Chapter 3

Software engineering

The concept of organizational maturity emerged in early 1990s in the software development industry. It departs from the old concepts of software maturity, which fail to consider the impacts of organizational maturity on software development.

The first model type includes traditional systems analysis techniques. The second model type looks at the SEI Capability Maturity Model (CMM) and the LBMS Process Engineer. The promises and problems of each are considered.

3.1 Software maturity

Software maturity refers to the maturity of the a software development life cycle in developing a product—from concepts to operation. It is based on software engineering principles stating that the models of software development are derived from other engineering activities. Feasibility studies and product design

are some of the engineering activity examples. An overview of systems analysis follows.

3.1.1 Traditional systems analysis

This section discusses three types of traditional system analysis:

- Original waterfall;
- Incremental waterfall;
- Spiral waterfall.

Traditional systems analysis started with the waterfall life cycle approach, as developed in the late 1960s and early 1970s. This approach usually involves four steps: requirements analysis and definition, software design, implementation, and testing.

The process begins with requirements analysis and design. When this step is completed, the process moves on to the software design step. The process continues until the project reaches its end.

This approach works primarily if requirements and software design remain unchanged. However, in the real world the requirements and design do change. In addition, errors in requirements and software design emerge at the end of a project. The risks of changing requirements and software design are unmeasurable.

To overcome some limitations of the original waterfall approach, incremental and spiral waterfall approaches (see Figures 3.1 and 3.2) have been considered. According to Edward Yourdan [1], each increment of software development can be managed as a waterfall cycle. Barry Boehm [2] has suggested a series of "increments" in software development. Yourdan refers to DeGrace and Stahl [3] for the following illustration of an incremental life cycle.

In Figure 3.1, the life cycle consists of three main components: system feasibility/validation, software plans and requirements/validation, and the preliminary product design/verification. The product design component is grouped into increments of detail design/verification. Each increment is treated as a waterfall life cycle of the following phases:

1. Detail design;
2. Code/unit test;
3. Integration/product test;

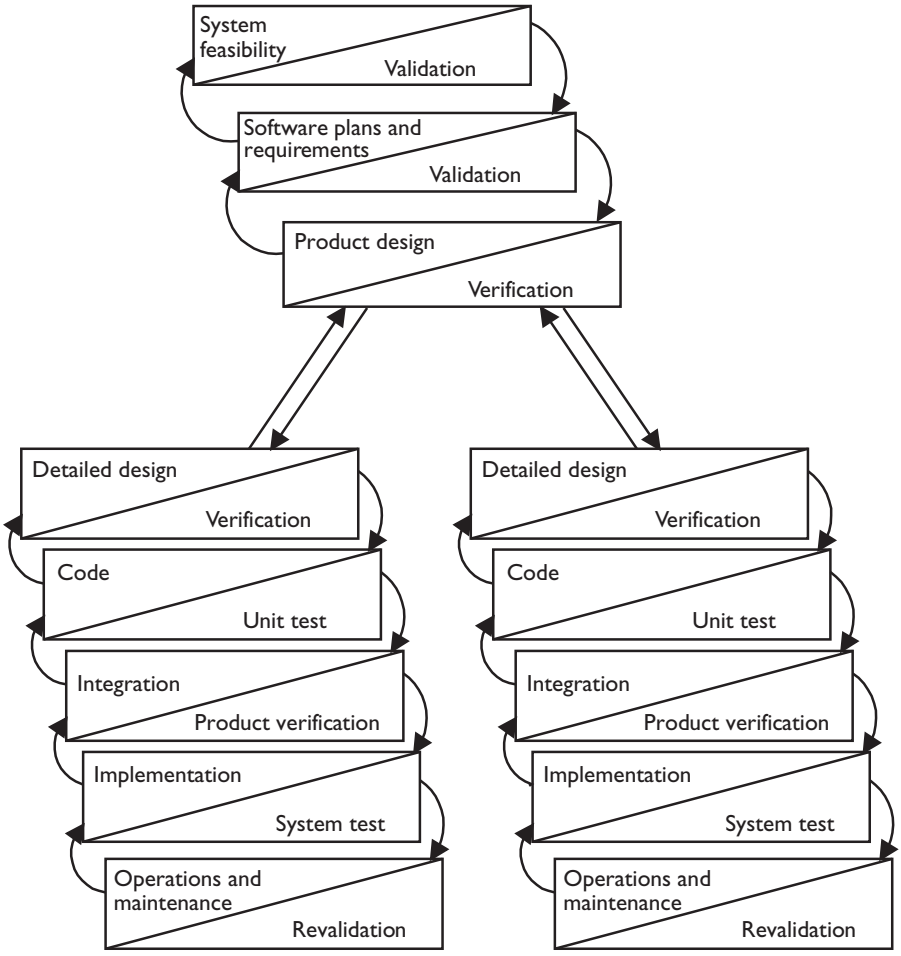

Figure 3.1 The incremental life cycle. (*After:* [1].)

4. Implementation/system test;
5. Operations and maintenance/revalidation.

If an error is found in an increment, the incremental life cycle allows backtracking of the errors in the first previous increment. The backtracking continues until it reaches the increment containing the source of error.

In addition to the series of increments, Barry Boehm has discussed the possibility of developing and managing a software productivity system as a series of "spirals" (within the context of the culture of TRW Defense Systems Group). Unlike the incremental life cycle, the Boehm spiral includes objectives, alternatives, constraints, risk analysis, and prototyping in software development. It identifies and analyzes risks of alternatives for each spiral.

The illustration shows the spiral is divided into four quadrants:

- *Quadrant I*: Determine objectives, alternatives, and constraints.
- *Quadrant II*: Evaluate alternatives, identify, and resolve risks.
- *Quadrant III*: Develop and verify next-level product.
- *Quadrant IV*: Plan next phases.

The horizontal axis divides the spiral into quadrants I and II in the upper portion and quadrants III and IV in the lower portion of the model. It is obvious that the upper portion consists of the objective, alternative, and constraint determination components in quadrant I and the alternative evaluation and risk resolution component in quadrant II, and the lower portion contains the product development and verification component in quadrant III and planning next phases component in quadrant IV. The planning next phase component involves a review of the progress of prior steps and commitment of funds for the next phase.

The spiral divides into four parts. Each is identified with a group function, as follows:

- *Spiral 1*: Concept spiral;
- *Spiral 2*: Simulations spiral;
- *Spiral 3*: Models spiral;
- *Spiral 4*: Benchmarks spiral.

The first spiral is the innermost spiral. Each spiral has its own set of determination, evaluation, development, and planning phases. For example, spiral 2 identifies the risks in alternatives of the requirements validation phase of the life cycle. Spiral 4 identifies the risks of the integration and test alternatives.

The concept spiral starts with the objectives, alternatives, and constraints in the first quadrant. It proceeds to risk analysis of the alternatives and prototype 1 in the second quadrant. The spiral then moves to the concept of operation in the third quadrant. It completes its round with the requirements and life cycle plans in the fourth quadrant.

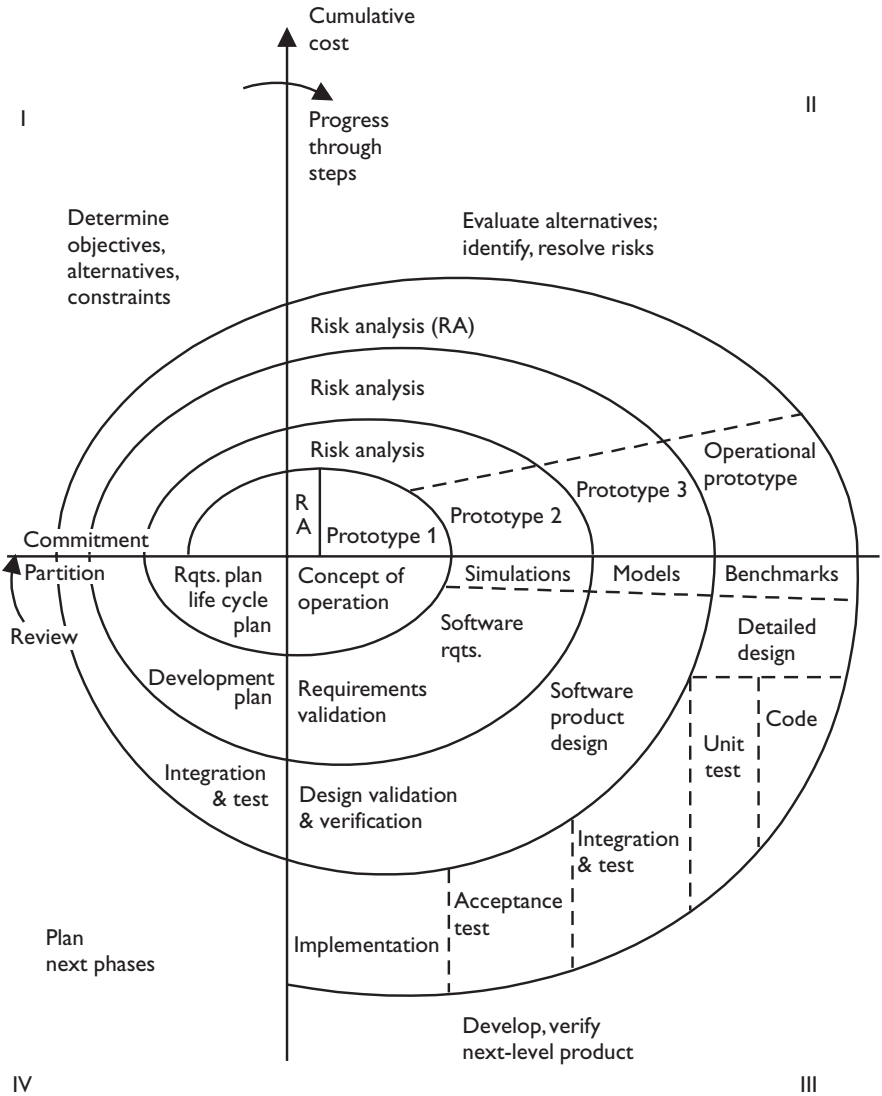

Figure 3.2 The spiral life cycle. (*Source:* [4]. © 1988 IEEE.)

Then, the spiral progresses to the next phase: the simulations spiral. This spiral requires another set of objectives, alternatives, and constraints for the requirements phase. The risks in the requirement alternatives are identified and resolved. The spiral moves to the third quadrant to develop and verify software

requirements and requirements validation. It completes with a development plan to prepare for the models spiral.

In this spiral, objectives, alternatives, and constraints of the development phase are determined. This spiral identifies the risks of development alternatives. It proceeds to develop and verify software product design, design validation, and design verification, and then to plan for the benchmarks spiral.

In this final spiral, the risks in integration alternatives are identified. It moves to the next quadrant to develop and verify detailed design, unit test and code, integration and test, acceptance test, and, finally, implementation.

The vertical axis shows the increase in cumulative cost from the implementation component of the benchmarks spiral in quadrant III to the risk component in the outermost spiral. Risk analysis in the innermost spiral costs the least. The costs between the components on each side of the vertical axis are theoretically similar, such as between the development plan component in quadrant IV and the design validation and verification component in quadrant III (see Figure 3.2).

For easier readability, the author has transformed the spiral life cycle illustration into Table 3.1, with a column of quadrants on the left side and a row of spirals at the top. A user feedback component has been added to the end of spiral 4 (to indicate user input into next software releases).

The Boehm spiral model has its limitations. It is difficult to determine from the table the type of risk analysis methodology used for each spiral. Risk analysis is not an exact science, and appears to be a generic term in the second quadrant in all four spirals. Other components are more specific for each spiral in the third and fourth quadrants. For example, concept of operation, software requirements, software product design, and detailed design specifically belong to spirals 1 through 4 in the third quadrant. They are more identifiable than design as the generic word for the third quadrant.

The spiral progresses through steps four times around the quadrants. Similar to the original waterfall model, the spiral completes the current step and proceeds to the next step. However, errors are not detected until the spiral reaches its end. Correcting the errors may be costly because of the many steps involved in backtracking to the source of errors in the spiral.

3.1.2 Systems analysis with CASE tools

This section gives a general discussion of computer-aided software engineering (CASE) and looks at systems analysis with CASE tools as the middle CASE component. CASE is a software tool that provides automated assistance to software development, software maintenance, or project management.

Table 3.1 Viewing Spiral Life Cycle in Table Format

Action	Spiral 1	Spiral 2	Spiral 3	Spiral 4
Quadrant I: Determine objectives, alternatives, and constraints				
Quadrant II: Evaluate alternatives	Risk analysis	Risk analysis	Risk analysis	Risk analysis
Identify and resolve risks	Prototype 1	Prototype 2	Prototype 3	Operational prototype
Quadrant III: Develop and verify next-level product		Simulations	Models	Benchmarks
	Concept of operation	Software requirements	Software product design	Detailed design
		Requirements validation	Design validation and verification	Unit test
				Integration and test
				Acceptance test
				Implementation
Quadrant IV: Plan next phases	Requirements plan; life plan	Development plan	Integration and test	User feedback

After: [4]

Alan S. Fisher [5] shows how CASE methodologies and tools have evolved since 1965, as follows:

- 1965 Structured methodologies;
- 1970 Data modeling techniques;
- 1975 Database 4GLs and database schema design;
- 1980 Design specification software tools;
- 1985 User prototype tools;
- 1990 Code generation tools.

CASE tools automate existing system analysis and design methodologies that have been practiced in various forms since 1970s and 1980s. Some tools are limited in generating codes from specifications. Full automation of code generation directly from specifications remains an academic subject.

The list should be updated to include the reverse engineering tools for 1995. To date, reverse engineering is primarily limited to high-quality programs written in high-level languages such as COBOL, C, and Ada. It is dependent on the strengths of the CASE tool used. The merits of reverse engineering are reflected in the development time it could save. CASE tools have not yet included industry standard risk methodologies.

Today, more and more vendors are offering integrated CASE (I-CASE) tools—either as a software package for a hardware platform or a set of packages for diverse platforms. These tools facilitates the flow of data in an integrated environment.

Breaking down I-CASE into components makes it easier to understand the frame of reference for discussion. According to Michael Lucas Gibson [6], the components contain upper, middle, and lower CASE tools. Upper CASE refers to a component that supports computer-aided planning. Middle CASE refers to a component that support systems analysis and design. Lower CASE refers to a component that supports systems development (e.g., programming). The concept of middle CASE emerged to reserve and expand the concept of upper CASE for corporate planning.

With the advent of middle CASE tools, the waterfall life cycle has become more manageable. Errors can be checked at any point of the cycle. Changes in analysis requirements and definition are possible to correct the problems in the later stages of the cycle, such as software design and specification. CASE tools encourage the use and reuse of libraries of requirements, designs, and specifications to build a software product.

However, Len Fertuck [7], among others, considers the upper CASE as a component on systems analysis and design (see Figure 3.3). For example, Fertuck suggests that standard upper CASE products include analysis and design tools, data modeling tools, and prototyping tools. The first two tools apply to the requirements, analysis, design, and specification stages of the development life cycle. This means when an error, for example, is found in the design or specification stage, the analysis and design tool can be used to trace the error to the requirements stage. When the error is corrected, the tool automatically generates the changed data for the design or specification stage.

Like others, Fertuck acknowledges that lower CASE products include prototyping tools, coding tools, testing tools, and implementation tools. Prototyping tools are useful for the analysis, design, specification, and code and test

stages. These tools act as a bridge between the upper and lower CASE products. For example, if an error is found in the test stage, the prototyping tool is used to locate the error in the analysis stage. To backtrack the source of error to the requirements stage, the systems and analysis tool picks up the information from the prototyping tool.

Gibson lists three major benefits of middle CASE, but fails to address the risks associated with this CASE component. The first benefit provides easier methods of changing system design via interactive dialog between the analysts and the users. For example, the analysts consider and analyze the users' needs, and document the analysis with diagrams and dictionary entries. Then, the users make suggestions after reviewing the diagrams and dictionary entries. The analysts consider the suggestions and make appropriate changes to them.

The second benefit facilitates joint applications/design sessions. The end users can quickly influence systems analysis and design. System professionals interact with end users to document requirements at the beginning of development projects. The end users do not wait until the end of the project to provide their feedback to the system professionals.

The third benefit involves prototyping facility, which allows the analysts to simulate the screens early in the analysis and design part of the project. Prototyping serves as the blueprint for building screens to browse, access, and update data.

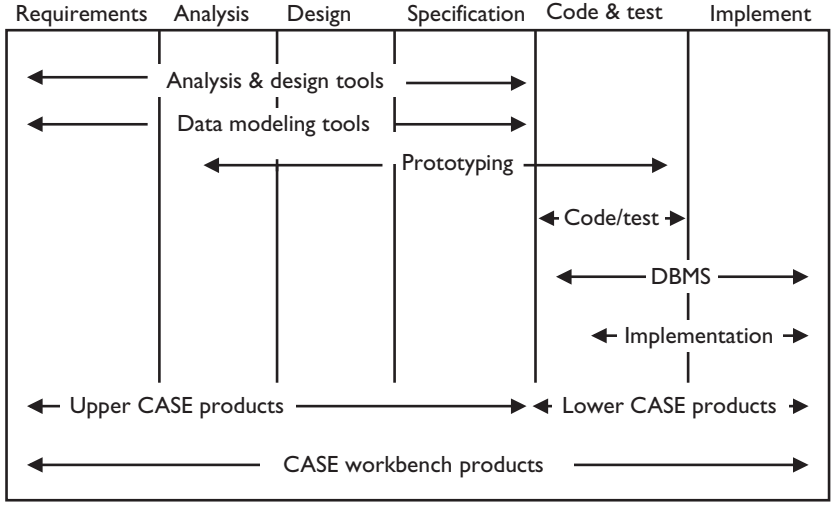

Figure 3.3 Case tools in development life cycle. (*Source:* [7].)

Vessey et al. [8] consider multiuser CASE tools as "collaborative support technologies." They require the collaboration of specialists working together to develop a software product. Vessey's model of a collaborative support environment shows that collaboration technology consists of cooperation technology and coordination technology. Coordination technology is "teamware oriented"; it involves coordination of group activities. Cooperation technology is groupware-oriented; it involves the cooperation of group members to communicate and schedule meeting times about a product.

According to Vessey, an assessment of the cooperation requirements of the CASE tools will require answers to the following questions: Does the tool provide electronic mail facilities? Is it possible to provide anonymous feedback to a team member on his/her work? CASE tools also involve coordination activities in terms of control (namely, access control), information sharing (data sharing, consistency enforcement, and concurrency control) and monitoring (product and user). The following are examples of questions used to assess coordination activities:

Control

Access Control Is it possible to force users to change their passwords periodically? Is it possible to specify read-only passwords for certain parts of the data dictionary? Can one analyst have read-only access to another analyst's work?

Information sharing

Data Sharing Is it possible to simultaneously display the diagrams on all workstations? Is it possible to attach "electronic notes" to objects for all team members to read?

Consistency Enforcement Does the tool automatically notify an analyst whose work might be affected due to a change in the data dictionary? Is it possible to freeze parts of the design work to protect it from changes?

Concurrency Control Is it possible to access the data dictionary concurrently? If concurrent access is controlled by locking, can one query to find who has locked the item of interest?

Monitoring

Product Can the tool flag changes to a data dictionary after a certain date? Can the tool generate reports on every reference to an object in the dictionary?

User For any given user, is it possible to find the most recent login time, date, and session length? For any given user, is it possible to query information on

user activity such as the number of changes made at the last login, or data dictionary import/export operations performed on the dictionary?

It is obvious that most questions, more or less, assess risks associated with collaborative and cooperative activities of the CASE tools.

3.2 Organizational maturity

This section looks at the organizational maturity, rather than software maturity, in developing a product in an organization. SEI's Capability Maturity Model (CMM) is chosen from several software engineering models for discussion on organizational maturity of software development processes. The model presents a framework for process management software to automate software development processes. A discussion of the LMBS Process Engineer product suite follows the section on CMM.

3.2.1 SEI's Capability Maturity Model

This section gives a brief history of the CMM and a discussion of the maturity levels of the model. The section also covers an overview of the People Management Capability Maturity Model (PM-CMM) and the Personal Software Process (PSP). It shows how the maturity framework of the CMM is applied to the PM-CMM and PSP.

The concept of organization maturity emerged in early 1990s in the software development industry. It departs from the old concepts of software maturity that fail to consider the impacts of organizational and management styles on software development processes.

Humphrey [9] shows that software engineering is a better method of controlling and managing software development than other attempts at developing software in a chaotic work environment. In 1989, Humphrey presented a model of the Software Engineering Institute's software process maturity model as a framework for evaluating and improving the process of developing software. The model refers to the maturity of a software organization in improving the processes of developing a product (see Figure 3.4). The focus is shifted from fixing problems to preventing them. The maturity is divided into five levels, as follows:

- *Level 1*: Initial process;
- *Level 2*: Repeatable process;
- *Level 3*: Defined process;

- *Level 4*: Managed process;
- *Level 5*: Optimizing process.

The following gives a brief description for each maturity level of the CMM. Most organizations are at level 1 or level 2.

- *Level 1*: Initial process. Software process development at this stage is semichaotic and depends on individual efforts. It provides an opportunity for an automated approach for development and implementation of software process changes.
- *Level 2*: Repeatable process. This stage looks at project management controls for repeating the project successes. A set of predefined process procedures could be used as templates for several approaches to software development. Yourdon [1] has suggested software commitment management, software planning and cost estimation, configuration management, and change control.

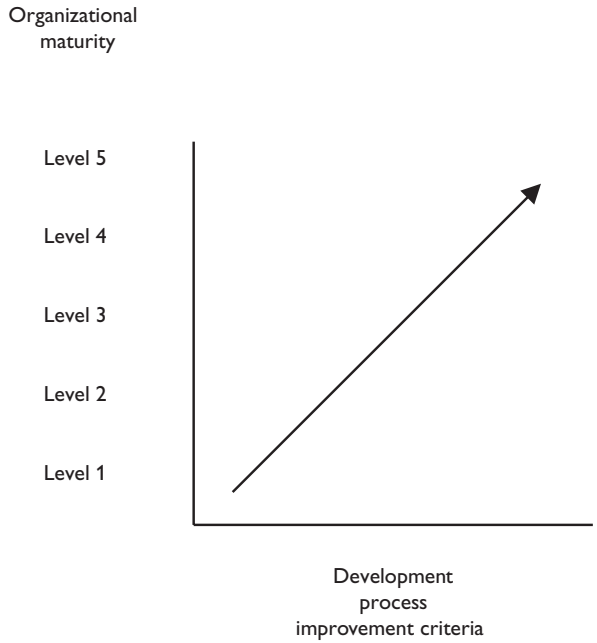

Figure 3.4 Higher maturity level corresponds to improved development processes.

- *Level 3*: Defined process. Process definitions are built from the standardization of development activities in organization-wide software processes. They can be used to tailor the procedures specific to a management or engineering process activity. Yourdon lists the following needed to get to level 3: introduction of formal standards, inspections, formal testing policies, advanced forms of configuration management, formal process models, and establishment of a software engineering process group.
- *Level 4*: Managed process.[1] Detailed quantitative methods, such as data gathering and analyses, are emphasized. They are used to measure the quality of the product and the process by which the product is developed. A comprehensive metrics program would keep track of the defects and the efforts to repair them.
- *Level 5*: Optimized process. The emphasis is placed on the quantitative basis for continued capital investment in process automation and improvement. Innovative process changes and technologies may be used in quantitative feedback.

Martin Thomas [10] points that the CMM indicates "an ordering within process improvement." For example, if an organization is unable to establish effective process management at level 2, the organization will not likely to show any improvements within the process. As a result, the organization will not be able to move to level 3.

One drawback of the CMM is that it emphasizes process, not people. Continued process improvement requires significant changes in the way the development organizations manage people. These are the changes that are not fully accounted for in the CMM. To focus on developing the organization's talent in software and information systems development, PM-CMM was conceived as an adaptation of the CMM. It was felt that as the process of developing people management improves performance, the performance of their teams and projects will improve (see Figure 3.5). Bill Curtis and others [11] show how the four levels of CMM are adapted to people management, as follows:

- The repeatable level focuses on establishing people management practices. They include staffing, performance, training and career, compensation and reward, as well as participatory culture.
- The defined level involves people management planning, knowledge and skills analysis, and competency development. Team building integrates the knowledge and skills needed to accomplish the project tasks.

- The measured level is concerned with establishing a quantitative understanding for the process effectiveness of people management practices. Knowledge, skills, and performance are measured.
- The optimizing level covers the issues of implementing people management continuously. Continuous improvement of knowledge and skills is input from quantitative feedback and adoption of human resources innovation.

Both CMM and PM-CMM are suited for larger organizations that can absorb large overheads. As noted by Humphrey [12], PSP has been developed by the SEI to adapt the principles of software process improvement to small organizations or groups. PSP includes only the CMM items found to be useful for applications at the individual level. It has been found to improve engineers' performance in personal software development process.

Although PSP has a maturity framework of CMM, it is not organized into maturity levels. Instead, PSP progression consists of four components:

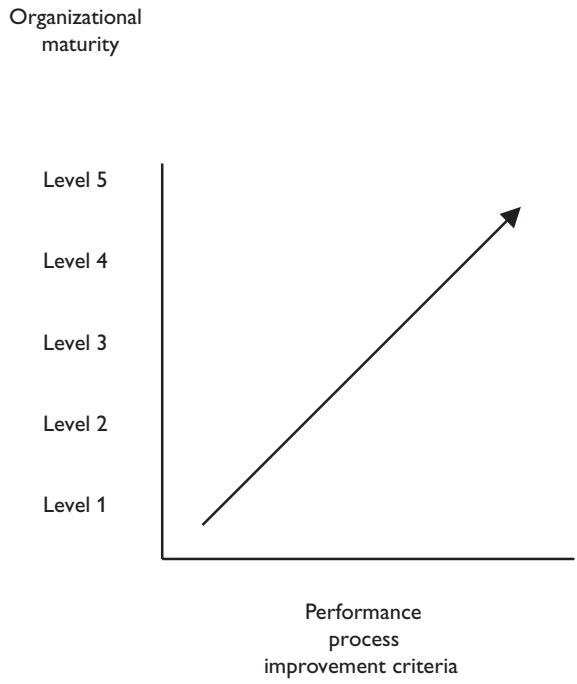

Figure 3.5 Higher maturity level may correspond to improved people performance.

- *PSP0*: The baseline process;
- *PSP1*: The personal planning process;
- *PSP2*: Personal quality management process;
- *PSP3*: A cyclic personal process.

The baseline process establishes a baseline for measuring personal progress in software development. The personal planning process includes size and resource estimations of software development. With the personal quality management process, personal design and code reviews aim to help the engineers to discover defects earlier in their processes.

The cyclic personal process, as described by Humphrey [13], starts with a requirements and planning step for a large program and moves to a high-level design step to partition the program into smaller elements. After this, the elements are developed in PSP cycles. A PSP cycle consists of seven steps: specify code, detailed design and design review, test development and review, implementation and code review, compile, test and reassess, and recycle. The process ends with an integration system test of a product.

The PSP work was conducted at Deimens Corporation Research and the AIS Corporation in Peoria, Illinois. Digital Equipment Corporation and Hewlett Packard Corporation indicated interest in introducing the PSP.

According to Yourdon [1], the following are other drawbacks of the CMM:

1. An organization cannot skip levels; for example, from level 1 to level 4. Much of the transition from one level to the next is cultural. The organization cannot go directly from the semichaotic stage to the stage of formal processes of software development.
2. It may take two or three years to move from one level to the next. On the other hand, mergers or acquisitions could cause the organization to fall back to lower levels. The mergers or acquisitions could result in turnover of critical staff within senior management and technical ranks.
3. Not many organizations are above level 1. As of 1993, the surveys and assessments indicate that 81% of the U. S. sites are at level 1.
4. The organization should not depend on the enterprise-wide introduction of new CASE tools to solve problems at level 1 and level 2. Small-scale, pilot projects on new technologies may continue. The organization must learn to improve development processes.

5. Level 3 is not the place for new software organizations to start. Newly hired people have not worked together as a team.

Three other drawbacks are noted. First, the CMM does not recognize that every project has a unique set of risks. Second, PM-CMM does not consider risks in developing people management. Finally, PSP omits the importance of risk assessment of personal software development process.

3.2.2 LBMS Process Engineer

This section discusses how the CMM provides LBMS, Inc., a maturity level framework for developing Process Engineer (PE) 2.5—a process management software. The product suite helps a development organization automate its attempts to access and improve its capabilities in software development processes.

According to Linda Garrett's product review [14], PE 2.5 is more suited to a large organization that can absorb the overhead costs of automating the model more easily than smaller organizations. It is also adapted to organizations that have proven track records of successes of managing people in a software development workgroup. PE 2.5 assists the organization to start in the initial level of the model. When the organization is ready, PE assists in the move to the next level—one level at a time. PE 2.5 does not allow the organization to skip levels. If the organization falls back by one level (i.e., level 3 to level 2) as a result of a merger or acquisition, PE is useful in returning the organization to a higher level.

PE 2.5 permits project managers to build project plans from methodology templates and automate the tracking and scheduling of application development processes. Users can apply multiple risk analysis and estimating models and customize reports.

PE 2.5 interfaces with a wide range of CASE tools and project scheduling tools, such as Microsoft Project 3.0, Project Workbench 3.0 (Windows) from Applied Business Technology Corp., and Timeline 5.0 for DOS from Symantec Corp. In addition, PE 2.5 can invoke other systems development tools, including LBMS System Engineer CASE tools.

The PE product suite consists of three components: PE/Process Library, PE/Process Engineer, and PE/Process Manager. PE/Process Library is a repository of process templates. Project Managers use PE/Process Engineer, a front-end application, to assess, measure and improve software development processes. PE/Process Manager provides a maintenance tool for the library.

In addition to predefined and customized process templates, the Process Library contains predefined and customized process templates, configurable metric models, and the Process Hyperguide. The templates are used to manage a company's software development activities, such as client/server, rapid delivery, project management, and strategic planning. Managers use metric models to estimate and measure the development process and reconfigure the models in response to changing business requirements.

Process Hyperguide is an online, hypertext reference of work processes and development techniques. The reference contains eight parts: Selecting Adapting, Strategic Planning, Project Management, Client/Server, Rapid Delivery, Classic, Express, and Incremental and Package. It does not reference metric models as one of the main topics.

Process templates are adaptable work breakdown structures (WBS) that provide all the stages, steps, tasks, and end products of a software development process. New or modified templates are built from process kernel modules. PE 2.5 allows the building of a project WBS as a diagram or outline view, and the tool will automatically reconstruct it in the complementary view. Project details are added, such as roles/responsibilities, testing, tools, dependencies, new activities, and new processes. Matrix Editor allows the editing of these details in a matrix form.

PE 2.5[2] provides risk analysis, weighted average, variance, estimate, and function point Albrecht models. Risk analysis assists a project manager to eliminate, reduce, or recognize the impact of risk on the over all project or risk categories (e.g., organizational and technical). It can take place at any point during the project. The relative values of numbers obtained from risk analysis are described as low, moderate, or high risk. Risk analyses are performed by managers who can modify project plans to reflect the results.

Other models look at the quality of the process without consideration for the probable risks to the project. Moreover, PE 2.5, unlike other process management tools, has an expert process analyzer. As the name implies, the analyzer checks the validity of new or modified processes and alerts the user of invalidation.

In PE 2.5, the step-by-step instructions provide onscreen reminders of the sequence of steps to accomplish each function in a project activity. The step-by-step window stays open in front of all PE/Process Engineer or PE/Process Manager screens, providing help for each step. The window can be iconized when not needed. Online guidelines are available to assist the users in selecting and adapting the right process templates.

PE 2.5 divides process management into five levels corresponding to the CMM maturity levels: initial, repeatable, defined, managed, and optimized as described below:

- At the ad hoc level (initial level), PE 2.5 allows an organization to automate the adoption of a methodology in assessing development processes for a project activity. Process success is dependent on individual effort. Formal process for management and development does not exist at this level.
- At the basic control level (repeatable level), PE/Library provides process templates for various approaches to software development. These templates can be reused for other projects. Basic project management controls must be used to enable repetition of project successes via the tracking of schedules, requirements and costs.
- At the process definition level (defined level), PE/Manager tools provide project managers with tools to change the templates or define new process templates in response to changing business requirements. These changes in the software process are documented, standardized, and integrated into organization-wide software processes.
- At the process management level (managed level), PE 2.5 provides configurable metrics and estimating formulas for detailed measurements of the features, facets, and functionalities of the process and the product. The results are collected to compare improvements for both the process and product quality.
- At the process control level (optimized level), the product enables organizations to optimize their software development processes. Quantitative feedback from the process and testing of new ideas and technologies provides continuous process improvements.

Two drawbacks of PE 2.5 are noted. First, the software emphasizes work processes and development techniques; it does not include people management practices. Second, the capability to post electronic notes by remote users is not available.

Notes

1. Level 2 and level 3 organizations measure lines of codes or function points to determine the program size. These organizations also use measurements to count the number of people assigned to a project and the time they spent on the project.
2. PE 2.5.3 provides two metrics models: Contingency and Risk, and Object Based.

References

[1] Yourdon, Edward, *Decline & Fall of the American Programmer*, Englewood Cliffs, NJ: PTR Prentice-Hall, 1993.

[2] Boehm, Barry, "A Spiral Model of Software Development and Enhancement," *Proc. of an Int'l Workshop on the Software Process and Software Environments,* Coto de Caza, Trabuco Canyon, California, 1985.

[3] DeGrace, Peter, and Leslie Hulet Stahl, *Wicked Problems, Righteous Solutions: Catalogue of Modern Software Engineering Paradigms*, Englewood Cliffs, NJ: Yourdon Press/Prentice-Hall, 1990.

[4] Boehm, Barry W., "A Spiral Model of Software Development and Enhancement," *Computer*, May 1988, pp. 61–72.

[5] Fisher, Alan S., *CASE Using Software Development Tools*, Second Edition, New York, NY: John Wiley & Sons, 1991.

[6] Gibson, M. L., "The CASE Philosophy," *BYTE*, Vol. 4, No. 4, April 1989, pp. 209–218.

[7] Fertuck, Len, *Systems Analysis and Design with CASE Tools*, Dubuque, IA: William C. Brown Publishers, 1992.

[8] Vessey, Iris, and V. P. Sravanapudi., "CASE Tools: Collaborative Support Technologies," *Communications of the ACM*, Vol. 35, No. 1, Jan. 1995, pp. 83–94.

[9] Humphrey, Watts S., *Managing the Software Process*, Reading MA: Addison-Wesley, 1990.

[10] Thomas, Martin, "Top-Down vs. Bottom-Up Process Improvement," *IEEE Software*, Vol. 11, No. 4, July 1994, p.12.

[11] Curtis, Bill B., W. E. Hefley, and S. M. D. Konard, *People Management Capability Maturity Model, Draft Version 0.2 (for public review)*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November 1994.

[12] Humphrey, Watts S., "The Personal Software Process," *Software Process Newsletter (Software Engineering Technical Newsletter)*, IEEE Computer Society Technical Council on Software Engineering, Vol. 13, No. 1, Sept. 1994.

[13] Humphrey, Watts S., "The Personal Software Process Paradigm" (Tutorial Presentation), 1994 Software Engineering Process Group National Meeting, Dallas, TX, April 25–28, 1994.

[14] Garret, Linda, "Product Review: LBMS' Process Engineer 2.5," *Application Development Trends*, Software Productivity Group, Inc., Vol. 1, No. 12, Northboro, MA, Nov. 1994, pp. 75–76.