The Process Revolution

Watts S. Humphrey
The Software Engineering Institute

To mature software engineering, we must learn from history, and to learn from history, we must define what we do when we develop software and how we do it. When we produce a process specification and define measures for the key process parameters, we are beginning to build the basis for reliably and consistently recording and learning from history. That is why the process revolution is so important.

Judging by the history of other fields, software engineering and computer science are still in their infancy. We have yet to build an accepted foundation of proven and generally followed principles, and there is little in the way of defined and widely taught practices to guide working professionals. When new problems arise in the software field, the tendency is to invent some new language, tool, or method to solve them.

The true measure of maturity for our field will be when we stop dismissing the lessons of history and start building upon them. While there are many proven truths in the practice of software engineering, few software people accept them until they have personally suffered through painful and expensive failures. Unfortunately, as our field has grown and expanded, the number of ways to fail has become so large that learning through failure is no longer a viable way to advance the field.

To have a consistently effective software industrial capability, we must use defined and precise methods and well-understood procedures. Learning from history is the only way to develop such methods. While I was not the first or only person to start introducing process methods into industrial software engineering practice, I was fortunate enough to be involved in some of the early key events. This brief history provides insight into what happened, why it happened, and what is likely to happen next.

Software Factories

Since the beginning of the computer age, software projects have been troubled. Schedules were rarely met, costs were unpredictable, and total project failures were common. Of the many attempts to address this situation, one particularly ill-conceived idea was an attempt approximately 20 years ago to solve software's problems by establishing what were called *software factories*. The rationale for this idea, as I understood it, was that since typical manufacturing factories regularly produced products with predictable costs and schedules, the software community should emulate factory practices. The hope was that this would somehow enable them to achieve factory-like results.

While at IBM in the early '80s, I visited Japan and met with several software groups. One proudly took me on a tour of their software factory. It was comical. Before entering the factory's cleanroom, we had to put on smock coats just like the ones worn in semiconductor cleanrooms. Inside were dozens of software developers lined up at rows of identical work stations. While this did actually look like a factory, I had trouble believing that this factory charade could have anything but a negative impact on software development creativity and productivity. Sure enough, in a relatively short time, the idea of the software factory disappeared into the growing dustbin of software's simplistic and superficial methods that do not address fundamental problems.

Mike Fagan

A more thoughtful approach to software's problems had actually been initiated a few years earlier when Mike Fagan published his landmark paper on software inspections [1]. In it, he not only described a way to economically improve software quality, but he provided a coherent description of software process principles and explained how process concepts can be used to improve the performance of software groups. Fagan's ideas had already been tested on several projects and were found to provide truly extraordinary results. The benefits were so substantial, in fact, that he was given a $100,000 outstanding contribution award by IBM corporate management.

The 1986 Attitude

In my final years at IBM, my views were heavily influenced by Fagan and Al Pietrasanta [2]. I even drafted a short paper for the *IEEE Spectrum*, outlining a vision of how, by improving the software process, we could greatly improve the performance of software groups [3]. When I joined the Software Engineering Institute (SEI) in 1986, Fagan's inspection process was widely used at IBM but practically nowhere else. Furthermore, the ideas that he, Leon Osterweil, Bob Balzer, and others were discussing about defining and using processes were not understood by most of the software community or even recognized as important [4]. In fact, when I talked about software process at the time, the typical question I was asked was, "What in the world is a software process?"

This attitude posed a problem. Fortunately the SEI then had a shortage of experienced business managers so I was asked to run the finance and administrative operations. I agreed to do so on two conditions: 1) that the assignment would be temporary, and 2) that I could start a process group of my own. That is how we started the SEI Process Program.

The Process Maturity Model

After joining the SEI, I was asked to work with Bill Sweet of the SEI and R.K. Edwards, G.R. LaCroix, M.F. Owens, and K.P. Schultz of the Mitre Corporation to define a way to identify the U.S. Air Force's most competent software vendors. One of the key background documents we were given was an Air Force study of 17 major projects[1]. We found that the average schedule overrun for these projects was 75 percent, meaning that a typical four-year project would be completed in seven years. Furthermore, not one of the projects was completed on time or within the original cost target. We soon realized that we did not have a selection problem – all of the vendors were failing – so the fundamental problem was capability. How could these organizations improve? How could we measure their improvement? How could we motivate

continuous improvement?

We then considered the causes of project failure as well as the characteristics typically associated with the infrequent successes. Even though all of the systems that we had studied were complex and had a myriad of technical issues, we found that the principal causes of project failure were not technical. Failing groups typically had no plans, were not tracking or managing the plans that they had, or had other very basic project management failings. While we all agreed that technical issues were important, we could not identify any language, support environment, or design method that would consistently distinguish successful projects from failed projects. The only exception was the use of Fagan inspections. This practice was closely associated with successful projects and was not found in failed projects.

While we realized that the processes used by software organizations were key, we also found that the processes then discussed in the software literature were entirely technical and that none of them addressed the topics we had found most important to project success. Once we realized that the process characteristics that would distinguish between successful and failed software projects concerned management practices, we listed the key practices in question form and tried to construct a way for acquisition groups to evaluate organizations based on a questionnaire [5]. Our questionnaire had 85 questions and our contention was that a high success rate on the questions would correlate to a high likelihood of project success.

Since a typical acquisition might involve 10 or more vendors, we then had to provide a way to rank these organizations according to 850 questionnaire answers. This required some kind of ranking system. We soon developed the idea of using a maturity model such as the one devised by Philip B. Crosby [6]. We then produced a technical report for the Air Force to complete our assigned task and to provide an initial assessment method. These early ideas were published in a technical paper and a book [7].

The Capability Models

What was most surprising about the maturity model was the high level of interest by the software development community, particularly among Department of Defense (DoD) contractors. Clearly, there was a widespread need for an easily understandable way to evaluate organizational capability, to identify improvement priorities, and to measure improvement progress. We did, however, need two things before process maturity concepts would be widely accepted and used.

The first need was to have the evaluation method accepted by the key users. We accomplished this by forming an advisory board for model development that would include key representatives from all stakeholder communities. This board had to be composed of knowledgeable representatives from industry and government, and the SEI had to be willing to defer to their views. The advisory board was formed and the resulting maturity framework was called the Capability Maturity Model® (CMM®) and subsequently the Capability Maturity Model Integration (CMMI®) [8].

The second key requirement was strong backing by the DoD. This was soon forthcoming in the form of a directive that all future DoD software contractors would have to demonstrate a CMM Level 3 capability or better to win contracts.

Process Integration

As the CMM became more widely used, other groups began adopting the idea. For example, maturity models were developed for systems engineering, acquisition, and people management. Soon, the number and type of maturity models became confusing, and talk began of establishing a common maturity-model framework or architecture.

At the same time, some people in the DoD recognized that the real issue was the maturity of the entire systems development process and that software, hardware, and systems engineering were all parts of a larger and more integrated framework. At that point, the DoD sponsored – and the SEI led – a major effort to develop an integrated, multi-disciplined model to replace the CMM. This model is called the CMMI.

Where We Are Now and What Is Next

While the CMMI-based process revolution has had a major and positive impact on the performance of software organizations, the story is not yet over. There are still many organizations that have not adopted the CMMI or addressed process improvement. We are also beginning to see some organizations that have a high CMMI rating that are not performing as well as expected. This situation suggests that there are three issues: support, implementation, and omission.

The support issue refers to the long time required for adoption of any new method or technology and the need for a continuing support and enhancement effort during this time. One example of this problem is illustrated by the Fagan inspection process at IBM. With Fagan's help, development managers were convinced that inspections would sharply cut test time, shorten development schedules, and improve product quality. Many inspection courses were given and the inspection process was implemented in all of the development laboratories.

Some years later, I found that several laboratories had stopped doing inspections. On investigation, we found that after inspections had been in place for several years and many of the managers and developers had changed jobs, people started to complain about the time that the inspections were taking. The inspection advocates explained that this method was important in holding down testing costs. However, since testing was not then seen as a problem, management decided to make inspections optional. Quality then rapidly deteriorated and testing times grew accordingly. With no defined process or measures, however, the history was lost and nobody recognized what had happened.

The implementation issue concerns the skill and competence of the groups using CMMI methods. This issue relates to the support issue and concerns the problems caused by addressing symptoms instead of causes. While the evidence for poor software performance might include lack of documented plans, ill-defined processes, or incomplete data, these are only symptoms. Fixing the symptoms might get a suitable maturity rating but it does not mean that the plans were properly made and used, the processes actually followed, or the data analyzed and used. The problem is that when the DoD started requiring CMMI maturity levels for contracts, some groups found that it was easier to produce artifacts than to change engineering behavior. However, with processes, the connection between symptom and performance is behavior, and unless behavior changes, performance will not change. Without an understanding of history, it is hard to change behavior.

The omission issue concerns additional process topics that should be addressed. In developing the initial process maturity concepts, we focused on project management issues because they were the most obvious source of project problems. We also assumed that management's motivation in using these methods would be to

® CMM, CMMI, and Capability Maturity Model are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

improve organizational performance. If, however, management was primarily interested in achieving a maturity level, it would not be as likely to detect the difference between merely addressing a symptom and actually improving performance.

To improve performance, we had to address the behavior of the managers and developers. We did not initially recognize that technical and team-level issues would be critically important. Once we did, however, we extended the CMMI concepts to the personal and team levels with the Personal Software Process℠ (PSP℠) and the Team Software Process℠ (TSP℠). These methods are designed to address the behavior issue at the developer and team levels [9]. We have also started to broaden the TSP to cover systems engineering, hardware development, and the other fields that are often involved in a systems-development project. In the video game business, for example, this includes artists and game designers as well as software developers. To be most effective, such integrated teams need an integrated process framework that can support a highly collaborative working style.

Conclusions

The process revolution is now in a situation much like that of a pond with a falling water level. While the most significant rocks and stumps may have been removed, new rocks will keep appearing. With improved management practices, the rocks we are now seeing concern quality management, integrated systems management, integrated development teams, and team member practices. Early work on these topics suggests that we now face another equally productive and exciting cycle in the process revolution. After these next issues are brought under control, the heretofore concealed-but-critical rocks in the process pond that concern safety, security, architecture, design, and domain expertise will almost certainly be revealed.

What is clear from this story is that learning from history works. While our field continues to advance and the future promises to be exciting, there are useful lessons to be learned from experience. Every project is a new experience, but not every part of every project is new. Where similar work has been done before, we need to learn from and build on that experience. To do this, however, we must define, plan, measure, track, and report on what we do. Only then will our experience be available in a form that is useful to us and to others. This is true for our organizations, for our projects, for our teams, and for us.◆

℠ Personal Software Process, Team Software Process, PSP, and TSP are service marks of Carnegie Mellon University.

Acknowledgements

Many people have contributed to the progress that has been made in the process revolution field and it is impossible to name them all. However, one of the people who had a profound influence on this field and who helped me personally was Al Pietrasanta. My first exposure to Pietrasanta's work was when he developed an early course to show programming managers how to plan and manage their work. This course was taught to more than 1,000 IBM managers, and had a major impact on IBM's ability to deliver the software required to support the 360 and 370 lines of computing systems in the 1960s and 1970s. Unfortunately, he died an untimely death a number of years ago.

The second person who has made major contributions to this field is Mike Fagan. He wrote the earliest paper that I am aware of on how to define and use processes to plan and manage development work. His early work profoundly changed the way IBM managed software quality and led to a new way to view software defects and defect management. We all owe Mike Fagan a debt of gratitude for his perceptive leadership.

For this article, I have been fortunate to have several well-informed and helpful reviewers. I particularly thank Dan Burton, Bob Cannon, Tim Chick, Jim McHale, Julia Mullaney, Jim Over, and Bill Peterson for their helpful comments and suggestions. I also much appreciate the helpful guidance of the CrossTalk staff.

References

1. Fagan, Michael E. "Design and Code Inspections to Reduce Errors in Program Development." IBM Systems Journal 15.3 (1976): 182-211.
2. Pietrasanta, Alfred M. On the Management of Computer Programming. Ed. George F. Weinwurm. Auerbach Publishers, 1970.
3. Humphrey, Watts S. "Speakout: We Can Program SDI." IEEE Spectrum Apr. 1986.
4. Osterweil, Lee J. Software Processes Are Software, Too. Proc. of the 9th International Conference on Software Engineering. Institute of Electrical and Electronics Engineers (IEEE) Computer Society Press. Apr. 1987.
5. Humphrey, W.S., and W.L. Sweet. "A Method for Assessing the Software Engineering Capability of Contractors." SEI Technical Report CMU/SEI-87-TR-23. Sept. 1987.
6. Crosby, Philip B. Quality Is Free: The Art of Making Quality Certain. New York: Mentor, New American Library, 1979.
7. Humphrey, Watts S. Managing the Software Process. Reading, MA: Addison Wesley, 1989.
8. Paulk, Mark C., Charles V. Weber, Bill Curtis, and Mary Beth Chrissis. The Capability Maturity Model. Reading, MA: Addison Wesley, 1995.
9. Humphrey, Watts S. Winning With Software: An Executive Strategy. Reading MA: Addison Wesley, 2002.

Note

1. Unfortunately, I never kept a copy of this report and I do not believe it was ever published. It might even have been classified.

Additional Reading

1. Humphrey, Watts S. "Characterizing the Software Process: A Maturity Framework." IEEE Software Mar. 1988: 73-79.
2. Chrissis, Mary Beth, Mike Konrad, and Sandy Shrum. CMMI, Second Edition. Reading MA: Addison Wesley, 2007.

About the Author

Watts S. Humphrey joined the SEI after his retirement from IBM in 1986. He established the SEI's Process Program and led development of the CMM for software, the PSP, and the TSP. During his 27 years with IBM, he managed all IBM's commercial software development and was Vice President of Technical Development. He holds graduate degrees in physics and business administration. He is an SEI Fellow, an Association of Computing Machinery member, an IEEE Fellow, and a past member of the Malcolm Baldrige National Quality Award Board of Examiners. In a recent White House ceremony, the President awarded him the National Medal of Technology.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213-2612
Phone: (412) 268-6379
Fax: (412) 268-5758
E-mail: watts@sei.cmu.edu