

Using the Software CMM with Good Judgment

MARK C. PAULK

Software Engineering Institute, Carnegie Mellon University

The Software Engineering Institute's Capability Maturity Model for software (CMM) has had a major influence on software process and quality improvement. Although the CMM has been widely adopted, there remain many misunderstandings about how to use it effectively for business-driven software process improvement, particularly for small organizations and small projects. This article discusses how to use the CMM correctly and effectively in any business environment, with examples for small organizations, rapid prototyping projects, maintenance shops, R&D outfits, and others. The conclusion is that the issues associated with interpreting the software CMM are essentially the same for any organization interested in improving its software processes—the differences are of degree rather than kind. Using the software CMM effectively and correctly requires professional judgment and an understanding of how the CMM is structured to be used for different purposes.

Key words: Capability Maturity Model, CMM, small organizations, small projects, software capability evaluation, software process assessment, software process improvement.

INTRODUCTION

The Software Engineering Institute (SEI) is a federally funded research and development center established in 1984 by the U. S. Department of Defense (DoD) to improve the state of practice in software engineering. The SEI's existence is, in a sense, the result of the "software crisis"—software projects that are chronically late, over budget, with less functionality than desired, and of dubious quality.

Much of the software crisis is self-inflicted, as when a chief information officer says, "I'd rather have it wrong than late. We can always fix it later." The emphasis in too many organizations is on achieving cost and schedule goals, often at the cost of quality. To quote Tom DeMarco (1995), this situation is the not-surprising result of a combination of factors:

- "People complain to us [software developers] because they know we work harder when they complain."
- "The great majority [report] that their software estimates are dismal...but they weren't on the whole dissatisfied with the estimating process."
- "The right schedule is one that is utterly impossible, just not obviously impossible."

DeMarco goes on to observe that the software industry is over goaded, and the only real (perceived) option is to pay for speed by reducing quality. This violates the principle of total quality management (TQM), which says that focusing on quality leads to reduced cycle time, increased productivity, greater customer satisfaction, and business success. Admittedly, the challenge of determining "good enough" is an ongoing debate within the software community and a delicate business decision, but the quality focus is central to the SEI's work.

Perhaps the SEI's most successful product is the Capability Maturity Model for software (CMM), a roadmap for software process improvement that applies TQM ideas to software and has had a major influence on the world software community (CMU SEI 1995). The software CMM defines a five-level framework for how an organization matures its software process capability. These levels describe an evolutionary path from ad hoc, chaotic processes to mature, disciplined software processes. The five levels and the 18 key process areas that describe them in detail are summarized in Figure 1.

This article addresses how the CMM can be effectively used in a variety of environments, with a focus on small organizations but also including examples for rapid prototyping projects, maintenance shops, and research and development (R&D) outfits. The observations and recommendations of the author, based on more than a decade of experience in CMM-based assessments and process improvement, are summarized here. The recommendations may appear dogmatic, but it seems unlikely that many will disagree with their intent. There is likely to be some debate, however, over what constitutes an "adequate" implementation in any given environment.

Although the focus of the current release of the software CMM, version 1.1, is on large organizations and large government projects, it is written in a hierarchical format that runs from "universally true" abstractions for software engineering and project

management to detailed guidance and examples. The key process areas in the CMM are satisfied by achieving goals, which are described by key practices, subpractices, and examples. The rating components of the CMM are maturity levels, key process areas, and goals. The other components are informative and provide guidance on interpreting the model. Although the requirements for the CMM can be summarized in the 52 sentences that are the goals, the supporting material comprises nearly 500 pages of information. The practices and examples describe what good engineering and management practices are, but they do not prescribe how to implement the processes.

Although key practices are not requirements, they are intended to be generally applicable. The goals of each key process area address end states, and each key practice contributes to achieving one or more goals. Although they set expectations, the key practices are not required; there may be alternative methods for achieving a goal. Assessment teams usually find a few key practices—typically three to five—where alternate implementations are used that satisfy the goals of a key process area. This is a large enough percentage of the 316 key practices in the CMM that the need for judgment is clear, but small enough to indicate that the key practices are generally good guidance.

About 10 percent to 15 percent of the key practices usually have to be interpreted, that is, the team

FIGURE 1 An overview of the software CMM

Level	Focus	Key Process Areas
5 Optimizing	<i>Continuous process improvement</i>	Defect prevention Technology change management Process change management
4 Managed	<i>Product and process quality</i>	Quantitative process management Software quality management
3 Defined	<i>Engineering processes and organizational support</i>	Organization process focus Organization process definition Training program Integrated software management Software product engineering Intergroup coordination Peer reviews
2 Repeatable	<i>Project management processes</i>	Requirements management Software project planning Software project tracking and oversight Software subcontract management Software quality assurance Software configuration management
1 Initial	<i>Competent people and heroics</i>	

©1999, ASQ

has to discuss at length whether an implementation is adequate. Key practices are not requirements, and there may be alternate implementations. This does not, however, abrogate the responsibility to make informed, reasonable, and professional judgments about each key practice and its associated goals—and assessment findings may be written against key practices and subpractices when an implementation is judged inadequate.

Although the CMM is a common-sense application of TQM concepts to software that was developed with broad review by the software community, small organizations may find the large organization/project orientation of the CMM problematic. Its fundamental concepts are useful to any size organization in any application domain and for any business context. If an organization's employees are satisfied with the status quo, there is little the CMM can provide that will lead to true change. Change occurs only when there is enough dissatisfaction with the status quo that managers and staff are willing to do things differently. This is as true for small organizations as large.

Intelligence and common sense are needed to use the CMM correctly and effectively (Paulk 1996), and this is true in all environments. Based on more than a decade's experience in software process work, environments where interpretation and tailoring of the CMM are needed include:

- Very large programs, with many organizations interacting
- Virtual projects or organizations
- Geographically distributed projects
- Rapid prototyping projects
- R&D organizations
- Maintenance (sustaining engineering) shops
- Software services organizations
- Small projects and organizations

SMALL ORGANIZATIONS AND SMALL PROJECTS

This article's emphasis on small organizations is due to the frequently asked question, "Can the software CMM be used for small projects (or small organizations)?" Yet the definition of "small" is ambiguous, as illustrated in Figure 2. At one time there was an effort to develop a

FIGURE 2 Defining a "small" project

Variant of "Small"	Number of People	Duration
Small	3-5	6 months
Very small	2-3	6 months
Tiny	1-2	2 months
Individual	1	1 week
Ridiculous	1	1 hour

©1999, ASQ

tailored CMM for small projects and organizations, but no one could agree on what "small" really meant. The result was a report on how to tailor the CMM rather than a tailored CMM for small organizations (Ginsberg and Quinn 1995). In a 1998 Software Engineering Process Group (SEPG) Conference panel on the CMM and small projects (Hadden 1998b), small was defined as "three to four months in duration with five or fewer staff." Judith Brodman and Donna Johnson (1996; 1997) define a small organization as having fewer than 50 software developers and a small project as having fewer than 20 developers.

Small-to-tiny projects are addressed by Watts Humphrey (1995) in his Team Software Process (TSP) work, and the individual effort is directly addressed by the Personal Software Process (PSP). TSP and PSP illustrate how CMM concepts are being applied to small projects. The "ridiculous" variant represents an interpretational problem. On the two occasions this variant has been discussed, the problem was the definition of "project." In both cases it was a maintenance environment, and the organization's projects would have been described as tasks in the CMM; the more accurate interpretation for a CMM project was a baseline upgrade or maintenance release, but there was a confusing terminology clash.

Appropriately interpreted, the CMM is being used effectively in organizations with fewer than 15 employees and for projects with as few as two people. The SEI's maturity profile as of December 1998 shows that 27 organizations with fewer than 25 employees have performed CMM-based appraisal for internal process improvement (CBA IPI) assessments.

Small organizations, like large ones, will have problems with undocumented requirements, the mistakes of inexperienced managers, resource allocation, training, peer reviews, and product documentation. The challenge of providing resources for process improvement—both to identify problems and systematically address them—will often result in these resources being part time rather than full time. The

business question is whether the kind of process discipline advocated by the CMM is needed by small projects and organizations. To answer this question, one must consider what discipline involves—and that leads to the heart of this article's CMM interpretation discussion.

IMPROVEMENT ISSUES FROM THE IDEAL PERSPECTIVE

The CMM is a tool that should be used as a systematic approach to software process improvement, such as the SEI's IDEAL model (McFeeley 1996), which depicts the activities of an improvement program in five phases:

1. Initiating (the improvement program)
2. Diagnosing (the current state of practice)
3. Establishing (the plans for the improvement program)
4. Acting (on the plans and recommended improvements)
5. Learning (the lessons learned and the business results of the improvement effort)

Obtaining senior management sponsorship in the *initiating* phase is a critical component of building organizational capability. People exercise professionalism and discipline within their sphere of control, but if an organization as a whole is to change its performance, its senior management must actively support the change. Bottom-up improvement, without sponsorship and coordination, leads to islands of excellence rather than predictably improved organizational capability. For small organizations, while the president (or founder) is the primary role model, a respected "champion" frequently has the influence to move the entire organization—including the president.

An opening question should always be: "Why is the organization interested in using the software CMM?" If the desire is to improve process, with a direct tie to business objectives and a willingness to invest in improvement, then the CMM is a useful and powerful tool. If the CMM is simply the flavor of the month, then it is a prescription for disaster. If the driver is customer concerns, ideally the concerns will lead to collaborative improvement between customer and supplier. Sometimes the supplier's concerns center on software capability evaluations (SCEs), such as those performed

by government acquisition agencies in source selection and contract monitoring. DoD policies on the criteria for performing SCEs would exclude most small organizations and small projects (Barbour 1996). There are circumstances, however, under which SCEs may still occur, such as field training an evaluation team.

The relationship to existing TQM programs should be identified in the initiating phase. Software process improvement efforts should be aligned with corporate quality improvement initiatives since the goals are the same, even if the scopes initially differ. TQM programs are relatively uncommon in small organizations, although following the principles of TQM is always recommended.

When assessing small organizations in the *diagnosing* phase, it is best to use a streamlined assessment process. A full-blown CBA IPI (Dunaway and Master 1996), with its criteria for validity, accuracy, corroboration, consistency, and sufficiency, can last two weeks, which might be excessive for small organizations (Strigel 1995; Paquin 1998; Williams 1998). The emphasis should be on identifying important problems, even if some are missed because of a lack of rigor. The CBA IPI assessment method can be tailored down, however, and there are other assessment methods, such as interim profile, that may be more appropriate (Whitney et al. 1994; Daskalantonakis 1994).

Perhaps the best recommendation regarding CMM interpretation is to develop a mapping between CMM terminology and the language used by the organization. In particular, terms dealing with organizational structures, roles, and relationships, and formality of processes need to be mapped into their organizational equivalents. Examples of organizational structures include independent groups, such as quality assurance, testing, and configuration management. Appropriate organizational terminology for roles such as project manager and project software manager should be specified. People may fill multiple roles; for instance, one person may be the project manager, project software manager, software configuration management (SCM) manager, and so on. Explicitly stating this makes interpreting the CMM simpler and more consistent.

Small organizations have an opportunity in the *establishing* phase that large organizations would find difficult: combining levels 2 and 3. Level 2 focuses on projects, yet a small organization should find it

comparatively easy to develop organizational standards at the same time it is defining its project-level processes, since there is much less cultural inertia to overcome.

The most effective organizational learning strategy is likely to be one stressing organizational assets that lessen project overhead. In large organizations resistance to change makes this strategy problematic; worker participation in improvement activities is crucial to deployment but more difficult to orchestrate. Even in small organizations there may be resistance to change, perhaps based on valid concerns, and addressing resistance should be part of the organization's learning process. Similarly, even small organizations have to focus on the "vital few" improvement issues, especially given their limited resources, which reemphasizes the level 2 priorities for improvement.

During the *acting* phase organizations should take advantage of the more detailed components of the CMM: subpractices and examples. These informative components are useful in characterizing an adequate process, yet do not specify a particular implementation. For example, the estimating practices in software project planning have subpractices on using historical data, but do not specify a cost model, such as COCOMO, Price-S, or Slim, or even state that a cost model should be used.

The loop to the *learning* phase may be tighter for small organizations than for large organizations. A one-year improvement cycle may be realistic for small organizations, whereas large organizations will usually take two to three years between assessments (Note that results for specific improvement actions should be observed well before the organizational improvement cycle delineated by an assessment rolls around). This reinforces the need for a low-overhead assessment process.

WHERE DOES THE SOFTWARE CMM APPLY?

The CMM is intended to provide good software engineering and management practices for any project in any environment. The model is described in a hierarchy, as shown in Figure 3.

The *normative* components of the CMM are maturity levels, key process areas, and goals. All practices in the CMM are *informative* as opposed to

FIGURE 3 The CMM structural hierarchy

Maturity levels	(five levels)
→ Key process areas (KPAs)	(18 KPAs)
→ Goals	(52 goals)
→ Key practices	(316 key practices)
→ Subpractices and examples	(many)

©1999, ASQ

normative. Since the detailed practices primarily support large, contracting software organizations, they are not necessarily appropriate, as written, for direct use by small projects and organizations. They do, however, provide insight into achieving the goals and implementing repeatable, defined, measured, and continuously improving software processes.

Key process areas and goals are usually relevant to any environment, with the exception of software subcontract management, which may not be applicable if there is no subcontracting. In contrast, there are no circumstances under which peer reviews could be reasonably tailored out for a level 3 organization. Deciding what is not applicable or an alternate implementation is a matter of professional judgment, implying a need for trained, experienced assessors and process definers, even for small organizations.

Getting to Level 2

At level 2, the CMM emphasis is on managing software projects. The question for small organizations is whether the ability to *manage* projects is a crucial business concern. If the organization is doing contract work, the answer is usually yes. If the organization is doing commercial shrinkwrap development, for instance, this issue is more debatable. Will bad management lead to serious business consequences? For many successful shrinkwrap software companies, no new product has ever been finished on time or provided the functionality originally envisaged (Moody 1995). It is unlikely, however, that bad management has contributed to the success of the thriving shrinkwrap companies, yet it has certainly contributed to the failure of many that have not survived.

Requirements management

Documented customer (system) requirements and communication with the customer (and end users) are always important, although the documentation may be as simple as a one-page letter of intent. Documenting the requirements can be a challenge in a rapid prototyping environment or for R&D. The

requirements may be scattered across several prototypes and actively evolving. This is acceptable so long as the history is maintained and the requirements are consolidated before getting out of hand (that is, too many prototypes capturing different potential requirements or before making the transition to full-scale development). The customer requirements/prototype functionality are similar to a lab notebook in the R&D context. Always document commitments and the requirements for the work to be performed—these documents are crucial for clarification and conflict resolution as the project progresses.

Software project planning

The number-one factor in successful process definition and improvement is “planfulness” (Curtis 1996). Planning is needed for every major software process (the organization determines what “major” means and how the plan should be packaged). A plan may reside in several different artifacts or be embedded in a larger plan.

Maintenance work is frequently level of effort. Problem reports and change requests are used to identify tasks that are performed in a certain order. A project is the next baseline update or major revision, which consists of a collection of changes to the baselined software that must be carefully controlled. Similarly, R&D projects are likely to be level of effort. Examples of key practices that may provide minimal value to small projects include the size estimating and risk identification practices—and both may be critical for many small projects. For small projects, however, effort and schedules may be estimated directly from a work breakdown structure, and the consequences of the project failing may be relatively minor.

A project management plan that includes a work breakdown structure and any external commitments is always important, although it may be simple and concise. It is also good practice to document internal commitments, especially those that cross organizational boundaries.

Software project tracking and oversight

Knowing what one has accomplished vs. what one has committed to is always important. Admittedly, managers must be careful not to over control, such

as asking for more progress data than provides true insight. Gantt charts and earned value are popular and reasonably effective mechanisms for tracking progress, with two caveats. First, work packages should be binary—done or not done—since detailed estimates of work completed (for example, being 87 percent done) are notoriously inaccurate. This has implications for how the packages should be defined. A useful rule of thumb is to track work at a granularity somewhere between two to three weeks for a work package and two to three work packages per week. Second, remember the critical path to project completion (and possible resource conflicts) when tracking progress.

Management by fact is a paradigm shift for most organizations, which must be based on a measurement foundation. To make data analysis useful, one must understand what the data means and how to analyze it, which implies collecting a simple set of useful data. The data collection effort needs to be minimized so small organizations are not overwhelmed by overhead. Although this is also a problem for large organizations, the margin for error for small organizations is razor thin.

Software subcontract management

For small organizations, rapid prototyping projects, maintenance shops, and R&D outfits, software subcontract management will usually not be applicable.

Software quality assurance

A small project is unlikely to have an independent software quality assurance (SQA) group, but it is always important to objectively verify that requirements are satisfied, including the process requirements in standards and procedures. It is usually a good idea to embed the quality assurance function in the process via checklists, tools, and so on, even if an independent SQA group exists.

Software configuration management

A small project is unlikely to need an SCM group or change control board, but configuration management and change control are always important. Always baseline work products and control changes to them. Microsoft, for example, uses daily builds to ensure that a stable baseline always exists (McConnell 1996).

Closing level 2 thoughts

Many of the context-sensitive, large-project implementation issues relate to organizational structure. Similar to the SCM and SQA examples mentioned previously, an independent testing group may not be established, but testing is always necessary. The intent of a practice is important even if the implementation is radically different between small organizations and large. The CMM definition of “group,” states that “a group could vary from a single individual assigned part time, to several part-time individuals assigned from different departments, to several individuals dedicated full time.” This flexibility (or ambiguity) is intended to cater to a variety of contexts. Independence is a consequence of organizational structure. Objectivity is the concept emphasized in the key process area goals.

Getting to Level 3

Organization process focus

At level 3, the CMM emphasis is on consistency and organizational learning. In most organizations, an SEPG or some equivalent should be formed to coordinate process definition, improvement, and deployment activities. One of the reasons for dedicating resources to an SEPG is to ensure follow through on appraisal findings. Many improvement programs have foundered simply because no action resulted from the appraisal. Small organizations may not have full-time SEPG staff, but the responsibility for improvement should be assigned and monitored. Thinking about the way one works and how to do better is always important, regardless of the models or standards used to structure one’s thoughts. Always assign responsibility, authority, and accountability for process definition and improvement, whether an SEPG is formed or not.

Organization process definition

The reasons for documenting a process (or product) are to:

- *Communicate.* Communicate to others now and perhaps to oneself later.
- *Understand.* If one cannot write it down, he or she does not really understand it.
- *Encourage consistency.* Take advantage of repeatability.

This is a general characteristic of learning organizations—even small R&D or maintenance organizations. Documented processes are always important, and important processes should always be documented.

Documented processes support organizational learning and prevent reinventing the wheel for common problems. Documents do not, however, need to be lengthy or complex to be useful. Keep the process simple. The CMM is about *doing* things, not *having* things. A one-to-two page process description may suffice, and subprocesses and procedures can be invoked as needed. Use good software design principles such as locality, information hiding, and abstraction in defining processes.

The degree of formality needed for processes is a challenge for both large and small organizations (Comer 1998; Hadden 1998a; Pitterman 1998; Sanders 1998). Should there be separate procedures for each of the 25 key practices at level 2 that mention “according to a documented procedure”? The answer, as discussed in section 4.5.5 of *The Capability Maturity Model: Guidelines for Improving the Software Process* (CMU SEI 1995), is a resounding NO! Packaging of documentation is an organizational decision.

Training program

The reason for training is to develop skills. There are many training mechanisms other than formal classroom training that can be effective in building skills. One that should be seriously considered is a formal mentoring program. In this case, formality means going beyond assigning a mentor and hoping that experience will rub off. Formality implies training people on how to mentor and monitoring the effectiveness of the mentoring.

Training remains an issue after the initial deployment of a process or technology (Abbott 1997; Williams 1998). As personnel change, the need for training may not be adequately addressed. Mentoring and apprentice programs may suffice, but they must be carefully monitored.

Integrated software management

The thoughtful use of organizational assets (overcoming the not-invented-here syndrome) is always important. Processes need to be tailored to the needs of the project (Ginsberg and Quinn 1995; Ade and Bailey 1996; Ahlgren 1996). Although standard

processes provide a foundation, most projects have unique needs. Unreasonable constraints on tailoring can lead to resistance to following the process. As Leo Hoffman (1998) expresses it, “Don’t require processes that don’t make sense.” Use organizational assets, but use them intelligently.

Some argue that software project management is really risk management. This implies that one should use an incremental or evolutionary life cycle. If risk management is the focus, the spiral model may be the preferred life-cycle model. If involving the customer is the focus, perhaps rapid prototyping or joint application design would be preferable. Few long-term projects have the luxury of the stable environment necessary for the waterfall life cycle to be the preferred choice—yet it is probably the most common. For small projects, however, the waterfall life cycle may be an excellent choice. In some cases, the risk to the organization if a small project fails is minimal, and formal risk management is not worth the overhead. R&D organizations, on the other hand, are continually pushing the envelope as they explore new areas, thus their intrinsic life cycle is evolutionary.

Software product engineering

Although some may disagree on the need for requirements analysis and design in small projects, thoughtfully defined and consistent software life-cycle processes—requirements analysis, design, coding, testing, installation, operations, and maintenance—are always important. Always spend significant time on requirements analysis, design, and test planning, and always consider the entire life cycle in planning a development project.

Intergroup coordination

For small projects and organizations, intergroup coordination may appear inapplicable because there are no other groups to coordinate with. This misses the point that this key process area is about communicating with the customer, documenting and tracking commitments, and resolving conflicts, which are as crucial for individuals as for different organizational entities. Communication and coordination are always important—even for one-person projects.

Peer reviews

Although one can argue over the best kind of peer review, the benefits of peer reviews far outweigh costs.

Most effective is some variant of inspections, but any form of collegial or disciplined review, such as structured walkthroughs, adds significant value. R&D organizations reflect this by emphasizing the scientific method. Small organizations, however, may be more vulnerable to schedule pressure. Unfortunately, recognizing the value of peer reviews does not mean they are done systematically, thus their placement at level 3. Peer reviews are always important and even recommended for level 1 projects where the chaotic environment makes their use inconsistent.

Getting to Levels 4 and 5

Organizations embarking on levels 4 and 5 need little guidance in CMM interpretation principles. Small organizations should seriously consider the PSP and TSP to bootstrap their process improvement efforts to level 5 (Ferguson and Kitson 1997; Hayes and Over 1997). Where the CMM addresses the organizational side of process improvement, PSP addresses building the capability of individual practitioners. The PSP course convinces individuals, based on their own data, of the value of a disciplined, engineering approach to building software. PSP and TSP take individuals and teams to a level 5 process capability that the organization can leverage.

Miscellaneous Improvement Issues

Sherry Paquin (1998) identifies five issues for small organizations and projects:

- Assessments
- Project focus
- Documentation
- Required functions
- Maturity questionnaire

Since there are no “shall” statements in the CMM, there are no “required functions,” but the CMM may describe more functions than there are people to fill them. This issue has been discussed as terminology or role mapping. The maturity questionnaire is a concern because it uses CMM terminology, which may be unclear to the people filling out the questionnaire. Therefore, wording the questionnaire in the organization’s terminology

is recommended for even an informal assessment or survey.

John Abbott (1997) identifies six keys to software process improvement in small organizations:

- Senior management support
- Adequate staffing
- Applying project management principles to process improvement
- Integration with ISO 9001
- Assistance from process improvement consultants
- Focus on providing value to projects and to the business

Senior management support and adequate staffing are universal issues. If applying good project management to software projects is the best way to ensure success, then the same should be true for process improvement, which should be treated like any other project. ISO 9001 is more of an issue for large organizations than small, so one might find it interesting that Abbott points this out for his small company. The advice to use process improvement consultants can be problematic. The experience and expertise of a good consultant is valuable, yet skilled consultants are costly—often too costly for a small organization. Unskilled consultants can be actively detrimental. Although none of the process guidance in the CMM or similar models and standards is rocket science, the pitfalls in changing individual, team, and organizational behaviors are nontrivial. There are no easy answers to this issue.

Brodman and Johnson (1997) identify seven small organization/small project challenges:

1. Handling requirements
2. Generating documentation
3. Managing projects
4. Allocating resources
5. Measuring progress
6. Conducting reviews
7. Providing training

They have also developed a tailored version of the CMM for small businesses, organizations, and projects (Brodman and Johnson 1996). Although the majority of the key practices (but few of the goals) in the CMM

were tailored in the LOGOS Tailored CMM, they characterize these changes as:

- Clarification of existing practices
- Exaggeration of the obvious
- Introduction of alternative practices (particularly as examples)
- Alignment of practices with small business/small organization/small project structure and resources
- Their analysis therefore agrees that the changes involved in tailoring the CMM for small organizations should not be considered radical.

CONCLUSION

To summarize this discussion and capture the essence of level 3 in language that may communicate better with small projects and organizations, the recommendations for effective software processes include:

- Document the requirements.
- Define a work breakdown structure and plan the work.
- Track significant accomplishments (no more than two or three per week).
- Build the SQA function into the process (as a buddy system or part of peer reviews, with an escalation mechanism to resolve conflicts).
- Determine how changes to work products will be identified and approved.
- Establish a configuration management system.
- Assign responsibility for defining and improving specific processes.
- Document both management and engineering processes and standardize them.
- Document commitments, and systematically resolve conflicts within the project.
- Install a peer-review process, with a preference for inspections.

This may seem simplistic, but it is the core of a disciplined process that can lead even a small software organization to level 5.

Many abuses of the software CMM spring from a fear of what others might do, such as evaluate simple

or alternative implementations adversely, thus leading to loss of a contract. If an organization uses the CMM as guidance rather than requirements, many of the model's interpretation problems vanish. Although the CMM provides significant guidance in making judgments, removing subjectivity implies a deterministic, repetitive process that is not characteristic of engineering design work. That is why it is an abuse of the CMM to check off practices for conformance. When the CBA IPI method requires collecting data on each key practice, it is for making consistent and comprehensive judgments; it is not implying that each key practice be literally implemented. Even weaknesses against a key practice that result in findings do not necessarily result in failure to achieve a goal.

Some are unwilling or unable to interpret, tailor, or apply judgment. It is easy to mandate the key practices, but foolhardy, even when driven by concerns about customer intentions and competence. People might know they are doing something foolish but are afraid the customer will not understand the rationale for doing things differently than described in the CMM. This is particularly problematic if SCEs by the customer are feared. It is true that judgments may differ—and sometimes legitimately so. What is adequate in one environment may not suffice for a new project. That is why process maturity should be included in risk assessment during source selection, rather than using maturity levels to filter offerors (Barbour 1996).

Unfortunately, there is no simple solution to this problem. In the SEI's CMM training, these points are repeatedly emphasized, but the problems persist. Standards such as the CMM can help organizations improve their software process, but focusing on achieving a maturity level without addressing the underlying process can cause dysfunctional behavior. Maturity levels should be measures of improvement, not goals of improvement, which is why improvement should be tied to business objectives.

The challenges in interpreting the CMM appropriately for different environments differ in degree rather than kind. The bottom line is that software process improvement should be done to help the business—not for its own sake. The best advice comes from Sanjiv Ahuja, president of Belleore: “Let common sense prevail!” The CMM is a proven tool to support process appraisal and software process improvement

in a wide range of environments, but it must be used with professional judgment and common sense to be truly effective.

ACKNOWLEDGMENTS

I would like to thank the folks who reviewed this article: Judi Brodman, Rita Hadden, Stuart Locklear, Henry Mendenhall, Gladys Mercier, Sherry Paquin, Kathy Paulk, Pedro Pinto, Marty Sanders, Francisco Valeriano, and Bin-Lan Woo, plus the anonymous reviewers. Any errors are sadly mine.

REFERENCES

- Abbott, John J. 1997. Software process improvement in a small commercial software company. In *Proceedings of the 1997 Software Engineering Process Group Conference, San Jose, Calif.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.
- Ade, Randy W., and Joyce P. Bailey. 1996. CMM lite: SEPG tailoring guidance for applying the capability maturity model for software to small projects. In *Proceedings of the 1996 Software Engineering Process Group Conference: Wednesday Papers, Atlantic City, N. J.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.
- Ahlgren, Magnus. 1996. CMM light for SMEs. *Conference Notebook: The First Annual European Software Engineering Process Group Conference, Amsterdam, The Netherlands.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.
- Barbour, Rick. 1996. *Software capability evaluation version 3.0 implementation guide for supplier selection (CMU/SEI-95-TR-012).* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.
- Brodman, Judith G., and Donna L. Johnson. 1996. *The LOGOS tailored version of the CMM for small businesses, small organizations, and small projects, version 1.0.* Needham, Mass.: LOGOS International Inc. Carnegie Mellon University, Software Engineering Institute. 1995. *The capability maturity model: Guidelines for improving the software process*, edited by Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis. Reading, Mass.: Addison-Wesley.
- Curtis, Bill. 1996. The factor structure of the CMM and other latent issues. In *Proceedings of the 1996 Software Engineering Process Group Conference: Tuesday Presentations, Atlantic City, N. J.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.
- Daskalantonakis, Michael K. 1994. Achieving higher SEI levels. *IEEE Software* 11, no. 4 (July): 17-24.
- DeMarco, Tom. 1995. *Why does software cost so much?* New York: Dorset House.
- Dunaway, Donna K., and Steve M. Masters. 1996. *CMM-based appraisal for internal process improvement (CBA IPI): Method description (CMU/SEI-96-TR-007, DTIC No. ADA307934).* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.
- Ferguson, Pat, and Jeanie Kitson. 1997. CMM-based process improvement supplemented by the personal software process in a small company environment. In *Proceedings of the 1997 Software Engineering Process Group Conference, San Jose, Calif.* Pittsburgh: Software Engineering Institute: Carnegie Mellon University.

Ginsberg, Mark, and Lauren Quinn. 1995. *Process tailoring and the software capability maturity model* (CMU/SEI-94-TR-024). Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Hadden, Rita. 1998. How scalable are CMM key practices? *Crosstalk: The Journal of Defense Software Engineering* 11, no. 4 (April): 18-20, 23.

Hadden, Rita. 1998. Key practices to the CMM: Inappropriate for small projects?" In *Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, Ill.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Hayes, Will, and James W. Over. 1997. *The personal software process (PSP): An empirical study of the impact of PSP on individual engineers* (CMU/SEI-97-TR-001). Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Hoffman, Leo. 1998. Small projects and the CMM. In *Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, Ill.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Humphrey, Watts S. 1995. *A discipline for software engineering*. Reading, Mass: Addison-Wesley.

Johnson, Donna L., and Judith G. Brodman. 1997. Tailoring the CMM for small businesses, small organizations, and small projects. *IEEE Computer Society Technical Council on Software Engineering Software Process Newsletter* (winter): 1-6.

McConnell, Steve. 1996. *Rapid development: Taming wild software schedules*. Redmond, Wash.: Microsoft Press.

McFeeley, Bob. 1996. *IDEAL: A user's guide for software process improvement* (CMU/SEI-96-HB-001). Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Moody, Fred. 1995. *I sing the body electronic*. New York: Viking, Penguin Books.

Paquin, Sherry. 1998. Struggling with the CMM: Real life and small projects. In *Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, Ill.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Paulk, Mark C. 1996. Effective CMM-based process improvement. In *Proceedings of the 6th International Conference on Software Quality, Ottawa, Canada*. Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Pitterman, Bill. 1998. Key practices to the CMM: Inappropriate for small projects? In *Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, Ill.* Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Sanders, Marty. 1998. Small company action training and enabling. In *The CMM and Small Projects, Society for Software Quality Roundtable, Washington, DC*. Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Strigel, Wolfgang B. 1995. Assessment in small software companies. In *Proceedings of the 1995 Pacific Northwest Software Quality Conference*. Portland, Ore.: PNSQC Self-Publisher.

Whitney, Roselyn, Elise Nawrocki, Will Hayes, and Jane Siegel. 1994. *Instant profile: Development and trial of a method to measure software engineering maturity status* (CMU/SEI-94-TR-4). Pittsburgh: Software Engineering Institute, Carnegie Mellon University.

Williams, Louise B. 1998. SPI best practices for 'small' projects. In *The CMM and Small Projects, Society for Software Quality Roundtable*. Washington, DC: SSO.

Capability Maturity Model (CMM) and IDEAL are service marks of the Software Engineering Institute, Carnegie Mellon University.

Copyright 1999 Carnegie Mellon University.

BIOGRAPHY

Mark C. Paulk is a senior member of the technical staff at the Software Engineering Institute (SEI) in Pittsburgh. He has been with the SEI since 1987, initially working with the software capability evaluation project. Paulk was the "book boss" for version 1.0 of the Capability Maturity Model (CMM) for software and was the project leader during the development of CMM version 1.1. He is also actively involved with software engineering standards.

Prior to joining the SEI, Paulk was a senior systems analyst for System Development Corp. (later Unisys Defense Systems) at the Ballistic Missile Defense Advanced Research Center in Huntsville, Ala.

Paulk has a master's degree in computer science from Vanderbilt University and a bachelor's degree in mathematics and computer science from the University of Alabama in Huntsville.

He is a senior member of the Institute of Electrical and Electronics Engineers (IEEE), a senior member of ASQ, and an ASQ Certified Software Quality Engineer. Paulk can be reached at the Software Engineering Institute, Carnegie Mellon University, 4500 Fifth Ave., Pittsburgh, PA 15213 or e-mail at Mark.Paulk@ieee.org.