A Slightly Skeptical View on CMM (Capability Maturity Model) as an example Of Cargo Cult Software Engineering

At worst, the CMM is a whitewash that obscures the true dynamics of software engineering, suppresses alternative models. If an organization follows it for its own sake, rather than simply as a requirement mandated by a particular government contract, it may very well lead to the collapse of that company's competitive potential.

James Bach

"In times of universal deceit, telling the truth will be a revolutionary act."

-- George Orwell

[Eric Arthur Blair] (1903-1950) British author

The Software Capability Maturity Model (CMM), is a software development methodology that is as close to scam as ISO 9000. The current version is was released in December 2001 by the Software Engineering Institute and is often called version 1.1 of the Capability Maturity Model Integration (CMMI). More politically inclined authors would claim that this is a variant of "Brezhnev socialism" applied to software engineering (or worse a variant of Lysenkoism as there is some government pressure to get the certification), but that's another story. But labels aside the fact that some organization is CMM-certified (and it does not matter to what level -- see below) in current environment should probably be viewed as a slick marketing trick (especially useful for outsourcers).

The initial development of CMM is attributed to Watts Humphrey, who founded the Software Process Program of the Software Engineering Institute (SEI) at Carnegie Mellon University. From 1959 to 1986 he worked for IBM. He holds a bachelor's degree in physics from the University of Chicago, a master's degree in physics from the Illinois Institute of Technology, and a master's degree in business administration from the University of Chicago. It looks like CMM originated from 1987 document written by Watts S. Humphrey (Evolution of CMM and CMMI from SEI):


This was a 40-page document containing a list of questions to be used as an assessment tool. Each question was mapped to the five levels, still present today. To achieve a level, an organization had to demonstrate they could answer "Yes" to 90% of the "starred" questions and 80% of all questions for that level.

However, there was also a second "Technology" dimension, with two levels A and B, which was displayed vertically (with the 5-levels horizontal)! The technology dimension assessed the level of automation present. Organizations "matured" from 1A to 5B.

Like it is often the case with questionable doctrines his views of CMM are more realistic then many of his followers and he has second thought about the effectiveness of his creation (Sidebar Watts Humphrey on Software Quality):

Is CMM the only quality tool software developers need?

The CMM framework is essentially aimed at how do you establish a good management environment for doing engineering work. It's about the planning you need, the configuration management, the practices, the policies -- all that stuff. It doesn't talk about how you do things.

When I looked at organizations that were at high [CMM] levels, I discovered that the engineering practices hadn't changed that much. I had naively assumed that when we put good practices in
place, like planning and measurement and quality management, that it would seep down to the engineers [programmers], and they'd start to use it in their personal work. It didn't happen.

The author probably never read Brooks famous The Mythical Man-Month book. In his later, 1987, essay "No Silver Bullet," Frederick P. Brooks wrote,

"The essence of a software entity is a construct of interlocking concepts ... I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation ... If this is true, building software will always be hard. There is inherently no silver bullet."

Road to hell is paved with good intentions. Good work can be done under any software development model. But excessive bureaucratization inevitably leads to bad work no matter what software development methodology is used. What is important is that CMM shifts focus from improvement of real capabilities of software organization into creation of useless, expressly defined bureaucratic perversions. Excessive, I would say obsessive, focus on procedures as well as an illusive goal of "process improvement" is the most detrimental and dangerous feature of CMM. It directly encourages excessive bureaucratization, mandates wasteful paperwork in best "mature socialism" style. Some problem associated with the CMM are somewhat similar to problems of traditional waterfall approach to software development: both are detached from reality. In many ways CMM's activity-based measurement approach is similar to the sequential paradigm mandated by the waterfall software development model. Here is a relevant quote from CMM vs. CMMI: from Conventional to Modern Software Management - article originally published in The Rational Edge, February 2002

Is the CMM Obsolete?

Some issues associated with the practice of the CMM are also recurring symptoms of traditional waterfall approaches and overly process-based management. The CMM's activity-based measurement approach is very much in alignment with the sequential, activity-based management paradigm of the waterfall process (i.e., do requirements activities, then design activities, then coding activities, then unit testing activities, then integration activities, then system acceptance testing). This probably explains why many organizations' perspectives on the CMM are anchored in the waterfall mentality.

Alternatively, iterative development techniques, software industry best practices, and economic motivations drive organizations to take a more results-based approach: Develop the business case, vision, and prototype solution; elaborate into a baseline architecture; elaborate into usable releases; and then finalize into fieldable releases. Although the CMMI remains an activity-based approach (and this is a fundamental flaw), it does integrate many of the industry's modern best practices, and it discourages much of the default alignment with the waterfall mentality.

One way to analyze CMM and CMMI alignment with the waterfall model and iterative development, respectively, is to look at whether each model's KPAs motivate sound software management principles for these two different development approaches. First, we will define those software management principles. Over the last ten years, I have compiled two sets: one for succeeding with the conventional, waterfall approach and one for succeeding with a modern, iterative approach. Admittedly, these "Top Ten Principles" have no scientific basis and provide only a coarse description of patterns for success with their respective management approaches. Nevertheless, they do provide a suitable framework for my view that the CMM is aligned with the waterfall mentality, whereas the CMMI is more aligned with an iterative mentality.

If you look at official and semi-official documents the amount of Dilbert type "management-speak" and acronyms is really staggering; see for example CMM-Tutorial and phillips2004. Some aspects of CMM might make some sense for software maintenance, but hardly for software development.

CMM presupposes certification of organizations for 4 "maturity" levels (from 2 to 5):

1. Initial. Ad hoc process (chaotic, ad hoc, heroic)
2. Repeatable. Basic project management (project management, process discipline)
3. **Defined level.** Process definition is used (institutionalized)
4. **Managed level.** Process measurement is used (quantified)
5. **Optimizing level.** Process control is used (process improvement)

Like one critic of CMM aptly noted, the initial level of CMM (Level 2 or "managed") is actually a certification "of the ability to stand upright and make fire" applied to software development. This is so basic (and fuzzy) that any software development organization can legitimately claim CMM level 2 readiness.

The most insightful critique of CMM was provided by James Bach in his article *The Immaturity of CMM* originally published in the September 1994 issue of American Programmer.

Here is one relevant quote from James Bach's paper (I strongly encourage to read it) which dispels the "institutionalization" myth that is the cornerstone of CMM:

*The idea that process makes up for mediocrity is a pillar of the CMM, wherein humans are apparently subordinated to defined processes.* But, where is the justification for this? To render excellence less important the problem solving tasks would somehow have to be embodied in the process itself. I've never seen such a process, but if one exists, it would have to be quite complex. *Imagine a process definition for playing a repeatably good chess game. Such a process exists, but is useful only to computers; a process useful to humans has neither been documented nor taught as a series of unambiguous steps. Aren't software problems at least as complex as chess problems?*

The CMM reveres institutionalization of process for its own sake. Since the CMM is principally concerned with an organization's ability to commit, such a bias is understandable. *But, an organization's ability to commit is merely an expression of a project team's ability to execute.* Even if necessary processes are not institutionalized formally, they may very well be in place, informally, by virtue of the skill of the team members.

*Institutionalization guarantees nothing, and efforts to institutionalize often lead to a bifurcation between an oversimplified public process and a rich private process that must be practiced undercover.* Even if institutionalization is useful, why not instead institutionalize a system for identifying and keeping key contributors in the organization, and leave processes up to them? The CMM contains very little information on process dynamics.

In other words, the right organizational processes can improve the output of a group of talented software developers, but they do not create one. By ignoring this critical fact, the CMM loses credibility with anyone experienced with a wide range of software development projects.

In his review of Back's groundbreaking and courageous paper Kelly Nehowig observed:

The author describes six basic problem areas that he has identified with the CMM:

1. **The CMM has no formal theoretical basis and in fact is based on the experience of very knowledgeable people.** Because of this lack of theoretical proof, any other model based on experiences of other experts would have equal veracity.
2. **The CMM does not have good empirical support and this same empirical support could also be construed to support other models.** Without a comparison of alternative process models under a controlled study, an empirical case cannot be built to substantiate the SEI's claims regarding the CMM. Primarily, the model is based on the experiences of large government contractors and of Watts Humphrey's own experience in the mainframe world. It does not represent the successful experiences of many shrink-wrap companies that are judged to be a "level 1" organization by the CMM.
3. **The CMM ignores the importance of people involved with the software process by assuming that processes can somehow render individual excellence less important.** In order for this to be the case, problem-solving tasks would somehow have to be included in the process itself, which the CMM does not begin to address.
4. **The CMM reveres the institutionalization of process for its own sake.** This guarantees nothing and in some cases, the institutionalization of processes may lead
to oversimplified public processes, ignoring the actual successful practice of the organization.

5. **The CMM does not effectively describe any information on process dynamics, which confuses the study of the relationships between practices and levels within the CMM.** The CMM does not perceive or adapt to the conditions of the client organization. Arguably, most and perhaps all of the key practices of the CMM at its various levels could be performed usefully at level 1, depending on the particular dynamics of an organization. Instead of modeling these process dynamics, the CMM merely stratifies them.

6. **The CMM encourages the achievement of a higher maturity level in some cases by displacing the true mission, which is improving the process and overall software quality.** This may effectively "blind" an organization to the most effective use of its resources.

The author’s most compelling argument against the CMM is the many successful software companies that, according to the CMM, should not exist. Many software companies that provide “shrink wrap” software such as Microsoft, Symantec, and Lotus would definitely be classified by the CMM as level 1 companies. **In these companies, innovation reigns supreme, and it is from the perspective of the innovator that the CMM seems lost.**

The author claims that innovation per se does not appear in the CMM at all, and is only suggested by level 5. **Preoccupied with predictability, the CMM is ignorant of the dynamics of innovation.** In fact, where innovators advise companies to be flexible, to push authority down into the organization, and to recommend constant constructive innovation, the CMM mistakes all of these attributes to the chaos that it represents in level 1 companies. **Because the CMM is distrustful of personal contributions, ignorant of the environment needed to nurture innovative thinking, and content to bury organizations under an ineffective superstructure, achieving level 2 on the CMM scale may actually destroy the very thing that caused the company to be successful in the first place.**

The highest level (Level 5) means the ability to use a double set of books and produce a lot of bogus paperwork in English language. It has tremendous marketing value, especially if the other side is represented by PHBs. For that reason this level of CMM certification is simply loved by outsourcers. Nine Indian firms claim level 5 certification, and not without a reason :-). If you look at Usenet discussion of CMM hype the most strong defenders of this marketing trick are people connected to outsourcers. The level of argumentation reminds me the USSR Communist Party Congresses with its long applauses, changing into a standing ovation for each monstrous stupidity invented by the Politburo jerks ;-).

Sometimes CMM-compliance is mandated. In this case the less efforts spend on obtaining it the better. In fact, anyone can proclaim themselves to be on CMM Level they want without any significant changes in the software development process. This is a "paper tiger" type of certifications: all that is needed is paperwork.

At the same time too much zeal in achieving CMM-compliance can be very destructive for the organization. The CMM absolutize the value of formal processes, but ignores people. And it is people, the software developers, who are the key to success. This is readily apparent to anyone who is familiar with the work of Gerald Weinberg on programming psychology. The net result of excessive zeal in achieving CMM compliance can be the proliferation of dangerous and clueless "software development bureaucracy" and micromanagers. If this is possible I would recommend for CIO to find volunteers who work on CMM-compliance, created an appropriate organization unit and after it is achieved dismantle or outsource the unit and let go people who were the most enthusiastic about the whole process ;-) They were extremely dangerous for the organization health anyway.

All-in-all obtaining CMM certification is by-and-large a waist of organizational resources, but you might need to do in order to participate in government contacts. If you do, please it take it easy and understands that this is pretty much useless exercise. Still the world is not perfect and sometime you need to play the game. The most constructive was to play CMM game is to concentrate on introduction of automation tools like bug tracking.
software (for example Bugzilla), test automation tools (for example DejaGnu) and compilation and linkage automation software (for example Tivoli can be adapted for this purpose).

But the key issue here is to block the promotion to management ranks a special category of people who thrive under the organizational atmosphere of "software development socialism". Those people are the most dangerous and destructive for any software development organization and CMM can serve as a litmus test for exposing them. If CMM process helps them to grow in the management ranks everything is lost, if opposite is true (as with the suggestion above that CMM-compliance unit should first be created and then outsourced :-) then CMM process can even be useful.

Remember about the danger of "software development socialism", folks ;-). Such side effects of typical CMM adoption as bureaucratization, micromanagement and promotion of wrong type of people should never be overlooked as they kill software developers creativity and any innovation capability within the organization. Everything becomes way too predictable as in "predictable failure". And you know what happen with the USSR with its "mature socialism", don't you?

Software companies which try to push technology envelope would be better off ignoring CMM. As Back noted "Studies alleging that the CMM is valuable don't consider alternatives, and leave out critical data that would allow a full analysis of what's going on in companies that claim to have moved up in CMM levels and to have benefited for that reason." All-in-all CMM is just yet another variant of cargo cult science.