IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Hershey PA 17033-1117, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4159

Process Models: Should the Software and Systems Engineering Communities Capitalize On Their Similarities Or Go Their Separate Ways?

Rick Gibson, PhD
Department of Computer Science and Information System, American University, Washington, DC
Tel: (202) 885-2735, rgibson@american.edu

INTRODUCTION

In response to increasing concerns about software development failures, the Software Engineering Institute (SEI) pioneered a *software* process improvement model in 1988, with the fully developed version of the Capability Maturity Model for Software (SW- CMMâ) appearing in 1993. Since the early nineties, there have been comparable improvement models introduced in the *system* engineering community as well, some of which have been published and widely accepted: Systems Engineering Capability Maturity Model (SE-CMM) also known as the Electronic Industries Alliance Interim Standard (EIA/IS) 731, Systems Engineering Capability Model (SECM); and the Integrated Product Development Capability Maturity Model (IPD-CMM). The resulting avalanche of models and standards has been described by Sarah Sheard (Software Productivity Consortium) as a "Framework Quagmire." In December of 2000, the SEI initiated the Capability Maturity Model–Integrated (CMMI[SM]) project, which combines best practices from the systems and software engineering disciplines. Note: CMMâ and CMMI[SM] are copyrights and service marks of the Software Engineering Institute.

Issues and concerns regarding such an integration were articulated by Barry Boehm and Fred Brooks as early as 1975. Boehm suggested that the adoption of systems engineering reliability techniques by software engineers was counterproductive. Moreover, Brooks' Law suggests that a common system engineering solution to schedule slippage (add more people) will only make late software project even later.

More recently, Boehm (1994) expressed concerns that, in spite of the central function of software in modern systems, the two engineering disciplines have not been well integrated. Boehm articulated similarities and differences shown in Table 1.

Software engineering is defined by IEEE Standard 610.12 as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software—that is, the application of engineering to software. The International Council on Systems Engineering (INCOSE) defines systems engineering as an interdisciplinary approach and means to enable the realization of successful systems.

Definitions of the two disciplines from the CMMI[SM]

Systems Engineering—The systems engineering discipline covers the development of total systems, which may or may not include software. Systems engineers focus on transforming customer needs, expectations, and constraints into product solutions and supporting those product solutions throughout the product life cycle.

Software Engineering—The software engineering discipline covers the development of software systems. Software engineers focus on applying systematic, disciplined, and quantifiable approaches to the development, operation, and maintenance of software.

Table 1: Software and system engineering similiarities and differences

Similarities	Differences
Definition and analysis involves manipulation of symbols.	Software is not subject to physical wear or fatigue
Highly complex aggregation of functions, requiring satisfying (though not optimizing) among multiple criteria.	Copies of software are less subject to imperfections or variations
Decisions driven by need to satisfy quality attributes such as reliability, safety, security, and maintainability.	Software is not constrained by the laws of physics.
Easy and dangerous to suboptimize solutions around individual subsystem functions or quality attributes.	Software interfaces are conceptual, rather than physical—making them more difficult to visualize.
Increasing levels of complexity and interdependency.	Relative to hardware, software testing involves a larger number of distinct logic paths and entities to check.
	Unlike hardware, software errors arrive without notice or a period of graceful degradation.
	Hardware repair restores a system to its previous condition; repair of a software fault generally does not.
	Hardware engineering involves tooling, manufacturing, and longer lead times, while software involves rapid prototyping and fewer repeatable processes.

One purpose of the CMMI[SM] was to evolve the software CMM while integrating the best features of the systems engineering capability models. The combination of the practices of the models into one single framework required more than just combining practices because of differences in interpretation, focus, and terminology. Compromises and intentional inefficiencies were required in order to integrate these models. For example, the CMMI[SM] was released with two representations, continuous and staged, as shown in Table 2, to allow systems and software groups, respectively, to continue using the model representation with which they were already familiar.

With the CMMI[SM] there is significant coverage provided for the engineering dimension, more detailed coverage of risk management and measurement, and enhanced analysis that was less specific in the Software CMM. Moreover, the CMMI[SM] (continuous representation) process areas are grouped by categories, which include: Process Management, Project Management, Support, and Engineering. The Engineering category includes the process areas shown in Table 3, which

This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference.

Table 2: Continuous and staged representations

Systems Engineering—Continuous	Software Engineering—Staged
Migration path from EIA/IS 731	Migration path from SW-CMM
Encourages a focus on process areas to address business objectives—avoids the maturity plateau trap.	Encourages a proven sequence of improvements beginning with basic management practices and progressing through successive levels.
Encourages comparisons across organizations by process areas.	Encourages comparisons among organizations using maturity levels.
Provides increased visibility into capability achieved within a process area. Can measure below Level 2.	Cases studies and empirical data show return on investment for process improvement.
Provides a focus on risks specific to each individual process areas.	Summarizes process improvement results in a single maturity level number.
Encourages the generic practices from higher capability levels be more evenly and completely applied to all process areas.	

Table 3: Engineering process areas

Process Areas	Purpose
Requirements Management	Manage the requirements of the project's products and product components and to identify inconsistencies between those requirements and the project's plans and work products.
Requirements Development	Produce and analyze customer, product, and product components.
Technical Solution	Develop, design, and implement solutions to requirements.
Product Integration	Assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product.
Verification	Assure that selected work products meet their specified requirements.
Validation	Demonstrate that a product or a product component fulfills its intended use when placed in its intended environment

are intended to integrate software and systems engineering by targeting product-oriented business objectives.

INTEGRATED PROCESS MODEL BENEFITS

Rassa (2001) summarizes the benefits of the CMMI[SM] project as follows:

- Common, integrated vision of improvement for all organizational elements;
- Means of representing new discipline-specific information in a standard, proven process improvement context;
- Efficient, effective assessment and improvement across an organization's multiple process disciplines;
- Reduced training and assessment costs.

According to the SEI "The CMMI[SM] effort is intended to support process and product improvement and to reduce redundancy and eliminate inconsistency when using separate stand-alone models. The goal is to improve efficiency, return on investment, and effectiveness by using models that integrate disciplines such as systems engineering and software engineering that are inseparable in a systems development endeavor." (SEI CMMI Frequently Asked Questions). With the arrival of the CMMI[SM], a wider continuum of the product life cycle has been targeted for possible enhancement, no longer limiting process improvement only to the development of software. This integrated approach provides a reduction in the redundancy and intricacy resulting from the use of multiple, separate process improvement models. For organizations that wish to assess their process improvement efforts against multiple disciplines, the CMMI[SM] provides some economies of scale in model training and assessment training. This one evaluation method can provide separate or combined results for the fields of software and system engineering. Furthermore, software organizations can also focus on the amplifications for software engineering within the engineering shared process areas and take advantage of any systems engineering amplifications that are helpful. Although still subject to debate, a distinction is made between base and advanced engineering practices as model constructs. The CMMI[SM] provides the groundwork for enterprise wide process improvement with a new emphasis on products and services as well as process. This emphasis is on both organizational maturity and process capability, for the CMMI[SM] directs increased attention to measurement and analysis.

EVIDENCE OF SIMILARITIES: CMMI[SM] GENERIC PRACTICES

As explained by Ahren et al. (2001), the CMMI[SM] draws a distinction between model components that are required for process improvement (i.e., satisfied goals) and components that are expected to play an essential role as indicators that the required components are in place, and institutionalized as common features of the organization's culture. A *practice* is a statement of an expected component, and may be unique to a single process area (specific practice) of may apply across all process areas (generic practice). In short, generic practices (see Table 4) imply a bridge across the disciplines of software and systems engineering.

Table 4: Generic practices

Generic Practice	Maturity Level	Common Feature
Establish an Organizational Policy	2	Commitment
Establish Requirements and Plan the Process	2	Ability
Provide Resources	2	Ability
Assign Responsibility	2	Ability
Train People	2	Ability
Manage Configurations	2	Directing Implementation
Identify & Involve Relevant Stakeholders	2	Directing Implementation
Monitor & Control the Process	2	Directing Implementation
Objectively Evaluate Adherence	2	Verifying Implementation
Review Status with Higher-Level Management	2	Verifying Implementation
Establish a Defined Process	3	Ability
Collect Improvement Information	3	Directing Implementation
Establish Quality Objectives	4	
Stabilize Subprocess Performance	4	
Ensure Continuous Process Performance	5	
Correct Common Cause of Problems	5	

AVOIDING THE RATING GAME

In many ways, the philosophy of process maturity levels is much like Maslow's hierarchy, which suggests that before one can address higher level needs like self-actualization, the needs from lower in the hierarchy such as food and shelter need to be met. The CMMI[SM] staged approach provides organizations with a very structured approach to becoming a more mature institution. The definition of these maturity levels provides organizations with milestones of achievement. It also allows organizations to establish where they are in the software process improvement continuum. For the past decade, the Software CMM levels provided a structure and gave organizations milestones in the

process of becoming more mature, but also had the unintended effect of creating a competitive rating scale between software organizations. An assessed process may be adequate in one environment but may not suffice for a new project in a different environment (Paulk, 1999). Organizations who use maturity levels to assess contractors run the risk of neglecting many of other factors that would help determine the most appropriate contractor to work on a specific project.

The existence of maturity levels also introduces the risk of organizations setting maturity level goals instead of focusing on improving software/systems to address business goals. With all of the claims of return on investment surrounding process improvement efforts, it is easy to understand why management may strive to reach specific maturity levels for all the wrong reasons. A communication of this concept must be made to all stakeholders who will be affected by the implementation of process improvement. The implementation of a process improvement program needs to be a part of the means to achieve the business goal ends. Businesses can easily become preoccupied in reaching a specific maturity level and forget the ends that they are trying to accomplish. As a result, organizations may end up taking shortcuts in order to be assessed sooner at a certain level even though more attention to a specific process would have been beneficial to the organization in reaching a specific business goal.

In contrast, the systems engineering community adopted an alternative approach to visualizing process improvement: a continuous representation based on individual process areas. The continuous representation evaluates organizations based on capability levels instead of maturity levels. The main difference is that capability levels apply to an organization's process-improvement achievement in individual process areas. These capability levels are 0 (not performed) to 5 (optimizing). Maturity levels apply to an organization's overall process-improvement achievement using the staged model. Using the continuous representation, an organization would have a capability profile, consisting of a list of process areas and their corresponding capability levels.

One of the clear benefits of continuous representation is that it provides organizations with the ability to select the order and grouping of improvement areas that best compliment their business objectives (Shrum, 2000).

Adopting the continuous representation of CMMI[SM] not only forces software organizations to define business goals and choose process areas that should be implemented first to focus on these goals, but it also forces companies who are choosing a new subcontractor to do the same. One of the claimed benefits of a staged representation is that it facilitates comparisons among organizations. (Shrum, 2000) While it may simplify comparisons, it does so at the loss of additional details. Using a continuous representation, the comparison can be done based on the process areas that are judged by the organization as important rather than simply comparing the organization's maturity score. When using a continuous representation, there is less likelihood that organizations will try to attain a specific level without reasons within their business to do so. It provides an incentive to address processes that would have the greatest impact on their business goals.

Since the continuous representation of the CMMI[SM] implicitly encourages organizations to base process improvements on defined business goals, these organizations are less likely to ignore possible improvements outside of the scope of the CMMI[SM] model. If the CMMI[SM] model does not provide them with the means to work toward succeeding on a specific business model, organizations will need to find alternative process improvement methods. Importantly, there are several areas that are completely ignored within the CMMI[SM]. Examples include strategic level processes, such as business strategic planning, architecture definition and strategic planning and control. This group of processes focuses on the adjustments needed over time to meet the changing conditions and requirements of the environment. (Purvis et al., 1999). Since strategic level processes may have a greater impact on the core business goals than improved quality, it may be more beneficial for an organization to focus of these areas before moving forward with CMMI[SM] processes.

THE NEED TO OVERCOME THE DIFFERENCES

Despite anticipated problems, bringing systems engineering best practices into the established software process improvement models is expected to be very beneficial. Boehm (1994) reminds us that an important reason to overcome or bridge these differences is to establish an adequate supply of people who can deal with complex systems problems. The Bureau of Labor Statistics (1997) estimates of anticipated growth in information technology jobs, shown in Table 5, provides further support for this concern.

Table 5: Anticipated employment growth 1996-2006

Type of Job	1996 Employment	2006 Employment	% Change
Database Administrators and Computer Support	212,000	461,000	118%
Computer Engineers	216,000	451,000	109%
Systems Analysts	506,000	1,025,000	103%
Data Processing Equipment Repair	80,000	121,000	52%
Engineering, Science, and Computer Systems *Managers*	343,000	498,000	45%

The final job type, managers, is a significant concern addressed by Jerry Weinberg in an interview (Layman, 2001). Weinberg explains that the software development problems are growing faster than individuals' levels of competence. Moreover, he asserts that the current state of practice is one where we need to apply a few fundamentals (e.g., requirements, reviews, configuration management), that is, things known to be useful, but not adopted in the sense of consistent application.

It has been suggested that the systems engineering—hardware engineering interfaces have matured nicely over many years, but that the systems engineering—software engineering interface is not as mature as the various hardware engineering interfaces.

Meanwhile, the dependency on the systems engineering-software engineering interface has increased faster than it has matured. Specific concerns by discipline include:

The State of Systems Engineering

- Most successful projects rely on expertise established with similar systems.
- Lack of documented processes makes repeatability difficult.
- Development efforts for unprecedented or significantly different systems often encounter problems.

The State of Software Engineering

- The brief history of software development has been filled with problems of cost overruns, schedule slippage, and failure to achieve performance goals.
- Systems are increasingly dependent on software, yet hardware typically gets the most visibility.

Although, many software-only organizations remain adamant that they do not do systems engineering, all software must run on some computer system, and interface with others. This perceived separation of concerns exacerbates the difficulties associated with hardware/software/system tradeoff decisions, which are further complicated by terminology differences and disparate mental models.

However, the integration potential of the CMMI[SM] can allow the system and software engineering communities to get the most out of their similarities. The CMMI[SM] allows organizations to tailor the model to mesh with their own mission and goal statements as well as their business objectives. Each individual project can use CMMI[SM] models for individual disciplines and discipline combinations because the architecture of the CMMI[SM] does not force the employment of every discipline for every organization implementing it. Before the CMMI[SM], the systems engineering models shared many of the same principles as the software version of CMM, but were written to address

the needs and terminology of the systems engineering community. Because the CMMI[SM] includes the common and shared elements and best features of both software and system engineering together with discipline specific elements, an organization can generate integrated capability maturity models or discipline specific capability models. With CMMI[SM], an organization can still capitalize on these similarities and improve the efficiency of and the return on investment for process improvement. The resulting integrated capability models will adapt to an organization's business purposes.

CONCLUSION

The ongoing process improvement efforts centered on integration of software and systems engineering, as initiated by the SEI's CMMI[SM] project, has highlighted two key issues for researchers and practitioners:

1) A renewed focus on products and business objectives as drivers of process improvement;
2) Opportunities for high-leverage process improvements.

The concept of an architecture continues to serve as a theoretical link for both the software/system tradeoffs and the integration of process improvement efforts. While respecting the legitimate differences in areas such as reliability testing, it is important to sustain the hope that overlapping or underlying theories will emerge regarding areas of common concern such as: requirements, security, safety, and performance.

In order to achieve true integration of software and system engineering practices into one process improvement model, the remaining differences of terminology and model construction have to be addressed. These two communities have well-developed disparate languages and methodologies that are reflected in their different models, and entrenched in their organizational cultures. With the adoption of an integrated process improvement model, an organization can assess both software and systems engineering functions.

REFERENCES

Ahern, Dennis, Richard Turner, and Aaron Clouse. (2001) CMMI(SM) Distilled: A Practical Introduction to Integrated Process Improvement. Boston: Addison-Wesley.

Boehm, Barry. (1994) "Integrating Software Engineering and System Engineering". The Journal of INCOSE, Volume I, Number I: July – September

Layman, Beth. (2001). "An Interview With Jerry Weinberg." Software Quality Professional, 3:4 (September), pp. 6-11.

Paulk, M. C., Weber, C. V., & Chrissis, M.B. (1999). The Capability Maturity Model for software. In K. El Emam & N. H. Madhavji (Eds.), Elements of software process assessment & improvement (pp. 3-22). Los Alamitos, CA: IEEE Computer Society.

Purvis, R. L., Santiago, J., & Sambamurthy, V. (1999). An analysis of excluding IS processes in the Capability Maturity Model and their potential impact. (pp. 31-46). Idea Group Publishing.

Rassa, Bob "Beyond CMMI-SE/SW v1.0". Software Engineering Institute, March 13. 2001. http://www.sei.cmu.edu/cmmi/publications/sepg01.presentations/beyond.pdf

Software Engineering Institute. "Concept of Operations for the CMMI". *Software Engineering Institute*, January 15, 2001 http://www.sei.cmu.edu/cmmi/org-docs/conops.html.

Software Engineering Institute. "CMMI Frequently Asked Questions." *Software Engineering Institute*, March 2001. http://www.sei.cmu.edu/cmmi/comm/cmmi-faq.html

Software Engineering Institute. "Transitioning Your Organization from Software CMM Version 1.1 to CMMI-SW Version 1.0". *Software Engineering Institute*. http://www.sei.cmu.edu/cmmi/publications/white-paper.html (August 1, 2001).

Shrum, Sandy. "Choosing a CMMI Model Representation." *SEI Interactive*, December 1999. http://www.stsc.hill.af.mil/crosstalk/2000/jul/shrum.asp (July 17, 2001).