How Scalable Are CMM Key Practices?

Rita Hadden
Project Performance Corporation

Many software practitioners are convinced that cost outweighs benefit when the Software Engineering Institute Capability Maturity Model's (CMM) key practices are applied to small projects. Practitioners believe that only the complexity that typically comes with project size justifies the investment. This article is based on observations and experience with over 50 small projects. It describes using a repeatable and disciplined approach for system development efforts in spite of short durations. It illustrates the use of professional judgment to appropriately scale down and apply CMM key practices to make a difference in the outcome of small software projects.

Many software professionals believe that the key practices of the Software Engineering Institute (SEI) CMM are inappropriate and not scalable for small software projects (three to four months with five or fewer staff members). These practitioners are convinced that the costs outweigh the benefits unless the complexity that typically comes with project size justifies the investment.

Some professionals also believe that customer satisfaction is their most important measure of performance, meaning that to accept one more requirement is almost always "the right thing to do," no matter how close the production or release date might be. They are convinced that in the "real world," requirements cannot be frozen or "baselined" because there are too many legitimate reasons for requirements to change, and new ones must be added to the current version of the software.

Many developers frequently do not see the benefits of managing requirements, nor do they feel the need to take time for project planning, tracking, and oversight, as advocated by the CMM. They do not understand the payback from a software quality assurance function or configuration management. I recently worked with several managers and practitioners who held these views.

No Defined Requirements But a Set End Date — An Example

The project was to develop a software system that provides a consistent estimation approach and addresses software size, effort, and assumptions for planning. The system's goal was to improve the client organization's software estimation process. The organization's overall strategy was to use the SEI CMM Repeatable level (Level 2) key practices to move toward a more disciplined software development environment. When I started, no requirements had been defined for this estimation system but an end date had been set—the system had to be operational in two months.

The client was an information systems business unit in a large company. A formal CMM-based software process assessment conducted a month prior to my arrival found that this organization's estimation process for software change requests was ad hoc and inconsistent. This estimation process frequently produced understated effort estimates, perhaps due to institutional pressure to get things done faster or a desire to minimize what is involved. In turn, these understated estimates often led to budget overruns, crisis management, eleventh-hour firefighting, low employee morale, and high rates of post-release defects.

My immediate clients, all information systems professionals, had been working on improving their software management practices for over 18 months. Their corporate direction was to use the CMM to guide their process improvement. Some of them had been promoting the CMM for close to two years. The sponsor of our software project was the head of the Software Engineering Process Group. The project team consisted mostly of client personnel to the project part time, and three full-time outside consultants: a junior analyst, a Visual Basic programmer, and me.

Getting Buy-In

A fellow team member and I began the project by meeting with our sponsor and key stakeholders to define the business objectives, scope, and constraints of the estimation system. After reviewing existing relevant documentation, we developed a high-level project plan. It was an ideal opportunity to apply the CMM key practices to our day-to-day project work. In particular, I believed the management and technical practices from Levels 2 and 3 of the CMM were appropriate. These included requirements management; project planning, tracking, and oversight; quality assurance (QA); configuration management (CM); software product engineering; and peer reviews. I recommended to our clients that we apply scaled-down CMM key practices to meet the needs of our estimation system project.

"The system must be finished in two months. We need to start coding right away!" was my sponsor and other team members' reply.

I pointed out that automating a more disciplined software estimation process with a structured and repeatable approach would not only assist their

overall process improvement effort but also would help the current project meet its pressing deadline as well.

As far as I could tell, I had been brought into this project for three reasons. First, I had earned the trust and respect of some of the key players in this organization by contributing to a previous process improvement project. Second, I was a seasoned software engineer and manager with over 70 software projects from which to draw. Finally, I was certified by the SEI as a lead software process assessor.

My sponsor agreed to let me apply some of the CMM key practices on the grounds that they would legitimize the product we were developing. I knew I was going to have to make believers out of these skeptics.

Turning Expectations into Requirements

Our stakeholders were software project managers, team leaders, functional subject matter experts called "evaluators," software practitioners who performed estimation called "estimators," and coordinators of the many software change requests. These stakeholders were accountable for the effectiveness and efficiency of this organization's estimating process.

My team of one analyst, one developer, and two part-time client functional experts had to first come to consensus on "what is a requirement?" We agreed that a requirement is a functional or technical capability needed to solve a problem or achieve an objective. A good requirement is traceable to business objectives and related system lifecycle components. It is consistent with the scope and constraints of the product, incorporates stakeholder expectations, is measurable against acceptance criteria, and is maintainable over the product's lifecycle.

To expedite the requirements-gathering phase, my team developed a set of straw man functional expectations. These expectations were based on our documentation review, our understanding of the organization's estimation process and user desires, our knowledge of industry best practices, and the objectives, scope, constraints, and assumptions of the estimation system.

Next, we used the straw man set of expectations and Joint Application Design techniques to elicit additional input from our stakeholders. We held focus groups and one-on-one interviews. We then analyzed all gathered expectations and developed a manageable set of functional requirements. Where necessary, we consolidated similar expectations and identified and resolved conflicts between expectations.

Explicit vs. Implicit Requirements

My team then supplemented these "explicit" requirements with "implicit" technical requirements. Implicit requirements include stakeholder expectations that were not articulated but are essential to develop a product that achieves exceptional user satisfaction. For example, we ensured that the system's response time and maximum number of concurrent users were stated as measurable requirements. We also addressed usability, availability, security, maintainability, and portability of the estimation system.

We documented all requirements and validated them with all stakeholders. We requested that our key stakeholders review and sign off on the Statement of Requirements for the estimation system, then baselined these requirements. We obtained the sign-off three weeks after the project start.

Building In Quality

Based on the system requirements, we next developed a conceptual design that we revalidated with our stakeholders in small walk-through sessions where feedback and ideas were solicited. This conceptual design was informally documented in two days. Three concurrent efforts followed: one to create a detailed software development plan (SDP), a second to design and prototype user interface data entry screens and system navigation using a Rapid Application Development, and a third to design database structures and define data validation criteria. We involved our stakeholders throughout the process.

Using Requirements to Drive the Software Size Estimate

We now had what we needed to get a relatively reliable estimate of the size of the estimation system. We used function points and work breakdown structures to estimate size. The results from these methods took less than two days to produce and were remarkably close (4.3 staff months for the construction phase using work breakdown structures vs. 4.7 using function points). We met with senior management and presented our estimates of size, effort, cost, schedule, critical computer resources, and what it would take to do the project "right the first time." Faced with the detailed planning data, senior management had to choose among cutting scope, changing the schedule, or adding more people to the project. They agreed to allocate more resources and time: one programmer for four weeks and one additional elapsed month.

The SDP also contained a detailed task plan, test plan, QA plan, CM plan, measurement plan, and risk management plan. Because this was a small project (2,275 hours, [3.5 months]), we scaled our SDP accordingly to 28 pages. To save time, we rewrote an existing QA plan to meet our needs.

Involving Stakeholders and Controlling Requirement Changes

Five weeks into the project, our designs and prototype were ready for a walk-through with our stakeholders. Many issues and concerns surfaced at this session that we had to address. In addition, many requirements changes were proposed following the walk-through. We listened to each rationale for change and recorded the request. Where a change improved usability and the effort was negligible, we included it in our revised design. Dealing with these challenges upfront minimized rework for us later.

We resisted the temptation to add a new feature for every stakeholder concern. We presented the total requested change in scope and its associated impact on schedule, resource, and cost to the client senior management and Software Configuration Control Board. It

would have taken 31 additional staff days to incorporate all the requested changes.

The stakeholders decided to revisit the change requests in a subsequent release of the system. Our requirement control approach helped us gain buy-in and credibility from all project participants.

Paving the Way for Culture Change

Because senior management recognized that this project would require many people to fundamentally change the way they perform software estimation, we wanted to pave the way for this culture change. To meet this objective, we deployed a set of spreadsheets that emulated the new estimation process six weeks prior to piloting the new estimation system. With these spreadsheets, we trained future users of the estimation process in the "whys" and "hows" of the new procedures.

We left the detailed design of the reporting requirements until last. Key stakeholders were invited to sit by our side as we designed the reports and queries that contributed to their job effectiveness. We also requested stakeholder participation and feedback in our system testing to fine-tune our system. Meanwhile, we developed training materials and user documentation that emphasize audience participation and usability. Later, we provided multiple training sessions that gave participants hands-on system experience.

Tracing Test Cases to Requirements

As soon as we received approval for our design, we began to define test cases and expected results for each of the system functions. We had limited time for testing, so we wanted to ensure each test case was traceable back to one or more requirements and that our product met the stated requirements.

We then created a test database and seeded it based on our test cases. As each software module was completed, we performed functional testing using the test cases and seeded database. We compared actual results against our documented expected results for each test case. At the end of each testing session, we reviewed our test results with the responsible software developer. Hundreds of defects were uncovered early enough to correct with minimum effort. Some modules were particularly problematic and had to undergo several iterations of functional testing.

Keeping Our Project on Track

Our team met weekly to discuss status, issues, and required actions. I worked continuously to mitigate the risks to project success. These risks included a language barrier between the two programmers, since one spoke little English; insufficient face-to-face communication with users of the system, which could result in misunderstanding of the new estimation process; performance and usability issues that could cause stakeholders to not fully accept the product; delays in functional testing due to defects, which could jeopardize the start of system test and overall project schedule.

We also met periodically with the client organization's QA, CM, and senior management. QA's role was to review and monitor our design and development activities to verify compliance with the organization's standards and procedures. CM's function was to maintain the integrity of our product through configuration identification, change control, status accounting, and audit. CM also ensured that all requested changes to the functionality of the estimation system were reviewed by senior management and the Software Configuration Control Board.

All this oversight and coordination set the stage for an extremely smooth pilot. This pilot lasted a week and involved a dozen users working hard to exercise all aspects of the estimation system. These users entered "real-world" data from recent estimates they had prepared. They also tested the system's user's guide. Based on their feedback, we enhanced the user's guide to include exception handling. No rework was required on the system software.

The estimation system was accepted with enthusiasm and went into full production without delay. No significant defects had been identified to date six months following the rollout.

CMM Key Practices Are Scalable!

We held a feedback session to develop lessons learned with members of our team when the project was over. To our surprise, we heard that this was the *first* time that a PC-based system developed by this group had been delivered on time, within budget, and with satisfied stakeholders. Even the Visual Basic programmers admitted that a managed set of requirements, a documented design, early and frequent user involvement, and disciplined approach to testing contributed to the success of the project.

These practitioners' beliefs that design was "paperwork," that there was no time for walk-throughs, and that ad hoc testing was sufficient were beginning to change. Their conviction that requirements could not be baselined was also becoming less absolute. We had given them an alternate way to approach software development—we had made the CMM come alive for them!

Looking back, I feel tremendous satisfaction for sticking to a repeatable and disciplined approach for our system development effort in spite of its short duration. I can imagine the outcome of this project had we given in to pressure to start coding at the beginning of the project.

Using professional judgment to appropriately scale down and apply CMM key practices instead of using "code and load" did make a difference. Managing requirements helped us control "scope creep" and keep our stakeholders aware of the trade-offs, allowing them to make informed decisions. Using a disciplined process helped us discover defects prior to testing, minimize rework, and reduce post-release defects. We delivered a better product on

see CMM, page 23

cases are tested in the courts, no one knows how much protection software disclaimers will afford.

Article 2B

There is one more interesting development that has occurred: a draft of Article 2B of the Uniform Commercial Code (UCC) (which pertains to computers and computer services) was released Nov. 1, 1997 [2]. Article 2B will play an important role in defining software warranties. Article 2B will only serve as a model template, and each state in the United States will be responsible to modify it to their standards before adopting it as law. Further, Article 2B has the potential to relax the liability concerns that might force an ISV to use a certification laboratory. This could turn out to be a disaster for those parties most concerned with software quality.

Conclusion

Before we can determine what role SCLs will play in software liability, we must wait for more cases to be tested in court to see to what standard of professionalism ISVs are held. If the criteria for which SCLs test are not meaningful, SCLs will find that neither developers nor consumers of software care about the certification process.

For SCLs to succeed, it also is imperative that they employ accurate assessment technologies for objective criteria. If SCLs do this, malpractice suits against them will be difficult to win unless they mishandle a particular case or make false statements.

This article is entitled "Software Certification Laboratories: To Be or Not to Be Liable" because until these hard issues are resolved, it is hard to measure the degree of liability protection afforded an ISV by hiring the services of an SCL. Nonetheless, if SCLs can measure valuable criteria (and by this I do not mean "lines of code") in a quick and inexpensive manner, SCLs have the ability to foster greater software commerce between vendors and consumers. This could move an SCL certificate from being viewed as a tax to a trophy. ◆

About the Author

Jeffrey Voas is a co-founder of and chief scientist for Reliable Software Technologies and is currently the principal investigator on research initiatives for the Defense Advanced Research Projects Agency and the National Institute of Standards

and Technology. He has published over 85 refereed journal and conference papers. He co-wrote *Software Assessment: Reliability, Safety, Testability* (John Wiley & Sons, 1995) and *Software Fault-Injection: Inoculating Programs Against Errors* (John Wiley & Sons, 1997). His current research interests include information security metrics, software dependability metrics, software liability and certification, software safety and testing, and information warfare tactics. He is a member of the Institute of Electrical and Electronics Engineers and he holds a doctorate in computer science from the College of William & Mary.

Reliable Software Technologies
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166
Voice: 703-404-9293
Fax: 703-404-9295
E-mail: jmvoas@rstcorp.com

References

1. Jones, C., "Legal Status of Software Engineering," *IEEE Computer*, May 1995.
2. UCC Article 2B (Draft), November 1997, the American Law Institute and the National Conference of Commissioners on Uniform State Laws.

CMM, from page 20

time and within budget. Best of all, we achieved exceptional customer satisfaction, which is, after all, what counts. ◆

About the Author

Rita Hadden is the information systems performance practice leader at Project Performance Corporation. She has provided leadership, coordination, and coaching on more than 70 software projects for more than 35 organizations.

Her software engineering and management experience includes 28 years working with multiple teams of developers and managers. She has successfully managed cross-platform information system projects for the private and public sectors. She is an acknowledged leader in industry best practices, software process improvement, and corporate culture change. She has helped organizations worldwide mature their software capabilities and meet their business objectives. She is certified by the SEI as a lead software process assessor.

Project Performance Corporation
20251 Century Boulevard
Germantown, MD 20874
Voice: 301-601-1810
E-mail: rhadden@ppc.com.