SPMN Director Identifies 16 Critical Software Practices

Michael W. Evans
Integrated Computer Engineering Inc.

There are two distinct stages in improving a software process. The first stage is defining the optimum processes that can be applied successfully across multiple projects; to this end, tactical software processes have been developed. The second stage is embedding these processes in the project cultures that must apply them.

Author and Atlantic Systems Guild Principal Tom DeMarco makes an important observation in his book *Why Does Software Cost So Much* [1]. Instead of asking, "Why does software cost so much?" he says that we need to begin asking, "What have we done to make it possible for today's software to cost so little?" Like the question, the response seems to be "very little."

We as an industry seem to search in vain for the magic silver bullet—the right combination of methods and tools that would make predictable software cost, schedule, and quality performance a reality across the industry without a significant cost or schedule expenditure. Program managers, company presidents, controllers, and customers are continually surprised and disappointed when the magic does not appear.

Norm Brown, director of the U.S. Navy's Software Program Managers Network (SPMN), has recognized that, at least in the Department of Defense (DoD) and probably in other segments of the economy, the silver bullet is not now, or never will be a reality. Together with software industry leaders who meet regularly as part of the Airlie Software Council of 13, he has documented first a set of nine best practices, and now 16 Critical Software Practices for Performance-Based Management™, which appear to be common threads running through successful software projects. These 16 practices provide managers with specific methods that directly affect the bottom-line cost, schedule, quality, and user satisfaction metrics. Without exception, these practices may be implemented in less than a year.

Developers realize that implementation of the 16 Critical Software Practices requires the adaptation of each individual practice to each project, as well as cultural acceptance by the team who will do the work. As Tim Lister has pointed out, "Common processes represent 10 percent of the problem, adaptation is 90 percent [2]." The complexity of successful adaptation is as much due to cultural insertion as it is to tailoring to project needs and realities.

This paper discusses issues addressed by the 16 Critical Software Practices for Performance-Based Management, the rationale behind some key practices, and what can be expected when bringing the practices into a company culture or project environment.

The Basic Problem

As R.A. Radice and R. Philips point out in *Software Engineering: An Industrial Approach*, "We are still at the beginnings of becoming a science. We call ourselves computer scientists or software engineers, but it is more out of anticipation of what these roles offer than from a fully earned position. We, as an industry, still do not keep, analyze, or make public the necessary data to substantially prove our theories or to enable others to repeat our successes … We still cannot do as good a job on a new project as we did on the last [3]."

A reality since 1943, the software profession has had more than half a century to mature. However, since 1975, when *The Mudd Report* [4] was published, the DoD software community has been in a state of almost continuous crisis. Its attempts—without apparent result—to resolve crises through new technology, management practices, tools, or other actions have, at best, not improved the problems and have, at worst, compounded the effects. Reality is that in 1995, an estimated 53 percent of software projects cost nearly 190 percent of their original estimates, 31 percent of software projects were canceled before completion, and an estimated $81 billion was spent for canceled software projects by American companies and government agencies [5]. Apparently, we do not practice effective crisis management.

In an attempt to resolve the software crisis, DoD established the Software Engineering Institute (SEI) in 1984 as a federally funded research and development center at Carnegie Mellon University. SEI's mission is to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. The SEI vision is to bring the engineering discipline to software development and maintenance.

The Strategic Improvement Model

A key component of the SEI strategy is to establish a process to facilitate continuous process improvement within the software community. In SEI's view, continuous process improvement is based on many small, evolutionary steps rather than evolutionary innovations. The Capability Maturity Model® (CMM) provides a framework for organizing these evolutionary steps into five maturity levels that lay in order successive foundations for continuous process improvement [6].

In my view, the CMM has caused software providers to refocus on the benefits of, and the essential steps required for, improving software-engineering processes. Advancing up the CMM ladder is an essential strategy that many organizations follow to improve their software processes and enhance their image as serious software engineering contractors. CMM improvement requires significant time (12 to 18 months) and a significant commitment of resources to move up one level. Significant improvements in bottom-line metrics can be expected as new levels are achieved in projects where improvements have been applied (this does not include all organizations within the company).

However, this leads us to a dilemma. How do we guard against cost and schedule impacts, quality shortfalls, and user dis-

satisfaction with software systems produced or maintained by non-assessed projects or organizations? And how do we address the many projects and organizations that are just starting their climb up the CMM ladder? Current data show that for a total of 901 organizations assessed for SEI CMM implementation, 34.9 percent are at Level 1, 38.2 percent at Level 2, 18.5 percent at Level 3, 5.5 percent at Level 4 and 2.9 percent at Level 5.

A second problem in using CMM solely as a means to improve project performance is that the interpretation of the CMM focus is strategic in nature. It does not implement the tactical guidance required by individual projects to improve their performance to achieve immediate goals. Achievement of immediate goals may mean the difference between project continuity and project extinction.

Prior to further discussion on strategic versus tactical project implementation, it is important to define the two terms. Strategic processes are those of importance to the integrated whole or to the overall planned effect. While tactical processes are smaller in scale, serve the larger purpose when performed collectively, and are carried out with an immediate end in sight.

It is clear that the application and harmonious coexistence of both strategic and tactical processes will achieve overall project success.

CMM's five levels are a model for improving the software organization's capability. The CMM priorities, as expressed by these levels, do not address the root cause of problems faced by individual projects, nor do they point to remedies that are under the direct control of the project leader.

A troubled project's solutions might be of limited value to the rest of the organization. Other projects might have different problems or be unable to take advantage of its solutions because they lack the necessary foundation to implement the solutions [7].

There are two distinct stages in improving a software process. The first stage is defining the optimum processes that can be applied successfully across multiple projects. The second stage is embedding the process in the project cultures that must apply it. The process identification approach is fun and rewarding, while achieving cultural acceptance is tedious, frustrating, and difficult to accomplish. The truth is that if these improved common processes do not find their way into the project culture, project improvement of bottom-line metrics becomes a myth.

For example, a recently visited organization spent significant effort achieving CMM Level 3. They documented common processes, modifying them as necessary, and formed a very active software engineering process group (SEPG) composed of the "best software people in the company." The most frequent comment from projects was that the SEPG lived in an ivory tower. Management discounted the comment as *sour grapes* by a group of engineers who did not understand the real need to improve process.

As this organization moved up the CMM ladder, it became increasingly obvious that the expected improvements in development cost, schedule, product quality, and relationships with various user communities were not being realized and were, in some cases, getting worse. What was going on?

As the SEPG tried to force change on ongoing software projects, they were being *gamed.* The projects declared they had implemented the required key process areas (KPAs) but had segregated the project into two distinct teams—development and systems. The development team included all the software-related activities prior to delivery to the systems team, which integrated the products and delivered them to the customers. To minimize impact on the development culture, the KPAs were only applied to the products when they were released to the system team. Company management felt that directly impacting the development teams incurred too much risk.

I wish this example were the exception, but in my experience it is not. Too many organizations focus on the obvious recognition and marketing advantage that a higher CMM rating confers, without recognizing that the reason for pursuing continuous improvement is to improve bottom-line metrics, thereby improving the quality of products delivered within planned timeframes and within budget.

The strategic process role is to establish an environment within an organization that actively promotes, finances, and supports applying improved best practice initiatives. The strategic process should not attempt to precisely detail what steps need to be taken at any instant in time by each respective member of the project team. The strategic process should be tailor-made to specific project scenarios. It should give latitude to the project team to apply applicable solutions to specific project problems. In other words, the strategic process must allow the project team tactical freedom to achieve project objectives.

The Tactical Improvement Model

Brown and the SPMN recognized the need to develop a tactical model for software process improvement. SPMN is a grassroots organization of software managers formed by Brown in 1992 to share lessons learned and improve the bottom-line metrics of cost, schedule, quality, and user satisfaction on software projects.

Far too many large-scale software projects have become unaffordable and unable to deliver needed quality, reliability, and capability within the required time frame. Their outputs are not predictable. Their processes are little more than chaotic and do not effectively utilize the kinds of disciplines necessary to achieve success. They have not yet taken advantage of the types of practices used to effectively manage large-scale hardware projects [8].

From its inception, SPMN has focused on tactical issues and practical solutions that have been proven in industry and that focus on project rather than organizational issues. As part of its charter, SPMN has been a major identifier of software acquisition best practices both within and outside the DoD.

The [1995] Software Acquisition Best Practices Initiative [performed by the SPMN] was established to bring about substantial improvements in productivity, quality, timeliness, and user satisfaction by implementing best practices as a new foundation for DoD software management. Two purposes of the initiative include focusing the defense acquisition community on employing effective high-leverage software acquisition management practices, and enabling managers to exercise flexibility in implementing practices within disparate program cultures. The initiative is intended to influence both government software program managers and their industry counterparts.

Solutions are taken from successful programs' practices. When effectively implemented and given competent staff these

practices help bring order, predictability, and higher levels of productivity and quality. Each one includes key applicability factors enabling adaptation to particular situations and environments.

These practices are focused upon effective management processes, techniques for finding defects as they occur, eliminating excessive and unnecessary costs, increasing productivity, and other beneficial effects. The Airlie Software Council and other industry experts and consultants are convinced that projects effectively utilizing the Software Acquisition Best Practices and other appropriate best practices will achieve significant cost reductions while simultaneously increasing quality and reliability [9].

Originally SPMN defined nine practices as being essential elements of a successful project. The original nine practices were applicable to all large-scale projects (i.e., projects relying on the full-time efforts of 12 or more people annually). Observably best practices should appropriately be used according to the particular circumstances and environment of a given project. From the outset, SPMN recognized that the practices, as with any best practice, are of little value unless they are adapted to specific project environments and cultures and are embraced by those who must implement them.

The software development process consists of many components that must fit together to create a total and integrated project environment. These individual project pieces must interface and interact efficiently within other project segments if the project is to function efficiently [10].

Concern about internal project consistency and practicality of recommended practices in light of project realities and constraints has always been a SPMN concern when recommending practices. SPMN practices are termed *best,* not because they have been intensively studied and analytically proven to be best, but simply because they are practices used by and considered critical to successful software projects. SPMN does not feel that the original list of nine and the current set of 16 practices are presumed to be best; nor that there may not be other, perhaps even better, practices. The practices that comprise the 16 critical software practices, when implemented in projects, will go far toward engendering successful software development and maintenance.

Before engineers can work effectively in an integrated team environment, they need to know precisely what to do. Teams can waste a great deal of time trying to establish goals, resolve their working relationships, and determine what to do [11]. SPMN's 16 critical software practices provide specific guidance as to which practices have been proven to work on similar projects, and how they can be implemented within a specific project.

As illustrated in Figure 1, the 16 critical software practices address three primary areas of software management: project integrity, construction integrity, and product stability and integrity. Project integrity includes practices that identify basic project constraints, requirements, and expectations. It also encompasses planning and implementing practices for a project environment to predictably satisfy them. Construction integrity comprises those activities that specify the basic product requirements, maintain traceability to these basic requirements, plan and control content and change, and ensure that all components of the project communicate. Product stability and integrity ensures that defects, which are inserted in products as part of the software process, are identified and removed in a timely fashion. It also ensures that testing is complete and effective and results in the right product consistent with the agreed-to requirements and actual expectations.

Although these 16 practices are useful individually, their complementary nature provides a strong synergistic effect when used as an integrated set. Using them will not guarantee success, but they can help facilitate it and avoid failure. Those familiar with process improvement models such as CMM will quickly realize that these practices supply tactical solutions that compliment the model's strategic orientation. The practices map to many of the model's KPAs, and should assist organizations striving to advance to the next CMM maturity level.

SPMN has developed a software evaluation model (SEM) that is based on the 16 critical software practices [12]. The SEM provides for each of the critical practices, practice essential elements, implementation guidelines, and a detailed question set to assist in implementation. In addition to the SEM, a mapping matrix is available that maps each of the critical practices with SEI's CMM Level 2 and 3 activities.

These practices are not rocket science. They can be readily implemented. Although some practices may require training in basic skills such as conducting effective meetings as a necessary foundation for formal inspections, for the most part they can be implemented without making investments in new equipment, technologies, or staff.

Figure 1. *16 Critical Software Practices for Performance-Based Management*

Project Integrity

- Adopt continuous risk management.
- Estimate cost and schedule empirically.
- Use metrics to manage.
- Track earned value.
- Track defects against quality targets.
- Treat people as the most important resource.

Construction Integrity

- Adopt life cycle configuration management.
- Manage and trace requirements.
- Use system-based software design.
- Ensure data and database interoperability.
- Define and control interfaces.
- Design twice, code once.
- Assess reuse risks and costs.

Product Stability, Integrity

- Inspect requirements and design.
- Manage testing as a continuous process.
- Compile and smoke test frequently.

Successful Implementation

More than 200 risk assessments conducted during the past five years along with effective risk mitigation advice to software developers and government software program management offices has proven that the 16 critical software practices are a successful tactical software project process

Typical of the success stories is one in-process program that suffered from being over budget, late in delivery, and producing product that failed to meet specified requirements. This program's management recognized that it was in crisis when a major system it was preparing for delivery failed to meet its operational evaluation. The program manger initiated a problem analysis that identified three contributing factors. First, the program team was continually reacting to crises and never had time to plan ahead and anticipate problems. Second, there was no focus on success or failure accountability in the program office, nor did this accountability flow down to the contract suppliers. Third, there was no process in place that would ensure that a quality software product was fielded.

To address crisis management and effectual program control, an effective risk management process was implemented. To anticipate actions to be taken in managing the program, the program office utilized risk management process and risk management tools. This was not easy culturally because risk prediction was associated with bad news and failure. In addition contractors had signed up for unrealistic tasks, milestones, technical commitments, and delivery agreements that rightfully pointed to gloomy risk assessment outcomes. Crisis management rather than effective risk management occurred because of the significant overcommitment of resources.

All project and supplier management staff were provided with risk management training, while the program office was given expert assistance to help develop and implement risk policies and plans. The risk training program focused on both procedural risk issues and problems with cultural acceptance of the risk process within the program. To track individual risks, risk management experts assisted program staff in identifying risks and seeding them into an appropriate risk management tool. Risk experts assisted the program staff in identifying what could go wrong and mapping the road ahead.

Risks were assessed on the basis of their impact and likelihood of occurrence. Risks with high impact and high likelihood of occurrence were given the highest level of scrutiny and attention. A monthly risk reporting process was implemented that initiated corrective action taken by the program manager. After implementing the risk management process, the program took an about turn. It shifted from unsuccessful crises management to a process that identified potential problems early and permitted effective corrective action well before disastrous or costly failures had occurred.

Setting and enforcing clear goals addressed the second problem of accountability. Goals included setting quantitative targets that could be measured. Progress reports that described the progress toward assigned goals were required on a regular basis. Five targets were established:

- Bring the reliability of all developed systems to a specified number of hours within a 12-month period.
- Deliver all future products within cost and schedule.
- Have all contractors' risk management process in place and compliant with the risk management plan within six months.
- Deliver software that is supported by adequate documentation.
- Deliver software that meets user requirements.

Finally, to address software process development standardization, training was provided to program staff on the 16 critical software practices. A practice assessment of all the program suppliers was performed, and each supplier was evaluated by practice area and rated on a five-point scale (one being the practice was nonexistent and five being the procedure was in place in the culture of the organization).

The first assessment scores averaged 1.8. By the third assessment the average score was greater than four. The established process improvement was evident in the measured metrics of cost, schedule, quality, and user satisfaction. The program made great strides in software process improvement during a one-year period. The program manager rated the cost of installing the improved process as extremely cost effective (approximately 1 percent to 2 percent of the program cost).

Program suppliers measured success not only by product quality, but also by adherence to set program goals and their degree of implementing the 16 critical software practices. The program progressed from having consistent software acquisition problems to a competent software acquisition organization with a reputation for delivering quality product, on time, and within budget.u

References

1. DeMarco, Tom, *Why Does Software Cost So Much?* New York: Dorsett House Publishing Co., 1995.
2. Lister, Timothy, Software Management for Adults, Salt Lake City: Software Technology Conference, 1996.
3. Radice, R.A. and Philips R., *Software Engineering, An Industrial Approach*; Prentice Hall, 1988, pp. 2-3.
4. Weiss, David, *The Mudd Report: A Case Study of Navy Software Development Practices,* Washington, D.C.: Naval Research Laboratory, May 21, 1975.
5. Standish Group International, Chaos, *Open Computing*, March 1995.
6. Paulk, Mark C. et al., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Version 1.1, Reading, Mass., Addison-Wesley, pp. 15-17.
7. Ibid.
8. Software Program Managers Network, *The Program Manager's Guide to Software Acquisition Best Practices,* V. 2.2, Arlington, Va., SPMN, 1998, IV.
9. Ibid.
10. Evans, Michael W. and Marciniak, J., *Software Quality: Management and Assurance*, New York: John Wiley & Sons, 1987, p. 20.
11. Webb, David and Humphrey, Watts, Using the TSP on the TASKVIEW Project, CROSSTALK, February 1999.
12. SPMN, Software Evaluation Model (SEM), version 5.3.1, May 2000 [www.spmn.com].

About the Author

Michael W. Evans is president of Integrated Computer Engineering Inc. He is experienced in providing direct technical services and support in software engineering methods and processes, software standards, quality assurance, configuration management, and testing. He is cofounder and prime contractor for the Software Program Managers Network, the driving force behind the Department of Defense's Software Acquisition Best Practices Initiative. Evans is the author of *Principles of Productive Software Management, Productive Test Management, Software Quality Assurance and Management*, and *The Software Factory*.

ICE Inc.
142 North Central Avenue
Campbell, Calif. 95008
E-mail: candca@aol.com
Internet: www.iceincUSA.com

Get Your Free Subscription

Fill out and send us this form.
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056
Fax: 801-777-8069 DSN: 777-8069
Voice: 801-775-5555 DSN: 775-5555
Or request online at www.stsc.hill.af.mil

NAME:_____

RANK/ GRADE:_____

POSITION/ TITLE:_____

ORGANIZATION:_____

ADDRESS:_____

BASE/ CITY:_____

STATE:_____ ZIP:_____

VOICE: _____

FAX:_____

E-MAIL: _____@_____

CHECKBOX(ES) TO REQUEST BACK ISSUES:

JUL 2000____ CMMI

AUG 2000____ PROCESS IMPROVEMENT

SEP 2000____ COTS

OCT 2000____ NETWORK SECURITY

NOV 2000____ SOFTWARE ACQUISITION

DEC 2000____ PROJECT MANAGEMENT

JAN 2001____ MODELING/ SIMULATION

FEB 2001____ MEASUREMENT

How Process People View Life

"You want Daddy to read you a bedtime story? How about *Goldilocks and the Three Bears?"*

Once upon a time [What accuracy of time? How many digits of precision are needed? Is real-time really necessary here?] there was a little girl named Goldilocks [Why Gold? Did the customers request a specific color this early in the design phase?] who packed a picnic basket to go visit her grandmother. [What are the grandmothers' dietary requirements? What are her nutritional needs? Are there any food allergies?] Goldilocks set off through the forest to grandma's house [Why did she set off so quickly? What life-cycle model is she using that suggested implementation so early? Did she have a prior visit to base her projected arrival date upon?] to deliver the basket. [Was Goldilocks goal-oriented and quality-driven? There seems to be no evidence of an enabling infrastructure to help her self-actualize and fulfill her needs. Does Goldilocks need a nurturing work environment to be truly productive?]

At the same time, there lived three bears in the forest. [Same time? Is this a potential concurrent-processing problem? Are we going to have to worry about parallelism? Does our design methodology support real-time interfacing? We should probably use Ada for implementation.] The three bears were daddy bear, mommy bear, and little baby bear. [Is this a well designed, object-oriented system? Is inheritance correctly used?] Mommy bear had just prepared porridge [Was this on her list of deliverables for the current milestone?] and the family sat down to eat. Unfortunately the porridge was too hot, so they decided to take a walk to let it cool. [Is this interruption necessary? Once the team loses focus, it is difficult to recapture a synergistic mindset. Couldn't other team goals be worked on to ensure the team stays cohesive? Why didn't mommy bear perform a personal review prior to conducting a peer review?]

While the bears were out walking, Goldilocks wandered upon the house and, lost and hungry, smelled breakfast and decided to eat. [What kinds of protection scheme for critical resources are in place? Doesn't Goldilocks have both milestones and inch-pebble deliverables to keep her on track?] The first bowl of porridge was too hot, the second too cold, and the third was just right. [Good fence-post testing techniques. However, shouldn't independent verification and validation be contracted to ensure that her testing parameters were realistic and customer-oriented?] After eating, she went upstairs for a nap. [Good idea. Studies show that neural activity begins to decline after sustained exertion. Rest breaks promote quality and reduce rework.] Upstairs, Goldilocks found three beds. The first bed was too hard, the second too soft, and the third was just right. [Is she qualified to test both porridge and beds? Seems like a reusable test case, but is it correctly tailored and parameterized for beds? A dedicated test team might be needed, with both porridge and bed domain experts available] She laid down to rest.

While she was sleeping, the bears returned. Daddy bear found his porridge, mommy bear found her porridge, but baby bear discovered that his porridge was all gone! [Poor allocation of resources. Is the work breakdown structure set up to handle dynamic reallocation so that baby bear will not be the critical path?] Suspecting something was wrong, all three bears went upstairs. Daddy bear found his bed disturbed, as did mommy bear. Baby bear, however, found someone sleeping in his bed. [This is the second time baby bear has found serious defects in his personal deliverables that effect other team members' schedules. If this trend continues, we might have to assign baby bear some additional quality training. In the worst case, we might have to bring in a better team player.]

Hearing the commotion, Goldilocks awoke with a start, and ran down the stairs, out the door, and straight to her grandma's house. [When the bears saw the problem, did they take proactive steps to prevent a future occurrence? Are they using root-cause analysis to fix the problem, rather than just fixing the symptom? More importantly, if Goldilocks knew the way under pressure, was she just slacking off early in the project? Were sufficient key process areas in place to ensure her original schedule was realistic? Was she revising her schedule during each iteration of the life-cycle model? She wasn't self-assessed, was she?]

Along the way to her grandma's house, Goldilocks ran into the big bad wolf. The wolf convinced Goldilocks to accept work for a competing grandmother, at a higher salary with better benefits. Unfortunately, the basket to the original grandmother was delivered over-budget and behind schedule. Luckily, a source of continuing funding exists, so errors are still being fixed in Grandma's Basket release version 4.3a. [Gosh, I just LOVE a happy ending!]

"What, honey? What do you mean, you want Mother to read to you from now on?

– David Cook, Shim Enterprise Inc.