Software Engineering Project Management 20 Years Later

Arthur B. Pyster, *Science Applications International Corporation*
Richard H. Thayer, *Software Management Training LLC*

A little more than 20 years ago, we assembled several papers on software engineering project management for the January 1984 edition of *IEEE Transactions on Software Engineering*. Those papers portrayed the state of the practice in SEPM and looked

0740-7459/05/$20.00 © 2005 IEEE

into its future. We decided to revisit SEPM and assemble another set of articles that reflect how it has advanced over the past 20 years and offer a fresh prediction of what lies ahead.

Where were we 20 years ago?

In 1984, as some of you will recall, there was no Capability Maturity Model (first described in a 1987 technical report by Watts Humphrey) or ISO 9001 (approved in 1987). The Project Management Institute was just offering its certification program for the first time. The US Department of Defense was the largest creator of software technology, as demonstrated by its invention of the Ada programming language, sponsorship of the Software Technology for Adaptable, Reliable Systems program, and the formation of the Software Engineering Institute. Strong, rigorous configuration management, quality assurance, requirements management, formal reviews, earned-value management, and other software processes that are commonplace today were then quite unusual. Three years earlier, Barry Boehm had just defined the whole field of software engineering economics with the publication of his landmark book of the same name introducing COCOMO, the Constructive Cost Model for software.

The papers in that 1984 special issue were

- *Software Configuration Management* by Ed Bersoff,
- *Software Engineering Economics* by Barry Boehm,
- *Software Engineering Project Standards* by Martha Branstad and Patricia Powell,
- *Software Project Management in a Small Project Environment* by Bill Bryant and Stan Siegal,
- *Project Management Planning* by Jack Cooper,
- *Earned Value Technique* by Norm Howes,
- *In-house Training of Software Engineers* by Jim McGill,
- *Software Quality Assurance* by Fletcher Buckley and Bob Poston,
- *Software Engineering Projects* by Walt Scacchi, and
- *Reviews, Walkthrus, and Inspections* by Gerald Weinberg and Daniel Freedman.

Several of those papers stand out even today. Bersoff's and Boehm's articles were both seminal. Bersoff's article is still among the best at describing the ins and outs of software configuration management, which remains one of SEPM's most valuable tools. COCOMO retains wide popularity and usefulness, having spawned several updates, including Boehm's COCOMO II.

Branstad and Powell described the state of the practice in software engineering standards in 1984. They listed standards by the IEEE, the US Department of Defense, and some European software developers. At that time, the IEEE had developed five software engineering standards, so it was the most prolific standards developer, closely followed by the DoD. Since that time, the DoD has gotten out of the software standards business, the IEEE has developed approximately 36 standards, and the ISO (International Organization for Standardization) and the IEC (International Electrotechnical Commission) standard development groups have grown much more prolific.

Howes' paper on earned value called attention to another valuable management tool, which has become extremely popular in the last few years across the US government with strong encouragement from the White House Office of Management and Budget. ANSI Standard 748 for earned-value management, approved in 1998, elaborates on the process that is partially automated by several popular commercial tools. US government contracts now routinely require companies to use an ANSI-compliant earned-value management system.

Weinberg and Freedman wrote about three types of software project review:

- *acquirer-supplier review*—a management review used to determine the status of a large software development project;
- *inspection*—a formal peer review used to find errors in a software product; and
- *walkthrough*—an informal peer review used to find errors in a software product.

Inspections have proven to be one of the most valuable tools (if not the most valuable) for finding a software product's bugs. The US's NASA Space Shuttle program uses this technique extensively.

Where are we now?

Of course, many of the practices called out in the 1984 issue are now much more well

Bersoff's article is still among the best at describing the ins and outs of software configuration management.

Projects that would have taken years to complete in 1984 are now being done in months or even weeks.

known. A catalyst for their spread has been the popular CMM and its derivative, the Capability Maturity Model Integration (CMMI). But what was their effect on organizations' ability to deliver projects on time, within cost, and with required performance? Unfortunately, the 1995 Standish Group CHAOS Report and the more recent update in 2004 show quite clearly that the more things change, the more they stay the same. Despite the myriad books, training, consultants, and tools that enable best practice, most projects still fail. According to the Standish Group, fewer than 30 percent of information technology projects succeed, nearly 20 percent are canceled before completion, and the remaining 50 percent are challenged—that is, they're seriously late, over budget, or lacking expected features.

How is it possible that the best practices called out in 1984 aren't producing the desired results? We believe there are two primary reasons. First, even though development organizations commonly follow many of these practices to some degree, most still don't perform them rigorously. Most software development organizations are still Maturity Level 1 with respect to either the CMM for Software or the CMMI. Second, competitive pressures and national imperatives keep driving a stunning growth in projects' size, speed, and complexity. Projects today produce two and three orders of magnitude more code than their counterparts did 20 years ago. As an example, Microsoft Windows didn't even appear until late 1985 and was absolutely primitive by today's standards. (Remember DOS?) Projects that would have taken years to complete in 1984 are now being done in months or even weeks. Organizational leaders keep pushing what they demand from projects—and project managers—to the point where failure remains common.

What has really changed in the last 20 years that the earlier special issue on SEPM didn't address? Among the most fundamental changes have been these:

- The creation and broad adoption of SEPM-related standards, especially the prominence of ISO 9001 and the CMM in both their original and latest incarnations. For example, the CMM demands rigorous configuration management, quality assurance, peer reviews, and other techniques that the 1984 special issue highlighted.
- The backlash against the determinism of the waterfall and big-bang approaches to development and against the CMM's planning and document focus. Spiral development, incremental delivery, and agile methods are among the results of that backlash.
- The credentialing of project managers as reflected in the Project Management Institute's more than 150,000 members and the newly emerging credential standards for both systems engineers (through the International Council on Systems Engineering) and software engineers (through the IEEE).
- The ability to manage projects with a highly distributed workforce through collaboration technology made possible by the Internet and the World Wide Web.
- The reality of product lines as a well-defined software engineering discipline.

In this issue

The articles in the current special issue address aspects of how these sea changes affect SEPM.

Suzanne Garcia writes about standardization as an adoption enabler for project management practice. In the context of half a dozen standards, she analyzes where standards have helped, what makes a good standard, and the challenges that organizations face when adopting standards. Adopting the wrong standard or too many standards, even if it's trendy to do so, can be detrimental to an organization. Adopting a standard that requires an organization to shift its culture too rapidly will likely fail. Only in sustained crisis will people change rapidly and stay changed.

Agile methods are a direct reaction to the plan-driven software development approach the CMM articulates. Reconciling the flexibility of agile methods with the necessary rigor on larger projects remains a challenge. Barry Boehm and Richard Turner analyze that challenge and suggest ways to address it. For example, they propose developing architectures that support compartmentalization of agile and traditional teams. That way, the right techniques can be locally applied to the advantage of the project as a whole.

Walker Royce's analysis of successful software management style is perhaps the most controversial article in this issue. It's also a reaction to the plan-driven approach to software

development. Royce draws an analogy between creating a movie and creating software. Both, he asserts, have lots of scrap, but he doesn't see that as a failure—it's an inherent part of the creative process. Trying too hard to minimize mistakes along the way stifles needed creativity and flexibility. Both movie making and software development are successful because of a "steering leadership style rather than the detailed plan-and-track leadership style encouraged by conventional wisdom." He also argues that required process rigor is fluid. The correct answer is not either agility or rigorous planning, but each emphasized at the proper point in the development process. According to Royce, planning should generally be less detailed and rigorous at the beginning of a project and increase over time when more is known on which to base detailed plans.

C. Venugopal drives one of Royce's ideas to an extreme. A software project that's late or over budget or that doesn't meet its performance requirements is normally considered a failure. Venugopal argues that the reason for failure is often trying to crowd too much into early releases. He proposes limiting early requirements to a single or small subset of goals. Most often, he argues, even a large, complex system has a compact primary goal. It's another form of the 80-20 rule; that is, that 20 percent of the functionality provides 80 percent of a system's value. Initially, he argues, strive only for the 20 percent most critical functionality to achieve early success.

Distributed project management has become the norm in today's large projects. Despite every project manager's desire to have the entire team colocated, for all but the smallest projects this almost never happens. Teams are scattered across multiple companies, time zones, cultures, and continents. The Internet and World Wide Web have given project managers a technology base of Web conferences, whiteboards, instant messaging, and a host of other tools for a distributed team to work together both synchronously and asynchronously. Nevertheless, enormous process and cultural challenges remain. Kenneth E. Nidiffer and Dana Dolan look at how project management is altered in such a world and provide a list of potential enablers and current constraints on those enablers.

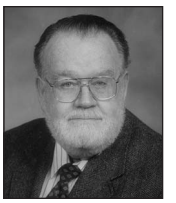
The article by Paul C. Clements, Lawrence G. Jones, John D. McGregor, and Linda M. Northrop describes software product lines, an idea that has been around for two decades but which is becoming a mainstream way of building software, whether for automobiles, telephones, or a myriad of other domains. Successfully building product lines requires different management approaches because there are really two different types of projects—those creating core reusable assets, and those applying them. Getting the right mix of people in those different types of projects, keeping reusable assets refreshed, and understanding how to manage risk are substantial challenges.

Finally, J. Fernando Naveda and Stephen B. Seidman write about the emerging credentialing of software engineers, following the patterns set by the Project Management Institute and the International Council on Systems Engineering for their constituencies. The notion of credentialing software engineers has been controversial over the years, with many disagreeing that the field was mature enough to test for the correct foundational knowledge. The emergence of an IEEE credentialing program is a major milestone in software engineering's maturation.

Twenty years is a very long time in the computing field. Yet, SEPM's progress has been agonizingly slow in many ways, probably because it's driven more by human behavior than by technology. People change their behavior much more slowly than technology advances. However, progress is notable in some areas, and this issue's articles illustrate some of the best advances in the field.

About the Authors

Arthur B. Pyster is senior vice president and director of systems engineering and integration at Science Applications International Corporation. Prior to that, during part of the time he worked on this issue, he was the deputy chief information officer of the US Federal Aviation Administration. His research interests include systems engineering and software processes. Among his accomplishments, he oversaw the development and application of three CMMs, was the chief architect of the first integrated digital environment at TRW, created and operated the information systems security program at the Federal Aviation Administration, and wrote *Compiler Design and Construction*. He is a senior member of the IEEE. He received his PhD in computer and information sciences from Ohio State University. Contact him at pystera@saic.com.



Richard H. Thayer is a consultant and lecturer in software engineering and project management and an emeritus professor of software engineering at California State University, Sacramento. He is also a Certified Software Development Professional and a registered professional engineer; he edited the *CSDP Resource Guide* and developed the CSDP Exam Preparation Course. He is a Fellow of the IEEE and a member of the IEEE Computer Society Golden Core and the IEEE Software Engineering Standards Committee. He is a principal author of two IEEE standards, including the *Standard for Software Project Management Plans*. He received his PhD in electrical engineering from the University of California, Santa Barbara. Contact him at thayer@csus.edu.