

# Using the Software CMM<sup>®</sup> With Good Judgment

**MARK C. PAULK**

*Software Engineering Institute  
Carnegie Mellon University*

*4500 Fifth Avenue*

*Pittsburgh, PA 15213*

*Telephone: +1 (412) 268-5794*

*Fax: +1 (412) 268-5758*

*Internet: mcp@sei.cmu.edu –or– Mark.Paulk@ieee.org*

## **Abstract**

The Capability Maturity Model<sup>®</sup> for Software (CMM) developed by the Software Engineering Institute (SEI) has had a major influence on software process and quality improvement around the world. 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 paper 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 other environments. 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, software capability evaluation, software process assessment, small organizations, small projects, software process improvement.

*Published in ASQ Software Quality Professional, Vol. 1, No. 3, June 1999, pp. 19-29.*

## **1. 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) with a broad charter to improve the state of the 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 have it late. We can always fix it later." The emphasis in all too many organizations is on achieving cost and schedule goals, frequently at the cost of quality. To quote DeMarco [DeMarco95], 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 our 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) that focusing on quality leads to decreases in cycle time, increases in productivity, greater customer satisfaction, and business success. Admittedly the challenge of determining "good enough" is an on-going

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 software community around the world [Paulk95]. 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.

The purpose of this paper is to discuss how the CMM can be effectively used in a wide range of environments, with a focus on small organizations but also including examples for rapid prototyping projects, maintenance shops, and R&D outfits. This paper summarizes the observations and recommendations of the author, based on over a decade of experience in CMM-based assessments and process improvement. The recommendations may appear dogmatic, but it seems unlikely that many will disagree with their intent, although there is likely to be some debate over what constitutes an "adequate" implementation in any given environment.

Level	Focus	Key Process Areas
<b>5 Optimizing</b>	<i>Continual process improvement</i>	Defect Prevention Technology Change Management Process Change Management
<b>4 Managed</b>	<i>Product and process quality</i>	Quantitative Process Management Software Quality Management
<b>3 Defined</b>	<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
<b>2 Repeatable</b>	<i>Project management processes</i>	Requirements Management Software Project Planning Software Project Tracking & Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management
<b>1 Initial</b>	<i>Competent people and heroics</i>	

**Figure 1. An overview of the Software CMM.**

Although the focus of the current release of the Software CMM, Version 1.1, is on large organizations and large projects contracting with the government, the CMM is written in a hierarchical form 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 how to interpret the model. Although the "requirements" for the CMM can be summarized in

---

© 1998 by Carnegie Mellon University.  
 This work is sponsored by the U.S. Department of Defense.  
 ® CMM and Capability Maturity Model are registered trademarks of Carnegie Mellon University.  
 SM IDEAL, Personal Software Process, PSP, Team Software Process, and TSP are service marks of Carnegie Mellon University.

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 are not prescriptive on 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 3-5, 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 10-15% of the key practices usually have to be “interpreted” – the team has to discuss at length whether an implementation is adequate as opposed to arriving at a quick consensus. Key practices are not requirements and there may be alternate implementations, but this does not 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, we believe, useful to any size organization in any application domain and for any business context. Are meeting schedules, budgets, and requirements important to small projects? To small organizations? If the employees of an organization are satisfied with the status quo, there is little that the CMM can provide that will lead to true change. Change occurs only when there is sufficient 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 [Paulk96], and this is true in all environments. Based on over 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
- research and development (R&D) organizations
- maintenance (sustaining engineering) shops
- software services organizations
- small projects and organizations

## **2. Small Organizations and Small Projects**

The emphasis of this paper on small organizations is due to a frequently asked question, “Can the Software CMM be used for small projects (or small organizations)?” Yet the definition of “small” is challengingly ambiguous, as illustrated in Table 1. At one time there was an effort to develop a tailored CMM for small projects and organizations, but the conclusion of a 1995 CMM tailoring workshop was that we could not even 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 [Ginsberg95]. In a 1998 SEPG conference panel on the CMM and small projects [Hadden98b], small was defined as “3-4 months in duration with 5 or fewer staff.” Brodman and Johnson define a small organization as fewer than 50 software developers and a small project as fewer than 20 developers [Brodman96, Johnson97].

**Table 1. Defining a “Small” Project**

<b>Variant of “Small”</b>	<b>Number of People</b>	<b>Duration</b>
Small	3-5	6 months
Very small	2-3	4 months
Tiny	1-2	2 months
Individual	1	1 week
Ridiculous!	1	1 hour

Small to tiny projects are in the range being addressed by Humphrey in his Team Software Process<sup>SM</sup> (TSP) work, and the individual effort is directly addressed by the Personal Software Process<sup>SM</sup> (PSP) [Humphrey95]. 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 effectively used in organizations with less than 15 employees and for projects with as few as two people. The SEI’s maturity profile as of December 1998 shows 27 organizations with fewer than 25 employees that have performed CBA IPI assessments.

Small organizations, just like large ones, will have problems with undocumented requirements, the mistakes of inexperienced managers, resource allocation, training, peer reviews, and documenting the product. The challenge of providing resources for process improvement – both to formally identify problems and systematically address them – will frequently 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 really needed by small projects and organizations. To answer this question, we need to consider what discipline involves – and that leads to the heart of this paper’s CMM interpretation discussion.

### **3. Improvement Issues from the IDEAL Perspective**

The CMM is a tool that should be used in the context of a systematic approach to software process improvement, such as the SEI’s IDEAL model [McFeeley96], which depicts the activities of an improvement program in five phases:

- I** Initiating (the improvement program)
- D** Diagnosing (the current state of practice)
- E** Establishing (the plans for the improvement program)
- A** Acting (on the plans and recommended improvements)
- L** Learning (the lessons learned and the business results of the improvement effort)

Obtaining senior management sponsorship in the *Initiating* phase is a crucial component of building organizational capability. As individuals, we can exercise professionalism and discipline within our sphere of control, but if an organization as a whole is to change its performance, then 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 you have 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 are 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 [Barbour96]. There are circumstances, however, under which SCEs may still occur, such as field training an evaluation team.

The relationship to any existing TQM programs should be identified in the Initiating phase. Software process improvement efforts should be aligned with 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 advisable to use a streamlined assessment process. A full-blown CMM-based appraisal for internal process improvement (CBA IPI) [Dunaway96], with its criteria for validity, accuracy, corroboration, consistency, and sufficiency, can last two weeks, which is probably excessive for a small organization [Strigel95, Paquin98, Williams98]. The emphasis should be on efficiently identifying important problems, even if some are missed due to lack of rigor. The CBA IPI assessment method can be tailored down, however, and there are a number of other assessment methods, such as Interim Profile, that may be more appropriate [Whitney94, Daskalantonakis94].

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 example, one person may be the project manager, project software manager, SCM manager, etc. Explicitly stating this makes interpretation of the CMM much 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 that it is defining its project-level processes since there is much less "cultural inertia" to overcome, even when significant cultural change occurs. The most effective organizational learning strategy is likely to be one stressing organizational assets that lessen the overhead of projects. 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 they do not specify a particular implementation. For example, the estimating practices in Software Project Planning have subpractices on using historical data, but they do not specify a cost model, such as COCOMO, Price-S, or Slim, or even state that a cost model should be used.

For a small organization, the loop to the *Learning* phase may be much tighter than for large organizations. A one-year improvement cycle may be realistic, where large organizations will more typically take 2-3 years between assessments.<sup>1</sup> This reinforces the need for a low-overhead assessment process.

---

<sup>1</sup> Note that results for specific improvement actions should be observed well before the organizational improvement cycle delineated by an assessment rolls around.

## 4. 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 2.

<b>Maturity levels</b>	(5 levels)
→ <b>Key process areas</b>	(18 KPAs)
→ <b>Goals</b>	(52 goals)
→ <i>Key practices</i>	(316 key practices)
® <i>Subpractices and examples</i>	(many)

**Figure 2. The CMM structural hierarchy.**

The “normative” components of the CMM are maturity levels, key process areas, and goals. All practices in the CMM are informative as opposed to 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 small organizations – but they do provide insight into how to achieve the goals and implement repeatable, defined, measured, and continually improving software processes. This emphasizes the need for reasonable and professional practices and helps prevent such “processes” as the estimating procedure that was simply “Go ask George.”

In general, key process areas and goals are always relevant to any environment, with the exception of *Software Subcontract Management*, which may be “not 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 competent professional judgment, implying a need for trained, experienced assessors and process definers, even for small organizations.

### 4.1 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 almost certainly yes. If the organization is doing commercial shrinkwrap development, this issue is more debatable. Will bad management lead to serious business consequences? For many successful shrinkwrap software companies, no new product in company history has ever been finished on time or provided the functionality originally envisaged [Moody95]. It is unlikely, however, that bad management has contributed to the success of the thriving shrinkwrap companies, and it has certainly contributed to the failure of many that have not survived.

**Requirements Management.** Documented customer (system) requirements and communication with 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, i.e., too many prototypes capturing different potential requirements or before transitioning 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 #1 factor in successful process definition and improvement is “planfulness” [Curtis96]. Planning is needed for every major software process, but within the bounds of reasonable judgment, 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 some identified priority 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 a good practice to document internal commitments, especially those that cross organizational boundaries.

**Software Project Tracking & Oversight.** Knowing what you have accomplished versus what you have committed to is always important. Admittedly, managers have to be careful not to over-control – 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 (e.g., we’re 87% 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 2-3 weeks for a work package and 2-3 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, you need to understand what the data means and how to analyze it, which implies collecting a simple set of useful data. The effort of collecting the data needs to be minimized so the small organization is not overwhelmed by overhead. This is also a problem for large organizations, but the margin for error is razor-thin for small organizations.

**Software Subcontract Management.** For small organizations, rapid prototyping projects, maintenance shops, and R&D outfits, Software Subcontract Management will usually be not applicable.

**Software Quality Assurance.** A small project is unlikely to have an independent 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 QA function in your process via checklists, tools, etc., even if an independent SQA group exists.

**Software Configuration Management.** A small project is unlikely to need an SCM group or a 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 [McConnell96].

**Closing Level 2 Thoughts.** Many of the context-sensitive, large-project implementation issues relate to organizational structure. Similar to the SCM and SQA examples above, 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. If one reads the CMM definition of “group,” it 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.

## 4.2 Getting to Level 3

**Organization Process Focus.** At Level 3, the CMM emphasis is on consistency and organizational learning. In most organizations, a software engineering process group (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 explicitly assigned and monitored. Thinking about the way you work and how to do better is always important, regardless of the models or standards that you may use to structure your 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

- 1) communicate – to others now and perhaps to yourself later;
- 2) understand – if you can't write it down, you don't really understand it; and
- 3) 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 – they put repeatable processes in place. 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 1-2 page process description may suffice, and subprocesses and procedures can be invoked as needed and useful. Use good software design principles, such as locality, information hiding, and abstraction, in defining processes.

The degree of formality needed for processes is a frequent challenge for both large and small organizations [Comer98, Hadden98a, Pitterman98, Sanders98]. 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 “Documentation and the CMM” of **The Capability Maturity Model: Guidelines for Improving the Software Process** [Paulk95], 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 [Abbott97, Williams98]. As personnel change, the incremental need for training may not be adequately addressed. Mentoring and apprentice programs may suffice to address this issue, but they cannot be assumed to be satisfactory without careful monitoring.

**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 [Ginsberg95, Ade96, Ahlgren96]. Although standard processes provide a foundation, most projects will also have unique needs. Unreasonable constraints on tailoring can lead to significant resistance to following the process. As Hoffman expresses it, “Don't require processes that don't make sense” [Hoffman98]. Use organizational assets, but use them intelligently.

Some argue that software project management is really risk management. This implies that you should use an incremental or evolutionary life cycle. If you want to focus on risk management, the spiral model may be the preferred life cycle model. If you want to focus on involving the customer, 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 life cycle. 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. 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, communication over time and perhaps with those who succeed you is important.

**Peer Reviews.** Although you can argue over the best kind of peer review, the simple fact is that the benefits of peer reviews far outweigh their 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 that we do them 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.

### 4.3 Getting to Levels 4 and 5

Any organization embarking on Levels 4 and 5 needs little guidance in CMM interpretation principles. Small organizations should seriously consider the Personal Software Process<sup>SM</sup> (PSP<sup>SM</sup>) and Team Software Process<sup>SM</sup> (TSP<sup>SM</sup>) to bootstrap their process improvement efforts to Level 5 [Ferguson97, Hayes97]. Where the CMM addresses the organizational side of process improvement, PSP addresses building the capability of individual practitioners. The PSP course convinces the individual professional, based on his or her own data, of the value of a disciplined, engineering approach to building software. PSP and TSP take the individual and the team to a Level 5 process capability that the organization can leverage.

### 4.4 Miscellaneous Improvement Issues

Paquin identifies five issues for small organizations and projects [Paquin98]:

- 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 the slots. 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. Expressing the questionnaire in the terminology of the organization is therefore a desirable precursor to even an informal assessment or

survey. Assessments should be “light weight,” and documentation should focus on minimum essential information.

Abbott identifies six keys to software process improvement in small organizations [Abbott97]:

- 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 frequently an issue for large organizations than small, so it is interesting that Abbott points this out for his small company. The advise to use process improvement consultants can be problematic. The experience and expertise of a good consultant is unquestionably of great value, yet skilled consultants charge a premium price – frequently beyond what a small organization can afford. Unskilled consultants can be actively detrimental. Although none of the process guidance in the CMM or similar models and standards is particularly “rocket science,” the pitfalls in changing individual, team, and organizational behaviors are non-trivial. There are no easy answers to this issue.

Brodman and Johnson identify seven small organization/small project challenges [Johnson97]:

- handling requirements
- generating documentation
- managing projects
- allocating resources
- measuring progress
- conducting reviews
- providing training

They have also developed a tailored version of the CMM for small businesses, organizations, and projects [Brodman96]. Although the majority of the key practices (but very 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.

## 5. 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 2 or 3 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

- concisely 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 (and be) simplistic, but this is the core of a disciplined process that with a philosophy of continual process improvement can lead even a small software organization to Level 5.

Many of the abuses of the Software CMM spring from a fear of what “others” may do, such as evaluate simple or alternative implementations adversely, thus leading to loss of a contract. If an organization applies common sense to the guidance in the CMM as guidance rather than requirements, then many of the interpretation problems for the model vanish. Although the CMM provides a significant amount of 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<sup>2</sup>, it is for making consistent and comprehensive judgments; it is not implying a requirement that each key practice be literally implemented. Even weaknesses against a key practice that result in findings do not necessarily result in failing 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. On more than one occasion I have heard someone say they were doing something that was foolish, but they were afraid that the customer was so ignorant or incompetent that they would be unable to understand the rationale for doing things differently than literally 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 we recommend that process maturity be included in risk assessment during source selection, rather than using maturity levels to filter offerors [Barbour96].

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. That is why we emphasize the need to tie improvement to business objectives.

The challenges in interpreting the CMM appropriately for different environments differ in degree rather than in 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 Bellcore: “Let common sense prevail!” The CMM is a proven tool to support process appraisal and software process improvement in a wide range of environments<sup>3</sup>, but it must be used with professional judgment and common sense to be truly effective.

## References

- Abbott97      John J. Abbott, “Software Process Improvement in a Small Commercial Software Company,” , **Proceedings of the 1997 Software Engineering Process Group Conference**, San Jose, CA, 17-20 March 1997.

---

<sup>2</sup> The CBA IPI method requires that each Activity Performed be investigated and the key practices in the other common features be sampled. I recommend investigating each key practice rather than just sampling the practices in the institutionalization common features.

<sup>3</sup> There are far too many case studies and analyses of CMM-based improvement to cite here, but those interested may wish to examine <http://www.sei.cmu.edu/cmm/cmm.articles.html#biblio.case.studies>.

- Ade96 Randy W. Ade and Joyce P. Bailey, "CMM Lite: SEPG Tailoring Guidance for Applying the Capability Maturity Model for Software to Small Projects," **Proceedings of the 1996 Software Engineering Process Group Conference: Wednesday Papers**, Atlantic City, NJ, 20-23 May 1996.
- Ahlgren96 Magnus Ahlgren, "CMM Light for SMEs," **Conference Notebook: The First Annual European Software Engineering Process Group Conference**, Amsterdam, The Netherlands, 26-27 June 1996, section C413.
- Barbour96 Rick Barbour, "Software Capability Evaluation Version 3.0 Implementation Guide for Supplier Selection," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-95-TR-012, April 1996.
- Brodman96 Judith G. Brodman and Donna L. Johnson, **The LOGOS Tailored Version of the CMM for Small Businesses, Small Organizations, and Small Projects**, Version 1.0, August 1996.
- Curtis96 Bill Curtis, "The Factor Structure of the CMM and Other Latent Issues," **Proceedings of the 1996 Software Engineering Process Group Conference: Tuesday Presentations**, Atlantic City, NJ, 20-23 May 1996.
- Daskalantonakis94 Michael K. Daskalantonakis, "Achieving Higher SEI Levels," IEEE Software, Vol. 11, No. 4, July 1994, pp. 17-24.
- DeMarco95 Tom DeMarco, **Why Does Software Cost So Much?**, ISBN 0-932633-34-X, Dorset House, New York, NY, 1995.
- Dunaway96 Donna K. Dunaway and Steve M. Masters, "CMM-Based Appraisal for Internal Process Improvement (CBA-IPD): Method Description," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-96-TR-007, DTIC Number ADA307934, 1996.
- Ferguson97 Pat Ferguson and Jeanie Kitson, "CMM-Based Process Improvement Supplemented by the Personal Software Process in a Small Company Environment," **Proceedings of the 1997 Software Engineering Process Group Conference**, San Jose, CA, 17-20 March 1997.
- Ginsberg95 Mark Ginsberg and Lauren Quinn, "Process Tailoring and the Software Capability Maturity Model," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-94-TR-024, November 1995.
- Hadden98a Rita Hadden, "How Scalable are CMM Key Practices?" Crosstalk: The Journal of Defense Software Engineering, Vol. 11, No. 4, April 1998, pp. 18-20, 23.
- Hadden98b Rita Hadden, "Key Practices to the CMM: Inappropriate for Small Projects?" panel, Rita Hadden moderator, **Proceedings of the 1998 Software Engineering Process Group Conference**, Chicago, IL, 9-12 March 1998.
- Hayes97 Will Hayes and James W. Over, "The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-97-TR-001, December 1997.
- Hoffman98 Leo Hoffman, "Small Projects and the CMM," in "Key Practices to the CMM: Inappropriate for Small Projects?" panel, Rita Hadden moderator, **Proceedings of the 1998 Software Engineering Process Group Conference**, Chicago, IL, 9-12 March 1998.
- Humphrey95 Watts S. Humphrey, **A Discipline for Software Engineering**, ISBN 0-201-54610-8, Addison-Wesley Publishing Company, Reading, MA, 1995.

- Johnson97 Donna L. Johnson and Judith G. Brodman, "Tailoring the CMM for Small Businesses, Small Organizations, and Small Projects," Software Process Newsletter, IEEE Computer Society Technical Council on Software Engineering, No. 8, Winter 1997, p. 1-6.
- McConnell96 Steve McConnell, **Rapid Development: Taming Wild Software Schedules**, Microsoft Press, Redmond, WA, 1996.
- McFeeley96 Bob McFeeley, "IDEAL: A User's Guide for Software Process Improvement," Software Engineering Institute, Carnegie Mellon University, CMU/SEI-96-HB-001, February 1996.
- Moody95 Fred Moody, **I Sing the Body Electronic**, Viking, Penguin Books, New York, NY, 1995.
- Paquin98 Sherry Paquin, "Struggling with the CMM: Real Life and Small Projects," in "Key Practices to the CMM: Inappropriate for Small Projects?" panel, Rita Hadden moderator, **Proceedings of the 1998 Software Engineering Process Group Conference**, Chicago, IL, 9-12 March 1998.
- Paulk95 Carnegie Mellon University, Software Engineering Institute (Principal Contributors and Editors: Mark C. Paulk, Charles V. Weber, Bill Curtis, and Mary Beth Chrissis), **The Capability Maturity Model: Guidelines for Improving the Software Process**, ISBN 0-201-54664-7, Addison-Wesley Publishing Company, Reading, MA, 1995.
- Paulk96 Mark C. Paulk, "Effective CMM-Based Process Improvement," **Proceedings of the 6th International Conference on Software Quality**, Ottawa, Canada, 28-31 October 1996, pp. 226-237.
- Pitterman98 Bill Pitterman, "Key Practices to the CMM: Inappropriate for Small Projects?" panel, Rita Hadden moderator, **Proceedings of the 1998 Software Engineering Process Group Conference**, Chicago, IL, 9-12 March 1998.
- Sanders98 Marty Sanders, "Small Company Action Training and Enabling," in **The CMM and Small Projects**, Society for Software Quality Roundtable, Washington, DC, 26 January 1998.
- Strigel95 Wolfgang B. Strigel, "Assessment in Small Software Companies," **Proceedings of the 1995 Pacific Northwest Software Quality Conference**, 1995, pp. 45-56.
- Whitney94 Roselyn Whitney, Elise Nawrocki, Will Hayes, and Jane Siegel, "Instant Profile: Development and Trial of a Method to Measure Software Engineering Maturity Status," CMU/SEI-94-TR-4, Software Engineering Institute, Carnegie Mellon University, March 1994.
- Williams98 Louise B. Williams, "SPI Best Practices for 'Small' Projects," in **The CMM and Small Projects**, Society for Software Quality Roundtable, Washington, DC, 26 January 1998.

**Acknowledgements.** I would like to thank the folks who reviewed this paper: 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.