Misconceptions of the Capability Maturity Model[1]

Karl Wiegers
Process Impact
www.processimpact.com

Many organizations are striving to improve their software development processes with the help of the Capability Maturity Model[SM] (the CMM[SM]) for software. Most people who have heard of the CMM know three things: It describes five levels of process maturity, higher is supposed to be better, and most of us are presently at Level 1. However, people hold many common misconceptions about the CMM: how it is structured, how it should be applied, how to advance from one maturity level to the next, and so on.

While using the CMM to guide software process improvement efforts, I have heard software engineers and managers express as facts a variety of misinterpretations about the CMM. The purpose of this article is to clarify some of these common points of confusion. The model really makes quite a bit of sense, but it is not intuitively obvious, and it's not easy to find all the right answers in the standard CMM documentation. Let's begin by reviewing the overall structure of the CMM.

Overview of the CMM

The CMM was developed by the Software Engineering Institute (SEI), a federally-funded research and development center operated by Carnegie Mellon University. The definitive resource is *The Capability Maturity Model: Guidelines for Improving the Software Process*, (Carnegie Mellon University/Software Engineering Institute), published in 1995 by Addison-Wesley. The CMM serves two major purposes: to guide process improvement efforts in a software organization, and to assist with identifying contracting organizations that are qualified to perform software work.

The five maturity levels (Initial, Repeatable, Defined, Managed, and Optimizing) represent evolutionary plateaus on the road to a high level of software process capability. Each maturity level, except the first, defines several key process areas or KPAs—groups of related software practices—all of which must be satisfied in order for an organization to attain that level (Table 1).

Each KPA has two to four goals, all of which must be achieved in order to satisfy the objectives of that KPA. In addition, each KPA describes a number of key practices that typically lead to achieving that KPA's goals. These practices are grouped into five "common features." The key practices of the common feature called Activities Performed define technical and managerial activities that typically lead to satisfying the KPA goals, thereby establishing a specific process capability in that key process area.

---

[1] This article was originally published in *Software Development* magazine November 1996 It is reprinted (with modifications) with permission from *Software Development* magazine.

The other four common features relate to institutionalization of the practices performed in a software organization. Institutionalization means that a practice is routinely applied across the organization, even in times of crisis. Application of that practice has been ingrained in the group's culture, and it is supported with an infrastructure of policies, tools, training, and standards. Without effective institutionalization, process improvements may turn out to be temporary. The institutionalizing common features are:

- Commitment to Perform (encompasses the presence of an organizational policy pertaining to the KPA and specifically assigning key responsibilities)
- Ability to Perform (includes training, resources, and other prerequisites)
- Measurement and Analysis (describes the status and quality measures that are used to control and improve the process)
- Verifying Implementation (describes steps taken to ensure that activities are performed according to established processes and procedures)

Several common themes run through the key process areas of the CMM. For starters, written organizational policies state management expectations around the practice of each KPA. Status and issues are to be reviewed periodically with senior management. A recurrent expectation is that practitioners are trained to perform the activities expected of them. Most activities are to be performed according to documented procedures (in contrast to the oral history of many software cultures). Appreciating such philosophical and practical themes of the CMM is as important as remembering the nuances of each KPA.

As organizations attempt to apply the CMM framework to their software process improvement activities, it's easy to get confused by incomplete or erroneous interpretations of the CMM. Let's clarify some of the misconceptions I have heard about this process improvement model.

Misconceptions About Maturity Levels

Misconception #1: If you are at Level 1, you are pond scum. One problem with the term "process maturity" is that if you are on the low end of the scale, you are "immature" by definition. Some people object to the term "maturity" in this context because it sounds like a value judgment, rather than being an objective appraisal of process capability.

The fact is that most software organizations are operating at the Initial level of the CMM today, yet some of them are successful by many commonly accepted measures (profitability, market share, customer satisfaction). However, other organizations are struggling, delivering poor quality products (or nothing) with substantial cost and schedule overruns.

Being Level 1 does not mean that the members of a software organization are barely breathing (as one manager put it). It does mean that the organization's projects are likely to have less predictability, more rework, more defects, and more schedule slippage than those in a higher maturity organization. The CMM is concerned with organizational process capability; it does not pass judgment on the performance or capabilities of individual software practitioners.

Misconception #2: Level 2 is mostly about software engineering activities, such as requirements analysis, design, coding, and testing. Actually, the Repeatable Level addresses practices that relate to planning, managing, and tracking several fundamental aspects of a software project (see Table 1). The standard software engineering tasks of analysis, design, coding, testing,

and documentation are all included in a large KPA called Software Product Engineering at Level 3.

For an organization to have a repeatable process, the project management controls and discipline that are provided by the Level 2 KPAs must first be established. Without a foundation of disciplined project management, even effective software engineering procedures may be abandoned during times of schedule pressure or rapidly changing requirements.

Misconception #3: You have to perform all of the activities and practices defined at some maturity level in order to achieve that level. Over 120 key practices are defined for the Repeatable Level alone, counting the activities, commitments, abilities, measurements, and verification steps. You do not have to perform every one of these practices in order to achieve Level 2. However, you do need to demonstrate that you are satisfying all of the goals that are defined for the applicable Level 2 KPAs. (If an organization does not engage in subcontracting, the Software Subcontract Management KPA is not applicable.) You may have alternative practices that accomplish the goals of a KPA, but which are not mentioned in the CMM. The key practices are only a guideline—not a requirement—for determining whether goals are satisfied.

Misconception #4: Software measurement is not required until you are approaching Level 4. People having a superficial knowledge of the CMM may be aware that the Managed Level addresses the quantitative measurement of software products and processes. These people sometimes are surprised to learn that measurement is a part of every KPA at every maturity level. The Measurement and Analysis common feature provides a hint to this effect.

Most of these measurements determine the status of activities associated with application of the KPA. Consider the Requirements Management KPA. You should measure the status of each requirement, the effort spent on requirements management activities, and requirements stability (the frequency of change of the requirements). At the higher maturity levels, metrics are increasingly used to monitor progress and manage projects proactively. However, software groups at all maturity levels should begin incorporating fundamental software metrics into their routine operating practices.

Misconception #5: The SEI certifies an organization at a specific maturity level. There is no such thing as being "SEI-certified," and there is no certification associated with achieving a specific CMM maturity level. Perhaps this misunderstanding arises because some people erroneously refer to CMM-based process appraisals as "audits," and certain audits do result in a certification of some sort. The SEI maintains a database of accumulated results from formal process appraisals, but it will not divulge the results for any specific organization.

Misconceptions About Practices

Misconception #6: The CMM requires that you use specific software development practices, tools, and methodologies. The CMM does not stipulate how you must perform software development or management activities. It does require that you document the processes you use, that you follow these processes, and that they be technically sound. The CMM's KPAs define general areas of performance that must be satisfied in order to move to a higher maturity level, but they do not specify all of the techniques to be used.

For example, the Software Project Planning KPA requires that you derive estimates for a software project's effort and costs according to a documented procedure. However, it is *your* documented procedure, which should contain whatever estimating techniques work for you. The CMM does not dictate estimating algorithms, CASE tools, development methodologies, or

standards that you have to apply. Provided your practices in each KPA consistently and verifiably satisfy the goals of that KPA, you can apply any methods that you find to be effective.

Misconception #7: The CMM mandates a waterfall life cycle model. The CMM does require that "a software life cycle with predefined stages of manageable size is identified or defined." However, the CMM does not specify (nor even imply) the life cycle to be used. Indeed, the CMM explicitly states that "there is no intent either to encourage or preclude the use of any particular software life cycle." However, the CMM does impose discipline on the selection and application of each project's life cycle model.

The issue of life cycle models is most fully addressed in the Defined Level. The key points are: descriptions of approved software life cycles are documented as part of an organization's standard software process; a project must select one of those life cycles; and the project can modify the selected life cycle if necessary according to specified tailoring guidelines.

Misconception #8: The Software Quality Assurance KPA is mostly about testing. Many newcomers to the CMM fall into this trap, since the term "quality assurance" so frequently refers to defect detection activities like testing and reviews. Actually, the word "testing" does not appear anywhere in the text of the Software Quality Assurance KPA; testing is addressed in the Level 3 Software Product Engineering KPA.

The CMM states that "the purpose of Software Quality Assurance is to provide management with appropriate visibility into the process being used by the software project and of the products being built." This visibility is provided through audits and reviews that determine whether work products comply with applicable standards, and whether processes conform to documented procedures.

To some, this application of quality assurance turns SQA practitioners into "process police." This negative interpretation should be balanced by the intent for SQA to help software developers consistently use the most effective practices to create high quality products. A group's culture has a lot to do with whether SQA is viewed as a value-added support function, or as a spy agency.

Misconception #9: The CMM requires that you perform software inspections to achieve Level 3. The Defined Level includes a Peer Reviews KPA. The purpose of peer reviews is to remove defects from software work products by having the author's technical peers methodically examine the products. Inspections are certainly one effective way to do this. However, as with other KPAs that deal directly with software engineering practices, the CMM does not dictate the specific review techniques to be used. This KPA primarily addresses the implementation of the organization's peer review activities, including:

- A written policy about peer reviews is required.
- Resources, funding, and training must be provided.
- Peer reviews must be planned.
- The peer review procedures to be used must be documented.

Misconception #10: Having a "tailorable" process really means that you can do whatever you want. Anyone making this claim may be trying to weasel out of following a disciplined process. At the Defined Level, you develop a standard process for the software organization, as well as guidelines and criteria for tailoring this standard process for each project.

You can't just do whatever you feel like, since each project's defined process must be adapted in a structured way from the broader organizational software process.

Misconception #11: Requirements management is the same thing as requirements engineering. Requirements *management*, a Level 2 KPA, focuses on establishing and maintaining an agreement between the customer and the software project team on the software requirements. Activities include having the software group review the requirements, using requirements as the basis for software plans and activities, and managing changes to the requirements in a controlled way. However, this KPA does not deal with requirements *engineering*, the tasks of gathering the right requirements from customers, documenting them, analyzing them, and so forth. Virtually every aspect of requirements engineering that the CMM addresses is found in Activity 2 of the Software Product Engineering KPA at the Defined Level.

Misconceptions About Application

Misconception #12: You cannot work on improving KPAs more than one maturity level higher than your current level. You can tackle process improvements in any area you feel will help your group, whether covered by the CMM or not, since the CMM doesn't address every issue that makes a software project successful. It's important to understand that while you can choose to practice, say, peer reviews, while you are still at Level 1, you *must* satisfy the CMM's goals around peer reviews in order to achieve Level 3.

The CMM's structure is based on the concept that you must stabilize the practices defined at Level 2 before you can be sure of sustaining any process improvements you might make on KPAs from the higher levels. Once the Repeatable Level KPAs (focusing at the individual project level) have been institutionalized, they provide a solid foundation for implementing the organization-wide KPAs at the higher levels. It's hard to create an organization's standard software process if the individual projects are still doing whatever they want in an ad hoc and undocumented fashion.

While you cannot skip maturity levels (say, jump directly from Level 1 to Level 3), you can certainly choose to work on improvements in practices found at the higher levels. Just recognize the risk that you may not be able to sustain those improvements over time or during periods of crisis.

Misconception #13: The CMM mandates bureaucracy and wasteful paperwork. In their zeal to apply the CMM, process enthusiasts sometimes forget to scale the procedures they create to the size of the problem being addressed. The CMM speaks frequently about performing an activity according to a documented procedure. This doesn't mean that you need a shelf full of procedures for every project you undertake. Defined processes are not intended to be barriers to productivity. They are intended to let practitioners apply the best available technical and managerial methods in a disciplined, repeatable, and efficient fashion.

As you develop processes, make them scaleable. As an example, a very simple project plan should be written for a 16-hour project, with specific additional planning components to be added for projects up to 160 hours. Plans for larger projects should contain still more detail to increase the chance of success. Adapt the guidance provided by the CMM to the magnitude and risk of the project, but always write a project plan! Rather than dogmatically applying every detail in the CMM to every project, use the CMM to help you identify those activities that will add the most value to each project.

Misconception #14: The CMM is a quick fix for short-term problems. If anyone believes the CMM is the long-sought software silver bullet, call me right away. I have some prime swamp land in Florida I just know you're going to love.

Sorry, but adopting the CMM will not instantly double your productivity or melt away those unsightly defects while you sleep. It takes time to incorporate improved practices into your current development process and see the benefits. However, a sustained commitment to software process improvement, using the CMM or any other approach, will gradually improve your quality, productivity, and team morale (except for the coding cowboys, who just might ride into the sunset).

Karl's Conceptions About the CMM

The framework for process improvement provided by the CMM can go a long way toward improving the ability of a software organization to be successful on project after project. However, the CMM is not a religion. You should use it as a guide to help you focus your improvement energies where they will likely pay off, rather than simply racing up the maturity scale as fast as you can. Attack your projects' real points of process pain, instead of treating the CMM as a sure-fire prescription for curing what ails you. All projects are different, but most can benefit from improvements in the Level 2 key process areas.

Your software process improvement efforts should run in parallel with your software people improvement activities: hiring, teamwork, skill-building, work environment, recognition, and enlightened management. The best software results come from a sensible balance of people, process, and technology; don't neglect any one of these vital components.

Table 1. Key Process Areas of the Capability Maturity Model.

Maturity Level	Key Process Area
1: Initial	None
2: Repeatable	Requirements Management, Software Project Planning, Software Project Tracking and Oversight, Software Subcontract Management, Software Quality Assurance, Software Configuration Management
3: Defined	Organization Process Focus, Organization Process Definition, Training Program, Integrated Software Management, Software Product Engineering, Intergroup Coordination, Peer Reviews
4: Managed	Quantitative Process Management, Software Quality Management
5: Optimizing	Defect Prevention, Technology Change Management, Process Change Management