Confusing Process and Product: Why the Quality is not There Yet

David A. Cook
Software Technology Support Center

For years now, the Department of Defense (DoD) and commercial software development organizations have embraced the Software Engineering Institute (SEI) Capability Maturity Model (CMM). In addition, there are many organizations that are rushing to meet the requirements of International Organizations for Standardization (ISO) 9000 and 9001. Unfortunately, organizations that meet CMM or ISO requirements are not necessarily producing quality software. This article discusses some impediments to software quality that remain in spite of CMM or ISO certification.

Quality Defined

Quality is a difficult thing to formally define. If you consider the strict definition, ISO 9001 suggests that it is "meeting requirements[1]." This is important, but not sufficient. In my experience, software that meets requirements is inadequate. Most software developers will quickly point out that many requirements are implied or implicit, often unstated, and frequently not addressed until implementation occurs. In addition, most end-users are concerned with reliability and robustness. Reliable software does what it is supposed to do, and does not do things it is not supposed to do. Robust software not only is reliable but also works dependably when confronted with unexpected or unanticipated conditions. Because software systems today are so large and complex, and are often expected to work under severe conditions where failure could mean loss of human life, these systems need to be robust.

The CMM and ISO

Regardless of how you feel about quality, reliability, or robustness as criteria, we all agree that most, if not all, software needs to have improved quality. Before we can improve the quality, however, we need to determine how the software is built. And for that, we need to define how we built the software. One of the best efforts in recent years to improve how we engineer software has been the SEI CMM [1]. The CMM still is not universally respected by all practitioners but it unquestionably alerts an organization to the practices that must exist for good, reliable software engineering to be performed.

Many DoD organizations have achieved CMM Level 3 (the "defined" level). The definition of this level is that the software processes for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. In effect, Level 3 of the CMM removes the "superprogrammer" as the main reason for a company achieving good software. Watts Humphrey said, "There is a common view that a few first-class artists can do far better work than the typical software team. … If this were true, one would expect that those organizations that have the best people would not suffer from the common problems of software quality. … Experience, however, shows that this is not the case" [2]. CMM Level 3 requires an organizational process that tries to overcome the "superprogrammer" mindset and focuses on sound software engineering principles for all developers.

Note, however, that organizational processes are usually insufficient for truly great software — software is still developed by individuals. Personal processes are still necessary — which is why several organizations have experienced great success in improving quality by using the Personal Software Process (PSP). PSP requires the equivalent of CMM for individual programmers — a process that addresses quality on an individual, not organizational, level [3]. PSP, when used in conjunction with the CMM, provides personal processes that complement the organizational processes, providing a better chance of quality. As another weapon in the fight against poor quality, some organizations look to ISO 9001. After all, it's very title implies that following it will produce a quality system. Unfortunately, most people confuse a quality system — which is what ISO 9001 is concerned with — a quality product. Quality systems are necessary to ensure the development of a quality product but they are not sufficient [4]. To make matters worse, ISO 9001 is not even sufficient for a complete quality system.

W. F. Fightmaster, vice president of quality for Square D (part of France's Groupe Schneider) said, "There are some people who believe that once you have ISO you have a quality system. That just isn't so. It is less than one-seventh of the system [5]." Still, it is a fact that a quality system is required if you want the end product to have quality. In mathematical terms, a quality system is necessary but not sufficient for a quality product. One customer warned me to be wary of ISO 9001 — that it was possible to design a quality system that produced concrete life jackets.

Why We Do Not Have Quality Yet

The point of this article is that we have tools that work. The CMM improves the organizational process, PSP improves the individual process, and ISO 9001 provides a quality system. The question remains: Why are we not seeing great increases in quality?

Based on several experiences, I can now point out three problem areas where most organizations fail. In my opinion, all

three "truths" are obvious. Perhaps that is why I need to state them, because obvious truths are sometimes the hardest to see and understand.

- *We are not using common sense.*
- *We have one process we publicize and another process we use.*
- *Good practices cannot overcome really poor management.*

Tailor the Process to Your Needs

Truth No. 1: We are not using common sense. The Capability Maturity Model is a process, not a product. Achieving CMM Level 3 is not the end, quality software is the end. I recently worked with an organization that wanted to organize their software engineering process group (SEPG) to help it achieve and coordinate development of its software process. It located another organization that had recently organized their own SEPG and copied the documentation almost directly. One problem — the organization they borrowed from had nearly 300 developers, plus several levels of management, while its own organization consisted of 17 people. Imagine, a 17-person shop following guidelines set up for a 300-person organization. A skimming of their SEPG documentation convinced me that it would have to spend more than 50 percent of their work time in SEPG-related meetings. Yet, the organization managed to achieve their CMM Level 3 — in spite of the fact that they could not produce software within their own process.

The point is not that common sense does not exist, it is that we forget the difference between the product (quality software) and the process (following the CMM). The CMM should be a "living process," in that frequent reviews lead an organization to self-improvement. This is the purpose and intent of the CMM, yet most organizations I have worked with treat the CMM-related documentation as a standard. Many of the individual developers treat the CMM with the same loathing that we used to regard Military Standard 2167A. MilStd2167A now is regarded for its perceived imposition of the waterfall development model, inflexibility related to object-oriented design, excessive documentation, no guidance on management indicators, and the need to incorporate new development techniques such as reuse and re-engineering [6]. Yet, some organizations have set into place processes that are equally inflexible in similar ways. We cannot afford to put processes in place that do not work.

As a further comment on the lack of common sense, a recent SEI monograph [7] discussed some problems with a government project. Integrated process teams were not integrated — there was a "government" side and a "contractor" side. This monograph is well-worth reading — it points out where common sense was lacking.

Use a Process that Works for You

Truth No. 2: We have one process we publicize and another process we use. It is my firm belief that most organizations I have worked with produce good software because most of the low-level developers have internalized the work-arounds in the system. The process does not work and is not modified. Yet, the developers have found ways to make the system work in spite of the documentation and process. This is a yardstick that I use when I consult with an organization: If the developers are not really following the process, then the process does not work. This is not saying that developers will automatically follow a good process; I think that good software developers have some type of genetic defect that makes them want to buck the system most of the time. But software developers can innately tell when the process will or will not work, and will follow a process once they are convinced that it is beneficial to them. One organization I worked with had developers that fought an organized review process. They fought until they saw the benefits in terms of rework and maintenance. If all developers are ignoring a process, then the "public process" is for show and the "hidden process" is the one that works.

Management Needs Common Sense, too

Truth No. 3: Good practices cannot overcome really poor management. When I was little, my heroes were Superman and Batman. Now, my hero is Dilbert (actually his creator, Scott Adams). His "Pointy-Haired Boss" [8] seems to typify what is wrong with software engineering. Every software development organization I visit has Dilbert cartoons posted. Why? Because the problems seem to hit home. Gerald Weinberg, in his book, *The Psychology of Computer Programming* [9] says, "bad supervision and leadership is more common than we would like to imagine." A recent customer I worked with had totally separated the developers from the analysts, and the analysts from the functional domain experts. The reason, supposedly, was to "improve communication by providing single-point interfaces." The real reason, of course, was a turf battle. This turf battle, where several managers were unable to allow free communication between co-workers under their control, resulted in software that could neither be verified nor validated. Managers need training in current practices and techniques, and they need to have a buy-in for the ISO and CMM. If management does not understand what is expected, they cannot be blamed for not following the process. Here is a sad fact — managers who have not really changed their processes since the '60s (and still think that the "waterfall model" is just a new-fangled, passing fancy) will never be able to creatively lead a team that produces quality software.

Often, the case is not even that managers are old-fashioned. Frequently they know nothing. With great regularity, we still have cases where medium- to large-scale software acquisition occurs by managers who do not understand the basic fundamentals about requirements and contracting. Asking "What are my requirements?" after half a million dollars has been spent on unusable software might stimulate the national economy, but only causes frustration on the part of the poor users trying to use a system that does not meet any of their needs. In short, developing and procuring software requires the expertise of people who have training and experience in software development and acquisition. Managers who ignore this advice and attempt to do it on their own end up with useless systems. Frederick

Brooks must have had these managers in mind when he made the observation, "Plan to throw one away: you will, anyway [10]." Alan Davis says that good management stifles motivation and erases good work they have accomplished [11].

Summary

So what is the result? Throw CMM and ISO to the winds? No. Proponents of CMM and ISO need to dig in harder. ISO 9001 and the CMM might not be the ultimate tools but they are the best tools we have. What is needed is a healthy dose of common sense. Quality software requires a process, and the process must be different for each particular organization. In addition, the process must be self-modifying and dynamic, again to meet the specific needs of each product and organization. Remember that quality software is the end, and that the process is the means. If we keep our eyes on the target and modify the process to allow us to reach it, quality software can be produced.

About the Author

David Cook is a principal member of the technical staff, Charles Stark Draper Laboratory, under contract to the STSC. He has more than 25 years experience in software development and has lectured and published articles on software engineering, requirements engineering, Ada, and simulation. He has been an associate professor of computer science at the U.S. Air Force Academy, deputy department head of the software engineering department at the Air Force Institute of Technology, and chairman of the Ada Software Engineering Education and Training Team. He has a doctorate degree in computer science from Texas A&M University and is an SEI-authorized PSP instructor.

Software Technology Support Center
7278 Fourth Street
Hill AFB, Utah 84056
Voice: 801-775-3055
Fax: 801-777-8069
E-mail: cookd@software.hill.af.mil

References

1. Paulk, M. et. al. "Capability Maturity Model for Software," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa. 1993.
2. Humphrey, W., *Managing the Software Process*, Addison Wesley, 1990.
3. Humphrey, W., *A Discipline for Software Engineering*, Addison Wesley, 1995.
4. Radice, R. "ISO 9001 Interpreted for Software Organizations," Paradoxicon Publishing, 1995.
5. Henkoff, R., "The Hot New Seal of Quality," *Fortune magazine*, June 28, 1993, pp. 116-120.
6. Newbery, G. "Changes from DOD-STD-2167A to MIL-STD-498," *CROSSTALK* The Journal of Defense Software Engineering, April 1995.
7. Carney, D. "Case Study: Signigicant Schedule Delays in a Complex NDI-Based System," SEI Monographs on the Use of Commercial Software in Government Systems, Software Engineering Institute, Carnegie Mellon, Pittsburgh, Pa. 1998.
8. Adams, S. *Seven Years of Highly-Defective People*, Andrews McMeel Publishing, 1997.
9. Weinberg, G., *The Psychology of Computer Programming*, Silver Anniversary Edition, Dorset House, 1998.
10. Brooks, Frederick P. Jr., *The Mythical Man-Month*, Addison Wesley, 1995.
11. Davis, Alan M., *201 Principles of Software Development*, McGraw-Hill, 1995.

Note

1. ISO 9001, Quality system — Model for quality assurance in design, development, production, installation, and servicing.