A Mature View of the CMM

Bill Curtis
TeraQuest Metrics, Inc., Austin, Texas
Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA

James Bach's philippic against the Capability Maturity Model for Software (Paulk, Curtis, et al., 1993; Paulk, Weber, et al., 1993; Paulk, et al., in press) provides an opportunity to respond to several misconceptions about the CMM and its application. Some of these misconceptions I shared when I read Watts Humphrey's (1988) first paper on the maturity framework. Many of these misconceptions were not in what Humphrey had written, but in how I interpreted it. As I reflected on what characterized the well run projects and caused the disasters I had experienced, the framework made more sense. I have never thought that the CMM was perfect, but having worked with and for many software development organizations for 17 years, I understand why industry and government need such a model to help guide their efforts to improve.

Misconceptions are easily formed if one has not studied the key practices of the CMM (as Bach admits he hasn't). The horrible fates Bach ascribes to process improvements guided by the CMM are not supported by organizations that have actually made these improvements. Many of Bach's arguments show little insight into how key practices of the CMM affect the culture and professional behavior of people in maturing organizations.

Bach seriously misrepresents Humphrey on many issues, and in so doing indicates he either has not read Humphrey's (1989) book, or at least has not understood its prose. The most egregious example is when he states that, "Humphrey and the CMM...decry [people] as unreliable and assume that defined processes somehow render individual excellence unimportant." Quite to the contrary Humphrey states, "Attracting the best people is vital, but it is also essential to support them with an effectively managed software process" (1989, p. ix). Bach interprets the CMM as replacing individual talent. The contributors to the CMM designed it to augment individual talent and foster learning throughout the organization.

What problems does the CMM address? The CMM was designed to help guide improvements in software organizations who continually miss schedules, overrun budgets, and deliver defective software. It was designed to help organizations get their software development operations under sufficient control so that the staff does not have to work late into the night and on weekends trying to avoid corporate embarrassment with customers or the trade press. Finally, it was designed to help software organizations consistently achieve quantitative business targets for quality, development time, cost, etc. that represent critical competitive parameters in their niche of the software market.

Let us examine some of the popular misconceptions.

Misconception #1: The CMM was developed for use in the Department of Defense, in the aerospace industry, or in some other setting that is not relevant to my organization.

The CMM was born when Watts Humphrey began trying to integrate what he had learned about differences between successful and unsuccessful software projects inside IBM with what he saw working successfully when total quality management practices were applied in other industries. Not satisfied that he had a model generic to non-IBM organizations, Humphrey started a vigorous program of process assessments with other companies after he came to the Software Engineering Institute in 1986. Through the hundreds of assessments performed since, it is clear that the CMM addresses problems that are endemic in software development and maintenance.

The CMM was developed with extensive public input and review. The practices comprising it have been gathered from archival records of software engineering successes, from best practices observed in software process assessments, from successful total quality management programs in non-software domains, and from public workshops. Beginning in 1987 the SEI's Software Process Program began holding workshops in which industry and government participants identified effective practices in many areas of software engineering. The last CMM Workshop held in McLean, VA in April 1992 drew over 200 participants who spent two days in group sessions critiquing and reworking each of the 18 key process areas in the CMM.

Prior to releasing a new version of the CMM, it undergoes extensive review by a correspondence group of over 500 software professionals. Some companies prepared a corporate response, and organized their internal review of the CMM to ensure that each key process area was reviewed by the company's experts in that area. A CMM Advisory Board of 14 senior professionals approves all of the revisions to new versions of the CMM before they are released. If the CMM is a “mythology”, it is not one formulated by a fanatical sect of theorists cloistered away in Pittsburgh.

The CMM was not originally devised to evaluate bidders on government contracts. Some of the most successful applications of CMM-based improvement programs have been in commercial companies such as Schlumberger (Wohlwend & Rosenbaum, 1993). However, when the Air Force's Electronic Systems Command asked the SEI to develop a method for evaluating the development capability of contractors, the maturity framework found a new application beyond guiding improvement programs. Companies such as Bellcore are now appraising the capability of software suppliers using software capability evaluations based on models similar to the CMM.

Misconception #2: The CMM has no theoretical basis and it institutionalizes the mindless regimentation of processes.

Some argue that there is no theoretical model underlying the application of total quality management or statistical quality control techniques. Nevertheless, a set of principles has emerged over the last several decades that when applied conscientiously seem to improve an organization's product development capability. These principles appear in the writings of W. Edwards Deming, Joseph Juran, Philip Crosby, and others. The CMM translates these principles into a model applied to software engineering. This model embodies a theory about how to improve the performance capability of an organization through the leverage available in its development process. Although an entertaining academic debate can be engaged over whether

this model qualifies as a theory, it has proven capable of improving the performance of software organizations (Herbsleb & Zubrow, 1994).

Primary among these principles is that a process cannot be improved if it cannot be repeated. All professional instructors of golfers, tennis players, or baseball batters know that they cannot help their pupils improve until they can repeat a swing, however inelegant. Most duffers' original problem is that they rarely execute the same swing twice in a row—first swatting, then pushing, then slapping, and all the while spraying balls in every direction. Once the instructor gets the beginner to repeat a reasonable swing, then they can work on systematically improving it.

Similarly, the Repeatable level of the CMM (level 2) is primarily focused on helping software organizations remove the impediments that keep them from repeating successful software development or maintenance practices. The most common impediments are schedule or resource commitments that the software staff could not meet regardless of how sophisticated their skills or process. Another particularly wicked impediment is uncontrolled requirements changes that devastate the original planning. In a rush to satisfy unreasonable objectives, the project staff begin making mistakes that are not caught until it is much more expensive to remove them.

When sound practices are sacrificed to schedule or other constraints, engineers have little chance to improve their performance or to follow through effectively on innovative ideas. Officials from IBM Federal Systems Company (recently sold to Loral) admitted in the business press that their problems on the FAA's Advanced Automation Project resulted from circumventing many of their standard development processes in trying to meet the customer's unrealistic schedule. Unfortunately, this scenario has been played out repeatedly in every segment of the software industry.

Having established an ability to make and protect achievable commitments, the organization can focus on transferring its best development or maintenance practices across the organization. Capitalizing on processes that work best is the heart of the Defined level (level 3). The organization identifies the design, testing, inspection, management, configuration control, and other processes that seem to have worked best on different projects, and integrates them into an organization-wide process model. In using these process assets, managers and technical staff benefit from lessons learned on earlier projects and do not have to reinvent successful methods. The process specialist can play the role in software organizations that industrial engineers play in manufacturing.

This reuse of successful processes is often interpreted as the mindless regimentation of talented professionals. Far from being restrained in process straight jackets, software projects are encouraged to select the most appropriate life cycle for their application (prototyping, spiral, waterfall, etc.) and then tailor the organization's standard processes to work within that context. Although a sound development process helps instill discipline, it does not follow that it subverts talent. Humphrey was clear on this point.

> *"discipline controls the environment and methods to specific standards, while regimentation defines the actual conduct of the work. Discipline is required in large software projects to ensure, for example, that the many people involved use the same*

conventions, don't damage each others' products, and properly synchronize their work. Discipline thus enables creativity by freeing the most talented software professionals from...many crises" (1989, p.13)

Many organizations have set challenging performance goals such as a 10X reduction in project schedules or 100X reductions in delivered defects. If a development process cannot be repeated, the organization has no way of ensuring that it can sustain improvements. Once the organization can execute its software processes consistently, it can use its process data to systematically eliminate the causes of wide variations in its performance. The objective of the Managed level (level 4) is to set quantitative performance and quality targets and reduce the variation in process and performance to stabilize the organization's capability toward achieving these targets.

During this attempt to reduce performance variation statistical process control principles can be applied. However, their application, and even the relevant statistical methods, may differ from those used in manufacturing. We are just beginning to learn about the statistics relevant to intellectual artifacts. Nevertheless, the experience of IBM's (now Loral's) Onboard Space Shuttle project (Billings et al., 1994) indicates that these principles can be applied effectively to help organizations achieve sustained performance against challenging criteria.

As even more challenging performance objectives are levied, the software organization begins to identify technology and process innovations that can improve its competitive posture. Thus, at the Optimizing level (level 5) the organization seeks to continually improve its performance. This is not process for process's sake. It is process in service of the organization's competitive performance. The quantitative targets set for process performance must be related to the needs of the customer and the competitive requirements of the business.

The basis for the CMM, theoretical or otherwise, is in trying to help organizations steadily improve their capability. The capability of an organization to develop software is the range of results it ordinarily experiences when executing projects. Capability is improved by establishing a learning environment where the organization has interpretable quantitative feedback on its performance. In the abstract the CMM seeks to establish a framework in which:

- processes can be repeated,
- consistency in performing the most effective processes supports the transfer of learning across projects,
- wide variations in performance are reduced, and
- average performance is continuously improved.

Misconception # 3: The CMM ignores people.

The CMM was designed to help improve the processes through which people coordinate their software development efforts. It was not designed to focus on individual performance, since its point of application is where at least several talented people must interweave their skills to produce a product that seems born of a single grand design. With the exception of a key process area for organization-wide training in core technical and process competencies, the CMM focuses mainly on issues of coordination among professionals. Coordination breakdowns continue to undo many projects staffed with talented people.

Organizations achieving maturity growth have reported characteristic cultural changes at each level. One characteristic of organizations at the Initial level (level 1) is that commitments are not taken seriously. Even though the staff will work hard to achieve unreasonable objectives, the targets have little credibility because the technical staff often did not participate in developing them. Managers in these environments are too often looking to assign blame for missed milestones and defective programs.

At the Repeatable level, a culture of commitment is established by involving those who must perform the work in the process of estimating and planning it. Since most low maturity organizations lack accurate historical data, the experience of the technical staff is the only source of reliable information on what a new project will require. This first step toward a participatory culture begins to empower technical professionals. Both management and the technical staff take commitments more seriously, since they had a hand in developing them. When commitments are missed, blame is placed on the process, not on the people. Lack of experience in similar applications is considered an organizational limitation, not a personal shortcoming.

Hughes Aircraft's Ground Systems Division reported that when they reached the Defined level, a common engineering culture emerged (Humphrey, Snyder, & Willis, 1991). Rather than a collection of institutionalized rules, the organization-wide software process established a set of norms which engineers expected to be observed in developing software. This culture of professionalism became a source of pride that accelerated the improvement program. When working from a defined process that has been tailored for use on a project, engineers can quickly develop expectations about their roles and responsibilities. Far from regimenting creative talents, a common process enables engineers to work effectively in a complex environment with intricate dependencies.

Software professionals are highly motivated to achieve excellence. At the Managed level they are provided with detailed process and product data that give continuous feedback on their individual and collective performance. Such data allow them to evaluate their performance and identify opportunities for improvement. Yes, such data can be abused by management if used improperly. However, the culture typically established in organizations achieving this stage trusts engineers to use data to control project performance. At this level engineers are empowered to manage their own processes.

As the organization moves to the Optimizing level they are empowered to change their processes. Since the culture of the Onboard Shuttle project has so internalized the organization's objectives and the processes through which they are achieved, many responsibilities usually retained by management have been assumed by committees run by the technical staff (Billings et al., 1994). Managers on this project spend much of their time ensuring that there are people with adequate technical skills to back up every critical position on the project. Motorola reports that their software operation in Bangalore, India recently assessed at the Optimizing level has become the preferred software employer in the area.

Far from stifling individual creativity, disciplined environments are much more effective at spreading and retaining knowledge than are chaotic, undisciplined environments. Learning requires feedback. The kind of environments the CMM tries to help create are rich in feedback. A hallmark of the CMM is to use this feedback for constant improvement of the process, the

product, and the professional staff. This is very much the environment discussed by Senge (1990) and others as the basis for a learning organization.

Even so, we recognize that the CMM does not contain many practices that are required to attract, develop, motivate, and retain top software talent. Accordingly, I proposed a CMM-based model for human resource development several years ago (Curtis, 1990). This has evolved into a People Management - Capability Maturity Model (PM-CMM) that is now a funded project at the SEI. It is being developed as a companion to the CMM for systematically increasing an organization's human talent. This model is being developed separately to avoid overloading the CMM and reducing its clarity in process improvement. It contains practices in areas such as recruiting, selection, performance management, training, career development, compensation and recognition, team development, work environment and culture, and organizational structure and work design.

The SEI intends to develop the PM-CMM through the same process of public participation and review that was used in developing the CMM. Accordingly, the SEI is hosting a public workshop to critique a preliminary version of the PM-CMM and contribute best practices on December 14-15, 1994 in McLean, VA. Anyone interested in attending can obtain more information by contacting SEI Customer Service (1-412-268-5800).

In contrast to Bach's accusation that he decries humans as unreliable, Humphrey has taken up the cause of helping software professionals develop their individual skills. Over the past several years he has been developing and evaluating a Personal Software Process (Humphrey, in press) whereby individuals develop a disciplined personal approach to writing programs (the specific processes differ by individual). Using this disciplined approach they monitor their developing skill and identify areas needing attention. Further, by developing a disciplined approach from the very beginning of their training, software engineers are better prepared for the process discipline required to build industrial strength systems. Humphrey's Personal Software Process is now being taught in introductory programming courses at several universities, and is being piloted in several companies.

Misconception # 4: The CMM puts level growth above performance.

Despite admonitions from the SEI, many organizations base their improvement programs on obtaining the next higher level rather than on systematically raising the organization's process capability. During the early 1990s I heard executives from respected companies say repeatedly, “Level 5 by 1995!” Now that the midpoint of the decade is near, these exhortations have been revised to achieving levels 2 or 3.

This slogan-based approach to process improvement has several problems. First, it is a slogan (Deming says eliminate slogans) and not a plan. Especially in low maturity organizations the process improvement team is trying to get the organization to adopt a management discipline based on planning and tracking. To be credible in championing these changes, they must plan their own activities and treat process improvement as a project deserving of project management discipline.

Second, level-based programs are not designed to solve development problems and boost capability, but to answer 'yes' to specific questions. This is the first step toward mindless regimentation. There is a sense that low maturity is a disease and once you have answered 'yes' to enough questions you are cured and you can turn to other things.

Third, both the improvement team and the organization are under tremendous pressure during the follow-up assessment. All have vested interests not in identifying real problems that need to be addressed, but in being rated at the next level. Under these conditions the assessment process loses its benefit in identifying the next set of process issues that must be addressed. The assessment process reverts to an audit and the answers are canned.

The key process area structure of the CMM is designed to allow organizations to demonstrate significant progress in improving their processes without having to achieve the next level. It is more important to ensure that the organization plans and estimates systematic process improvements. Periodically, but not on 2 year cycles, the organization will have satisfied all of the goals of a particular maturity level. Similarly, the SEI trains capability evaluators not to score maturity levels, but rather to use the profile of process strengths and weaknesses to compare the risks of awarding business to each bidder.

Misconception # 5: The CMM does not allow you to skip levels.

The CMM is designed to lay a series of foundations on which the improvements at the next maturity level can be built. There is no rule in the CMM that forbids organizations from installing improvements that come from a level higher than the one it is working to attain. The problem is that without the proper foundation in place, improvements from two or more levels higher than the organization's current capability are at risk of being discarded or used ineffectively.

For instance, although peer reviews are extremely powerful in improving performance, they are one of the first processes discarded when a project is assigned an absurdly short schedule. Once the organization has installed sufficient management discipline to make and manage reasonable commitments, it can protect peer reviews. Most experienced process improvement consultants like to train the organization in one of the many peer review techniques as soon as they believe the organization can protect the review process. They worry less about whether the organization has attained the Defined level then about whether reviews will be performed even in a crisis. Likewise, a database full of detailed process data is uninterpretable if there is no consistency in the processes that produced them.

The CMM was never intended to restrict organizations from making process improvements that make sense in their current situation. The CMM is a set of guidelines that help organizations prioritize their improvement activities. The priorities should match the situation. For instance, we normally recommend that an organization establish a Software Engineering Process Group shortly after initiating an improvement program. Establishing a process group to help at the organizational level is not listed as a practice until the Defined level. Managers can install the basic management disciplines on their individual projects without being facilitated by a process group. Even so, improvements will probably be more efficient and uniform if one exists.

Misconception # 6: The success of PC software companies invalidates the CMM.

One of the most frequent criticisms of the CMM is that the large Personal Computer software developers are probably all at Level 1 and still make gobs of money on software. If they are so successful, why does growth in process capability make a difference?

Let us examine by what measure PC software developers are successful? If you look at issues of *PC Magazine* from 1983-1984, you quickly realize that most of the PC software companies that have ever existed are no longer with us. This niche of the software market has seen a staggering number of failures. The venture capital that once so freely flowed into software startups has now mostly evaporated. The heady days of freewheeling entrepreneurs have given way to brutal competition among the few survivors.

Can any industry consider itself truly successful when it refuses to accept responsibility for its products? In what other industry do people have so little confidence in the quality of their products that one large company expects you to "agree that the use of the...software is at your sole risk". Another large company only warrants that "the software will work substantially in accordance with the accompanying written materials for a period of 90 days". After 90 days what decays first, the software or the written materials? Most small PC software companies only warrant that the disk is free of defects—and they didn't manufacture the disk.

The PC software industry is notorious for slipping delivery dates and shipping defects. In this regard they share the same problems of low maturity organizations in many other areas of software development. Perhaps Mr. Bach and I talk to different people in PC software companies. I hear from many who realize they have serious process problems that are costing them millions and threatening their competitiveness. Their salvation is that few, if any, of their competitors can produce software any more quickly or reliably than they can. If they fall seriously behind on delivering a hotly competed application, they will probably lose the product line. When I was in the Software Technology Program at MCC in the late 1980s, a senior executive from one PC software company visited us and queried, "I have several million lines of unstructured assembler, can you help me?"

In the past, defects in PC software were tolerated because the impact was typically only felt by individuals. Few companies account for the cost of defective software at the individual level. However, as more companies begin to entrust critical applications to PCs, the tolerance for poor quality will decrease. As PC software companies try to expand into enterprise-wide utilities and applications, they will collide with new competitors, some of whom may raise the bar for quality and schedule performance.

In the PC world, time to market is the crucial issue because it translates into market share—the key to survival. In immature environments time to market is heavily affected by setting unachievable schedules or seriously underestimating the size of the product. These problems result in hastily drawn designs, failure to detect defects when they are least time consuming to fix, and breakdowns in coordination. Between 30% and 45% of the time on most software projects is spent reworking faults detected late in the development cycle. These are process problems. Fixing them will increase the competitive posture of the business.

Heroic efforts occasionally pull organizations through difficult projects. The problem with relying on heroes is that the only way to ensure the same performance on the next project is to assign the same hero. The data are clear—there are not enough heroes to go around. Even worse, many of the heroes in PC software companies have passed 30 and want a life.

To predict performance in low maturity organizations, you need to know who will be assigned to the project. Such organizations do not possess their development capability, because they have not developed an environment that transfers skills efficiently. Chaotic environments severely restrict the time and opportunities for talented people to transfer their knowledge to other members of the team. A more disciplined environment benefits less experienced members of the technical staff as much as it benefits the organization.

The CMM was not designed for application in advanced research laboratories or in the garages of software startups. It was designed for application in established software organizations with customers who expect reliable products on the announced date. The CMM was not designed to turn an unimaginative company into a font of innovation. Yet, it does not cinch up programmers in a process harness that restrains their creative potential. Failure to innovate is usually less a problem of stultifying process than of projects in crisis and organizations run by conservative, technically outdated, stultifying management. Rather, the CMM seeks to remove some of the process-based impediments to exploring innovative designs. The bottom line is that when you don't have time to think, you don't innovate.

Misconception # 7: There is little empirical support for the CMM.

There is a growing body of case study evidence, the kind pioneered by the Harvard Business School in training executives, that CMM-based software process improvement works. Herbsleb and Zubrow discuss much of this evidence in a companion article in this issue. The Software Engineering Process Group Annual Meeting consists of reports from organizations in many segments of the software market on which techniques are working in process improvement. The 1994 meeting drew over 900 people to Dallas, and the 1995 meeting will be in Boston next spring. Information can be obtained from the SEI.

We would like to conduct a tightly controlled scientific experiment validating the CMM. However, we have found few volunteers for the control group. Few shareholders would be happy knowing their organization had been assigned placebo improvements. Further, the kind of interventions performed in a properly run improvement program cannot be simulated in controlled laboratory experiments because of scale issues. Accordingly, the SEI welcomes experience reports, positive or negative, from any organization willing to contribute them to an anonymous database.

It is no secret that many organizations have failed to achieve lasting gains after conducting a software process assessment. This is less a flaw in the CMM than in the execution of their improvement programs. In many cases these organizations either lacked executive support and never received adequate resources for making improvements. In some cases the improvement project was executed poorly, typically failing to produce action plans for the

intended improvements. To be effective, improvement programs must be resourced and managed like a project.

It is encouraging to read the process improvements James Bach is making at Borland. Having someone responsible for developing the organization's process assets, actually defining sets of processes that can be used to improve project performance, and having project teams tailor these processes to their specific needs are certainly consistent with several of the key process areas at the Defined level of the CMM. With improvements such as these perhaps Borland will avoid the fate that befell Ashton-Tate.

References

Billings, C. Clifton, J., Kolkhorst, B., Lee, E. & Wingert, W.B. (1994). Journey to a mature software process. *IBM Systems Journal*, 33 (1), 46-61.

Curtis, B. (1990). Managing the real leverage in software productivity and quality. *American Programmer*, 3 (8), 5-15.

Herbsleb, J. & Zubrow, D. (1994). article in this issue.

Humphrey, W. (1988). Characterizing the software process. *IEEE Software*, 5 (2), 73-79.

Humphrey, W.S. (1989). *Managing the Software Process*. Reading, MA: Addison Wesley.

Humphrey, W.S. (in press). *The Personal Software Process*. Reading, MA: Addison Wesley.

Humphrey, W.S., Snyder, T.R. & Willis, R.R. (1991). Software process improvement at Hughes Aircraft. *IEEE Software*, 8 (4), 11-23.

Paulk, M.C., Curtis, B., Chrissis, M.B., & Weber, C.V. (1993). The Capability Maturity Model for Software, Version 1.1. *IEEE Software*, 10 (4), 18-27.

Paulk, M.C., Weber, C.V., Garcia, S.M., Chrissis, M.B., & Bush, M. (1993). *Key Practices of the Capability Maturity Model, Version 1.1 (Tech. Rep. CMU-SEI-93-TR-25)*. Pittsburgh: Carnegie Mellon University, Software Engineering Institute.

Paulk, M.C., Weber, C.V., Curtis, B., & Chrissis, M.B. (in press). *The Capability Maturity Model for Software: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley.

Senge, P. (1990). *The Fifth Discipline: The Art and Practice of the Learning Organization*. New York: Doubleday.

Wohlwend, H. & Rosenbaum, S. (1993). Results of process improvement in an international company. *Proceedings of the Fifteenth International Conference on Software Engineering*. Washington, DC: IEEE Computer Society, 212-220.