The Capability Im-Maturity Model (CIMM)

Capt. Tom Schorsch
U.S. Air Force

The Capability Maturity Model (CMM) provides a framework to guide and measure software engineering improvement efforts by enabling organizations to assess their software engineering capabilities at one of the five levels of software process maturity. In the CMM, the higher the level your organization is assessed at the better (in theory) your organization is at consistently producing software that fulfills specifications, is on time and is under budget. This tongue-in-cheek article extends the existing five levels downward by describing additional levels of process maturity (or im-maturity). Each of the new lower levels has a characteristic behavior associated with it that defines the level (Negligent, Obstructive, Contemptuous, Undermining). It is my hope that this article will help us recognize these aberrant behaviors within ourselves and our organizations.

The Capability Maturity Model provides software organizations guidance on how to gain control of their processes to develop and maintain software and how to evolve toward a culture of software engineering and management excellence. The Software Engineering Institute's (SEI) Capability Maturity Model (CMM) defines five levels of Software Process Maturity as shown in Table 1. According to SEI data, more than 70 percent of all software organizations are at Level 1. This information may be misleading. In actuality, many organizations may lie well below the merely chaotic. All software organizations are assumed to be at Level 1 simply because no lower levels exist within the CMM framework. This article defines and describes lower maturity levels and their associated Kounter Productive Attitudes (KPAs) (in CMM literature KPA stands for Key Process Area). It is my hope that organizations can use this new expanded model to better judge their true maturity levels.

Table 1. *The five levels of software maturity.*

Level	Description
Characteristic	
5. Optimizing Continuous Improvement	The organization has quantitative feedback systems in place to identify process weaknesses and strengthen them pro-actively. Project teams analyze defects to determine their causes; software processes are evaluated and updated to prevent known types of defects from recurring.
4. Managed Predictable	Detailed software process and product quality metrics establish the quantitative evaluation foundation. Meaningful variations in process performance can be distinguished from random noise, and trends in process and product qualities can be predicted.
3. Defined Standard and Consistent	Processes for management and engineering are documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing software.
2. Repeatable Intuitive	Basic project management processes are established to track cost, schedule, and functionality. Planning and managing new products is based on experience with similar projects.
1. Initial Ad hoc and Chaotic	Few processes are defined, and success depends more on individual heroic efforts than on following a process and using a synergistic team effort.

What levels could possibly be worse than ad hoc and chaotic? Actually, quite a few. The CIMM addresses a serious oversight in the CMM by adding the maturity levels 0 to -3 as shown in Table 2 and described below.

Table 2. *The four levels of software immaturity.*

Level	Description
Characteristic	
0. Negligent Indifference	Failure to allow successful development process to succeed. All problems are perceived to be technical problems. Managerial and quality assurance activities are deemed to be overhead and superfluous to the task of software development process. Reliance on silver pellets.
-1. Obstructive Counter Productive	Counterproductive processes are imposed. Processes are rigidly defined and adherence to the form is stressed. Ritualistic ceremonies abound. Collective management precludes assigning responsibility. Status quo über alles.
-2. Contemptuous Arrogance	Disregard for good software engineering institutionalized. Complete schism between software development activities and software process improvement activities. Complete lack of a training program.
-3. Undermining Sabotage	Total neglect of own charter, conscious discrediting of peer organizations software process improvement efforts. Rewarding failure and poor performance.

This article is based on the original research on lower maturity levels presented by Finkelstein in his immortal work, "A Software Process Immaturity Model." [1] This article extends Finkelstein's work by introducing many new Kounter Productive Areas and extending the taxonomy of immaturity levels, thus accelerating the research being done in this vitally important area.

Level 0: The Negligent Level

The ad hoc and chaotic processes followed by software development organizations at Level 1 can, by exploiting the heroic efforts of individuals, produce software. Level 0 Negligent organizations act in such a way as to prevent these heroic efforts from ever bearing any fruit. Apathy, indifference, and disorganization are the Kounter Productive Attitudes of a Level 0 organization. Where specifications and documentation are produced, a Level 0 organization will fail to look at them ever again. When a successful product is nearly complete, a Level 0 organization will change the requirements or add to them to ensure the project's failure. When an executable is actually produced, the lack of configuration control will ensure that the wrong version is installed. At no time will the end user be allowed to view the process or products prior to delivery as this would eliminate the element of surprise.

While negligent attitudes exist in all people within a Level 0 organization, it is most evident in management. All immature software development organizations (those below CMM Level 1) fail to recognize that management is severely lacking. Negligent organizations believe firmly that technical problems cause the poor software quality and schedule delays. Usually, the only technical issue is that most Level 0 managers do not have a technical understanding of the systems they are managing. Any technical decisions that are made are done so on an isolated case-by-case basis with no long-term goals or vision to drive them. The high-level management that does take place in Level 0 organizations is typically classified as management by exception. This type of management occurs only in a reactive mode as crises develop

and the filtering threshold for problems is exceeded (commonly called "firefighting," "fighting alligators," or in the case of major crises, "dragon slaying"). In keeping with the animal motif, these crises are typically overwhelmed by the relentless activity of "tiger" teams.

Managers in Level 0 organizations rely heavily on the use of "silver pellets" to save them from themselves. Examples include case tools, reuse, metrics, open systems, client/server, business process improvement, total quality management, CMM, or whatever the "road to improvement du jour" is. These often excellent improvement efforts never make much difference in Level 0 organizations as each new silver pellet is introduced with great fanfare but is inevitably overcome by apathy and lack of commitment by management. Any real change requires achieving more momentum and follow through than a Negligent organization can muster. Most improvement efforts merely fade away because of the lack of empowerment in the initiators and implementors of such activity, and indifference by the majority not wanting to upset the status quo. Improvement activities that survive will not be given measurable goals that relate to the software being produced or the software development process. Prior to them fading away, these latter improvement activities will eventually be declared successes without any measurable change taking place.

In an additional manifestation of apathy, negligent organizations rarely have an organizational vision or goals. Creating an organizational vision and goals takes too much effort, and no one agrees on them anyway. Should a Level 0 organization manage to develop a set of goals, it will be derelict in creating a plan for achieving those goals (or the goals are not quantifiable; therefore, any plan they come up with will do). Without a plan and an associated schedule and resources, the vision and goals will languish in obscurity and eventually be forgotten completely.

Level -1: The Obstructive Level

Level -1 software development organizations impose additional hardships upon the software practitioner. Obstructive software development organizations go beyond Level 0’s simple negligence by subconsciously subverting software development activities. The Kounter Productive Attitudes in Level -1 organizations include being overly rigid and formal and having a one size fits all mentality. These organizations insist on complex processes, involving the use of arcane programming languages, outdated hardware and commercial-off-the-shelf products, and inappropriate documentation deliverables. Level -1 organizations deploy significant effort and a substantial portion of their human and dollar resources in order to impose these inappropriate processes and products across all projects, for all software development, no matter how large or how small. The unsuitable software processes have no owners and no method for changing the processes, and in fact modifications to the existing processes are actively discouraged. Strong Level -1 organizations actually make changes to the software development processes impossible except where additional controls can be added that increase the probability of failure.

Level -1 organizations use collective management to supervise software development efforts. Collective management is the process of dividing complex software development efforts into many pieces or phases each with their own manager (who usually functions as an overseer) and then doing away with the overall project manager. [With distributed management, as apposed collective management, the overall project manager is retained and is active in managing the software development project and coordinating the activities of the subordinate managers.] Thus, management of the overall software development effort is done collectively by the managers of the individual phases. Over time, these phases tend to grow in size and duration, requiring more and more sub-managers, while at the same time drifting apart leaving gaps in the development process or overlapping each other causing duplicative efforts. Since responsibility for the end product is also collective this style of management is very effective at hiding the root causes of problems by distributing it over a number of managers and software development phases, thus enabling each manager to blame the others.

The formalism and rigidity in Level -1 organizations reveals itself in excessive ritualism and ceremony that further stifles the software development process. The activities and deliverables in an Obstructive organization’s software development process all have acronym code words associated with them that hide their true purpose from the unwashed and uninitiated. Many activities are followed and documents are produced by the different software development phases with great fanfare and rigid punctuality. The

purpose of these activities and products is not documented or trained but is ritualistically followed and produced. Eventually the reason for the activities and products is forgotten. To ensure this happens quickly and consistently, the quality control efforts in Level -1 organizations revolve around ensuring the activity has taken place and the documentation has been produced according to the correct format. Quality control efforts in Level -1 organizations never actually check the quality of the activity or quality of the documentation contents. There are too many required activities and documents to check and it is difficult to judge the quality of activities and documents whose purpose is not defined. Software development practitioners in Level -1 organizations actually do double the work because they perform the ceremonial activities in parallel with the actual software development activities that produce the end product.

Level -1 organizations insist on using approaches for which tool support is unavailable. Where tool support is available, they impose procurement standards which prevent its purchase. Where purchasing the tool is an option, all existing commercial tools will be deemed inappropriate and the Level -1 organization will develop tool support in-house. Eventually, the tool development group within the organization will expand and grow until it consumes more resources than the original development effort.

Level -2: The Contemptuous Level

Level -1 organizations sincerely believe they are assisting software development efforts and following good software development practices despite overwhelming evidence to the contrary. In contrast, Level -2 organizations are openly contemptuous of software engineering practices. The Kounter Productive Attitudes in Level - 2 organizations include a complete disregard and near utter rejection of any effort to improve the organization or the way it develops software.

Level -2 organization exhibit their disapproval of improvement activities in their lack of a training program. Contemptuous organizations provide no training as

- There is no training budget.
- There is no time.
- It is an irrelevant waste of time anyhow.

All new people are expected to know their jobs already or to be trained by on-the-job training (by the person who left two months before they arrived for work). If trained software engineers are hired, they are criticized for having book learning but no real-world software development experience. If new hires have software development experience, they are criticized for having software "development" experience instead of software "maintenance" experience (or vice versa depending on the circumstances). If they have both types of experience, they are told that this system (the one being developed or maintained) is different, the organization is different, or the end user is different and that those software engineering ideas will not work in this environment. Any existing experienced software engineers are either disinvited from any forums that would enable them to air their views or are congratulated on their keen insight and then quietly reassigned to administrative positions far away from software development.

In order to hold off outside oversight of their activities, many Level -2 organizations display a feigned tolerance of software process improvement activities and products. For example, a software development manual inevitably exists in every Level -2 organization, but it will only be occasionally dragged out from its resting place to be presented in ceremonious fashion to some oversight body to prove that it actually exists. Many people within the organization acknowledge the manuals existence and make reference to it but have never actually read it.

Should a ground swell of software development improvement activities take place in a Level -2 organization, it will be immediately crushed from within. Managers within a Level -2 organization insist that they only have time to develop software, not to improve how they develop software. This leads to the separation of the "doers" from the "improvers" within the Level -2 organization. The doers believe that improving how software is developed is not their job; they only have time to "support the mission." The improvers do not accomplish any real mission support so their advice is obviously only academic as they are not doers. Any attempts at getting these two groups together is halted before it can begin.

Level -3: The Undermining Level

Not content to thwart its own processes, a Level -3 organization actively seeks to discredit and disrupt the work of other organizations. When the work done by a peer organization cannot be discredited, the Level -3 organization will claim credit for as much of the work as possible. A Level -3 ignores its own software development processes in favor of developing positive publicity for itself that focuses on creating a nice red skin over an apple that may be rotten to the core. The Kounter Productive Attitudes associated with a Level -3 organization include believing that looking good is more important than being good, that any attention is better than no attention, and that as long as they can keep the money rolling in they are successful. A sabotaging organization does not care if they produce poor software as this ensures job security and guarantees more money to maintain the software system over its entire lifetime.

A Level -3 organization tries to ensure that all other organizations are (or are perceived to be) worse than the Level -3 organization. Since it is easier to destroy than build up, an Undermining organization accomplishes this by deliberately compromising and damaging any competitors. After all, the best defense is a good offense. Any weaknesses in neighboring organizations are willfully and methodically exploited. What better way to keep "process fascists" at bay than to demonstrate how badly improvement efforts have failed in a neighboring organization. What better way to gain more stature and responsibility, money, resources, and people for yourself than to show how incompetent adjacent organizations are.

Level -3 organizations glorify failure and poor performance. Given two identical software development efforts, the effort that is on time, under budget, and pleases the end user is ignored while the effort that is late, laden with problems, and is grudgingly accepted by the end user is praised for producing any results at all. No one cares about the "easy" success stories. Everyone loves the story where the "average Joe" overcomes (usually self imposed) adversity. This "reverse incentive" attitude actually encourages the use of mismanagement. Time spent on improving the way things are done (working smarter) and achieving direct results (working harder) is treated as infinitely less valuable than promoting the facade that the development effort is a success.

Conclusion

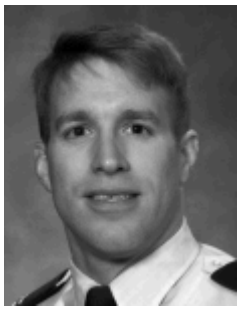
The usefulness of the CIMM level taxonomy described above in accurately judging an organization’s software development capability level when it is below CMM Level 1 cannot be determined until adequate assessment programs for the additional levels evolve. In the meantime, each Kounter Productive Area and level characteristic revealed above must be screened for insuccessive CMM assessments of all organizations to ensure that each software development organization can be appraised at its true capability maturity level.

Postscript

Obviously, the sub-level 1 layering described in this pseudoscientific article is a comic invention, but the Kounter Productive Attitudes are unfortunately far too real and exist in organizations at all CMM levels. Such attitudes, if widespread enough, can completely prevent process improvement efforts from making a difference.

After reading this article, I hope that we will be able to more easily recognize these Kounter Productive Attitudes in ourselves. Changing attitudes is difficult because they have become ingrained in individuals and part of the organizational culture. But recognizing them is the first step.

About the Author



Capt. Thomas M. Schorsch is a full-time doctorate student at The Air Force Institute of Technology at WrightPatterson Air Force Base, Ohio where he is doing research in software engineering. Most recently, Schorsch was a project manager at the Space and Warning Systems Center at Peterson Air Force Base Colo. where he led the development and fielding of a new Cheyenne Mountain Command and Control system that went from initial concept to operational status in 10 months. Before that, he was an assistant professor of computer science at the U.S. Air Force Academy.

AFIT/ENG Bldg. 640,
Box 4258
2950 P Street
WrightPatterson AFB, OH 454337765
Voice: 937-2553636 ext. 4602
DSN 7853636 ext. 4602
E-mail: tschorsc@afit.af.mil

Reference

1. Finkelstein, "A Software Process Immaturity Model," *ACM SIGSOFT, Software Engineering Notes,* Vol. 17, No. 4, October 1992, pp. 22-23.

This article is reprinted with permission from the August 1996 issue of Nexus, a software engineering process group newsletter.
