Discipline,* New York: Currency Doubleday, 1990, p. 21.

[3] Hansen, G. A., *Automating Business Process Reengineering,* Englewood Cliffs, NJ: Prentice Hall, 1997.

[4] Shewhart, W., *Economic Control of Quality of Manufactured Product,* 1931.