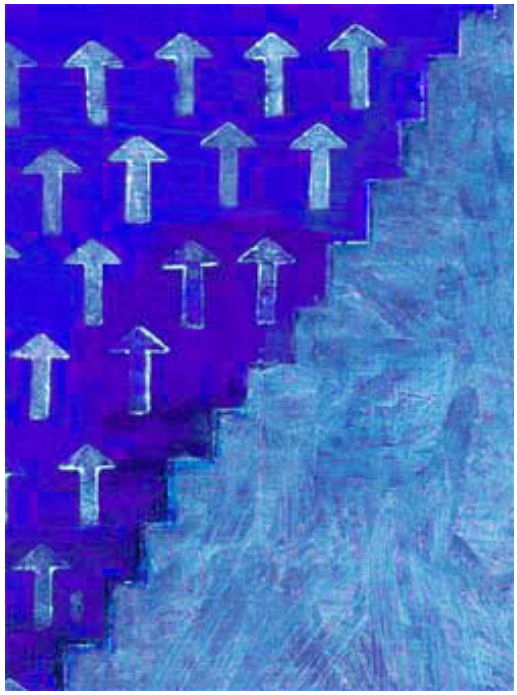
CMM vs. CMMI: From Conventional to Modern Software Management

by **Walker Royce**
Vice President and General Manager
Strategic Services
Rational Software Corporation



This article summarizes some thoughts on making the transition from conventional software management techniques to modern ones. In particular, I want to endorse the improvements in the Software Engineering Institute's new CMMI (Capability Maturity Model Integration)[1] approach and motivate development organizations to apply the approach correctly. Although I have always been a supporter of the spirit behind the original Capability Maturity Model (CMM), in practice, it has been misused and misinterpreted too much for my liking. Based on my twenty-five years of experience with many of the world's leading software development organizations engaged in process improvement, I am convinced that most organizations using the CMM are still entrenched in a default waterfall model mentality. I won't lay blame on the model itself, for I am aware of some process improvements made within a CMM context that were very much based on a modern, iterative approach to development. But this enlightened interpretation is not the norm.

CMM Overview

The CMM defines five levels of software process maturity, based on an organization's support for certain key process areas (KPAs). Level 1 (initial) describes an organization with an immature or undefined process. Level 2 (repeatable), Level 3 (defined), Level 4 (managed), and Level 5 (optimizing), respectively, describe organizations with successively higher levels of software process maturity.

The associated KPAs for these levels are:

- *Level 2*: requirements management; software project planning; software project tracking and oversight; software subcontract management; software quality assurance; software configuration management
- *Level 3:* organizational process focus, organizational process definition, training program, integrated software management, software product engineering, intergroup coordination, peer reviews
- *Level 4:* process measurement and analysis; quality management; defect prevention
- *Level 5:* technology innovation, process change management

The primary goal for most organizations is to achieve a Level 3 maturity. One instrument for assessing an organization's current maturity level is a software capability evaluation (SCE), which determines whether the organization "says what it does and does what it says" by evaluating its software process (usually in the form of policy statements) and project practices. The organization's process captures the "say what you do," and project implementations (specific tailorings and interpretations of this process) should demonstrate the "do what you say."

Issues with the CMM

One of the key issues I have encountered with the CMM is that the KPAs focus mostly on activities and supporting artifacts associated with a conventional waterfall process: requirements specifications, documented plans, quality assurance audits and inspections, and documented processes and procedures. Very few of the KPAs address the evolving results (i.e., the software product) and associated engineering artifacts (use-case models, design models, source code, or executable code) that capture the real target product. Also, there is no emphasis on the architecting/design process, assessment process, or deployment process, all of which have proven to be key discriminators for project success.

The CMM also overemphasizes peer reviews, inspections, and traditional Quality Assurance "policing" methods. Although manual reviews and inspections may be capable of uncovering sixty percent of errors, they rarely, if ever, uncover the architecturally significant flaws that plague most conventionally managed software projects. And I have never encountered an architect, lead designer, lead tester, or project manager who said inspections and peer reviews were the critical discriminators for success.

Another issue is that most implementations of the CMM drive organizations to produce more documents, more checkpoints, more artifacts, more traceability, more reviews, and more plans. Furthermore, thicker documents, more detailed information, and longer meetings are considered to be better. This flies in the face of the primary technique for improving software economics: reducing complexity and the volume of human-generated "stuff." In reality, the widespread belief that "more artifacts and more precise artifacts correlate to more progress" is not the CMM's fault, but

the CMM does not motivate organizations to behave otherwise.

Getting an accurate measure of an organization's current maturity level is also an issue. The CMM takes an *activity-based* approach to measuring maturity; if you do the prescribed set of foundation project activities, you are Level 2. If you then do a prescribed set of activities as an organization, you are Level 3. And so on. There is nothing that characterizes or quantifies whether you do these activities *well enough* to deliver the intended results. I prefer a more *results-driven* measurement scheme: If you can repeat the process on several projects with predictable cost, quality, and schedule, then you are Level 2. If you can improve one dimension of cost, quality, or schedule on subsequent projects, then you are Level 3. Improvements along multiple dimensions will get you to higher levels. In reality, however, neither an activity-based perspective nor a results-driven perspective alone is sufficient; we need to combine these two approaches to measure maturity accurately.

A default practice we see too often is that organizations define their process based strictly on traceability to the CMM, so that they will have a very clear mapping for assessment. The waterfall model's requirements-driven practices lead organizations down a similar path by making traceability of requirements specifications to design elements a more important measure of design quality than meeting the needs of the user (which are often poorly represented by conventional requirements specifications). Organizations concerned about real improvement -- as opposed to simply passing an audit -- focus on achieving a process that produces improved business results. A complete assessment framework, therefore, should also measure real improvement along project performance dimensions such as estimated time to market, probable cost to complete, and predicted quality of product. Software development standards (like the RUP or ISO 12207), frameworks designed to help organizations decide how to do things and what to produce, can lay the groundwork for accurate assessments. Unfortunately, default practice in many organizations is to apply the CMM as both a development standard and an assessment standard.

CMMI Overview

The initial Capability Maturity Model (CMM v1.0) was developed by the Software Engineering Institute and specifically addressed software process maturity. It was first released in 1990, and after its successful adoption and usage in many domains, other CMMs were developed for other disciplines and functions such as Systems Engineering, people, integrated product development, software acquisition, and others. Although many organizations found these models to be useful, they also struggled with problems caused by overlap, inconsistencies, and integration. Many organizations also confronted conflicting demands between these models and ISO 9001 audits or other process improvement programs.

The CMM Integration (CMMI) Project was conceived as an initiative to integrate the various CMMs into a set of integrated models. The source models that served as the basis for the CMMI include: CMM for Software V2.0 (Draft C), EIA-731 Systems Engineering, and IPD CMM (IPD) V0.98a.

The CMMI, like its predecessor, describes five distinct levels of maturity:

1. *Level 1 (initial)* represents a process maturity characterized by unpredictable results. Ad hoc approaches, methods, notations, tools, and reactive management translate into a process dependent predominantly on the skills of the team to succeed.
2. *Level 2 (managed)* represents a process maturity characterized by repeatable project performance. The organization uses foundation disciplines for requirements management; project planning; project monitoring and control; supplier agreement management; product and process quality assurance; configuration management and measurement/analysis. For Level 2, the key process focus is on project-level activities and practices.
3. *Level 3 (defined)* represents a process maturity characterized by improving project performance within an organization. Consistent, cross-project disciplines for Level 2 key process areas are emphasized to establish organization-level activities and practices. Additional organizational process areas include:
    - *Requirements development:* multi-stakeholder requirements evolution.
    - *Technical solution:* evolutionary design and quality engineering.
    - *Product integration:* continuous integration, interface control, change management.
    - *Verification:* assessment techniques to ensure that the product is built correctly.
    - *Validation:* assessment techniques to ensure that the right product is built.
    - *Risk management:* detection, prioritization, and resolution of relevant issues and contingencies.
    - *Organizational training:* establishing mechanisms for developing more proficient people.
    - *Organizational process focus:* establishing an organizational framework for project process definition.
    - *Decision analysis and resolution:* systematic alternative assessment.
    - *Organizational process definition:* treatment of process as a persistent, evolving asset of an organization.
    - *Integrated project management:* methods for unifying the various teams and stakeholders within a project.
4. *Level 4 (quantitatively managed)* represents a process maturity characterized by improving organizational performance. Historical results for Level 3 projects can be exploited to make trade offs, with predictable results, among competing dimensions of business performance (cost, quality, timeliness). Additional Level 4 process areas include:
    - *Organizational process performance:* setting norms and

benchmarks for process performance.

- *Quantitative project management:* executing projects based on statistical quality-control methods.

5. *Level 5 (optimized)* represents a process maturity characterized by rapidly reconfigurable organizational performance as well as quantitative, continuous process improvement. Additional Level 5 process areas include:
   - *Causal analysis and resolution:* proactive fault avoidance and best practice reinforcement.
   - *Organizational innovation and deployment:* establishing a learning organization that organically adapts and improves.

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.

Top Ten Principles of *Conventional (Waterfall)* Software Management

1. **Freeze requirements before design**. This is the essence of a

requirements-first process: The project team strives to provide a precise requirements definition and then implement exactly those requirements. Changing requirements can cause significant breakage in the code and test phases; consequently, requirements must be completely and unambiguously specified before the team makes major investments in other design and development activities.

2. **Avoid coding prior to detailed design review**. Again, because design changes can also cause significant breakage in the code and test phases, the team needs to ensure that the whole design is mature and complete before beginning the coding phase, when there will be much more resistance to change.

3. **Use a higher-order programming language**. Higher-order programming languages avoid a substantial set of error sources (through advanced data typing, interface separation, and packaging and programming constructs) and permit the software solution to be "programmed" in fewer lines of human-generated code.

4. **Complete unit testing before integration**. Whereas the design flows "top down," the test process flows "bottom-up": The smallest units are completely tested prior to delivery for integration testing. This sequencing constraint is an attempt to capture more bugs at the unit level, prior to integration, when they can cause substantially more scrap and rework.

5. **Maintain detailed traceability among all artifacts**. To ensure that program completeness and consistency can be maintained at each stage, the requirements artifacts need to be traced to design artifacts and test artifacts. When changes are proposed or identified downstream, this provides a full view of the change's actual or potential impact for assessment.

6. **Document and maintain the design**. Design without documentation is not design. In early phases, the documentation *is* the design. In later phases, as code becomes the primary engineering artifact, design artifacts must be updated to ensure consistency and provide a basis for decision making about changes.

7. **Assess quality with an independent team**. To maintain a separate reporting chain from the analysts, designers, and testers, the project should assign to an independent team responsibility for ensuring overall adherence to quality standards -- for both the product and the process.

8. **Inspect everything**. Inspecting the detailed design and code is a much better way to find errors than testing. Ensure that inspections cover all requirements, design, code, and test artifacts.

9. **Plan everything early with high fidelity**. A complete, precise plan down to the "inch-pebble" level that lays out detailed activities and artifacts over the entire schedule is necessary to identify critical paths, manage risks, and evaluate programmatic changes.

10. **Control source code baselines rigorously**. Once artifacts get into the coded stage, rigorous configuration management is necessary to maintain baseline control of formal releases in the test process, and to transition the product to a zero-defect state suitable for release.

Top Ten Principles of *Modern (Iterative)* Software Management

1. **Focus the process on the architecture first**. This requires a demonstrable balance among the driving requirements, architecturally significant design decisions, and lifecycle plans *before* the organization commits sufficient resources for full-scale development.
2. **Attack risks early with an iterative lifecycle**. An iterative process is required to refine understanding of the problem, and to shape an effective solution as well as an effective plan that ensures balanced treatment of all stakeholder objectives. Major risks need to be addressed early to increase predictability and avoid expensive scrap and rework later on.
3. **Emphasize component-based development.** To reduce the amount of human-generated source code and custom development, project teams must move from a line-of-code mentality to a component-based mentality within an existing architectural framework. A component is a cohesive set of pre-existing lines of code, either in source or executable format, with a defined interface and behavior.
4. **Establish a change management environment**. The dynamics of iterative development include concurrent workflows, as different teams work on shared artifacts. This calls for objectively controlled baselines that all project members can view.
5. **Enhance change freedom with tools for round-trip engineering**. Round-trip engineering provides the environment support necessary to automate and synchronize engineering information in different formats (e.g., requirements specifications, design models, source code, and executable code). Without substantial use of automation, it is difficult to reduce iteration cycles to manageable time frames that allow and encourage change. Freedom to change artifacts is a necessity in an iterative process, as it removes one of the predominant sources of friction perceived by the engineering teams.
6. **Use rigorous, model-based design notation**. A model-based approach (e.g., UML) supports the evolution of semantically rich graphical and textual design notations. Visual modeling with rigorous notations and a formal, machine-processable language permits more objective assessment than the traditional human review and inspection of ad hoc design representations in paper documents.
7. **Instrument the process for objective quality control**. Lifecycle assessments of both the process and all intermediate products must be tightly integrated into the process, using well-defined measures derived directly from the evolving engineering artifacts and integrated into all activities and teams.
8. **Use demonstration-based assessment of intermediate artifacts**. Transitioning the current, state-of-the-product artifacts (whether an early prototype, a baseline architecture, or a beta capability) into an executable demonstration of relevant use cases stimulates earlier convergence on integration, more tangible

understanding of design tradeoffs, and earlier elimination of architectural defects.

9. **Plan releases with evolving levels of detail**. It is essential that the software management process drive toward early and continuous demonstrations within the operational context of the system, namely its use cases. Each project increment and demonstration should reflect current levels of detail for both requirements and architecture. Use cases are the primary mechanism for organizing requirements, defining iteration content, assessing implementations, and organizing acceptance testing.
10. **Establish a scalable, configurable process.** No single process is suitable for all software development projects. To be pragmatic, a process framework needs to be configurable to a broad spectrum of applications. To ensure economies of scale and return on investment, the organization must instill a common process "spirit," so that all projects inherit a common set of best practices, especially for project management and context independent workflows, checkpoints, metrics, and artifacts. It should also allow tailoring and specialization so that each project can optimize the process implementation for the specific context of the project.

Alignment Between CMM and Both Sets of Management Principles

Table 1 identifies which principles in each set are directly motivated by the KPAs of the CMM. These are my *judgments*; they are not based on any science, just on anecdotal evidence, experience, and the combined opinions of many field practitioners at Rational. Furthermore, keep in mind that many of these principles are based as much on observations of default practices and organizational inertia as they are on the CMM.

Table 1: How the CMM Motivates Software Management Principles

CMM Motivation for Waterfall Principles	CMM Motivation for Iterative Principles
COLOR KEY • CMM directly motivates organizations to apply this principle. • CMM is neutral; provides no direct motivation but does not de-motivate the organization from applying this principle. • CMM de-motivates organizations from applying this principle.	

1. Freeze requirements before design. 2. Avoid coding prior to detailed design review. 3. Use a higher-order programming language. 4. Complete unit testing before integration. 5. Maintain detailed traceability among all artifacts. 6. Document and maintain the design. 7. Assess quality with an independent team. 8. Inspect everything. 9. Plan everything early with high fidelity. 10. Control source code baselines rigorously.	1. Focus the process on the architecture first. 2. Attack risks early with an iterative lifecycle. 3. Emphasize component-based development. 4. Establish a change management environment. 5. Enhance change freedom with tools for round-trip engineering. 6. Use rigorous, model-based design notation. 7. Instrument the process for objective quality control. 8. Use demonstration-based assessment of intermediate artifacts. 9. Plan releases with evolving levels of detail. 10. Establish a scalable, configurable process.

As Table 1 shows, the CMM's key process areas directly motivate most of the conventional principles but have little influence on the modern principles. In my opinion, a few of the modern principles are actually in conflict with the CMM's key process areas. I am sure this table will stimulate passionate debate among process improvement zealots, but in the end, I believe most engineers and project managers working on the front lines of software development projects will reach the same conclusions I have.

Alignment Between CMMI and Modern Management Principles

Now, lets take a look at the CMMI. If I do the same rough analysis, I come up with the results in Table 2.

Note: This table uses the same color coding scheme as Table 1.

Table 2: How the CMMI Motivates Iterative Software Management Principles

CMMI Alignment with Iterative Principles
1. Focus the process on the architecture first.
2. Attack risks early with an iterative lifecycle.
3. Emphasize component-based development.
4. Establish a change management environment.
5. Enhance change freedom with round-trip engineering.
6. Use rigorous, model-based design notation.
7. Instrument the process for objective quality control.
8. Emphasize demonstration-based assessment.
9. Plan releases with evolving levels of detail.
10. Establish a scalable, configurable process.

Note that my analysis is still based on the industry's observable, default practices rather than on the CMMI's intentions. Our ties to legacy approaches and organizational cultures will be obstacles in achieving the CMMI's real intentions, so I feel conservative in my judgments. Clearly, from my perspective, the CMMI represents a major improvement.

Time to Move On

Although I have made somewhat subjective interpretations of the CMMI and speculated on how organizations will implement various aspects of it,[2] I feel relatively comfortable that the process areas within it now motivate modern software management best practices and align with modern iterative development techniques. I still have concerns, however, that organizations will focus more on activity-based assessment techniques rather than results-based techniques.

In my view, it is time for system development organizations to phase out the CMM and to begin their transition to the CMMI. The CMM has done the software industry a great service by focusing more attention on software process, but after ten years in the field, it is time for the CMM to step aside to make way for the new, improved CMMI.

For those who would like more scientific support for this position, see Joe Marasco's article on assessing project progress in the November 2001 *Rational Edge*.[3] It shows us that by applying some very simple physics (S-Curve, derivatives, Newton's F=ma), we can validate some of the notions I have postulated in this article about modern, iterative development.

Notes

1 See http://www.sei.cmu.edu/cmmi/general/genl.html.

2 I could go through all the details of my analysis, but I doubt that would strengthen the resolve of those who agree with me, and I doubt even more that further rationale would sway the dissenters. Consequently, I will just assert my position.

3 See http://www.therationaledge.com/content/nov_01/k_projectProcess_jm.html.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!