Don't Say the 'P' Word

By Lori Pajerek
Lockheed Martin Federal Systems

Process maturity has been extensively analyzed and codified, and the goal of process maturity has become pervasive throughout industry, government, and academia. The Software Engineering Institute estimates that there are more than 30 process maturity models and more are being developed. 'Process' has been a buzzword for a long time, and it may seem at times that more attention is paid to the process used to produce a product than to the product itself. This article examines the background experiences that led to developing process maturity models, deconstructs some of the arguments that have been posited—both for and against—and discusses some lessons learned.

The problems linked to software development have retained their challenge since they were first documented by Frederick Brooks in the *Mythical Man-Month* [1]. Then as now, software-intensive development projects have been plagued by lack of predictability, schedule and cost growth, failure to meet requirements, and as a direct result of these, low customer satisfaction. We see numbers quoted that show how dismal the state of the art is; that the vast majority of software projects are failures [2]. We have searched for solutions to these problems, seemingly ever since the days of Ada Lovelace.

There has been no lack of proposed solutions. Alan Davis chronicled a list of fads that have appeared on the software scene every few years since the 1970s [3]. Structured programming, object orientation, reuse, commercial off-the-shelf (COTS) products and others have had their day in the sun. Each has been heralded as a silver bullet, which none has been. This is not to say these ideas have no value; rather, that one must separate the substance from the hype. We incorporate what is valuable and make it standard practice. Each is recognized as one piece of the puzzle, not the whole solution. Progress is achieved slowly, small gains are made with each step. Because the problems remain largely unsolved, and because each of these doctrines is introduced with such fanfare, we are repeatedly seduced by the promise of a Holy Grail. Davis categorizes process maturity as one of these fads.

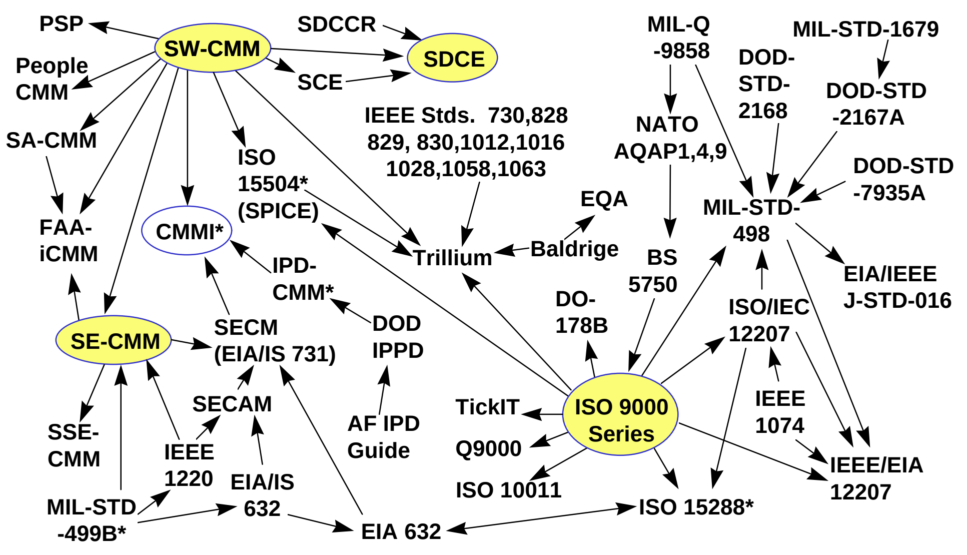
When Sarah Sheard writes about the Frameworks Quagmire [4], she is reflecting some of the frustration of those attempting to comply with a confusing and sometimes conflicting set of process dictates. One look at her pictorial representation (see Figure 1) is enough to make us say that this has gone too far. The word 'quagmire' evokes the feeling by many that they are drowning in too much process. Other writers note that there is a growing body of opinion that the practices that the Software Engingeering Institute's Capability Maturity Model (SEI CMM®) advocate are justified only for large and complex projects [5]. I can hear the debate now. Let's listen:

The Debate

Devil's Advocate: Process champions are quick to cite data to support the contention that organizations that adopt better processes produce better software. However, this is far from proving a cause-and-effect relationship between the two. The data put forth as evidence would never hold up against the standards applied to hard science. Statisticians know that there is a big difference between demonstrating a statistical coincidence and proving cause and effect. Because two phenomena coincide in a statistically significant sample does not necessarily mean that one causes the other.

Statistical coincidence is never accepted as proof of cause and effect. There must be additional empirical evidence to prove that one thing is the direct cause of another. Organizations that have good processes and also produce good software may be a coincidence, and may be due to a third factor, such as exceptionally good software engineers. Davis makes the point that with the right people you can succeed without process maturity, but that the best process in the world will not make you successful if you have the wrong people. It seems likely that if an organization is blessed with good engineers, it will be able to create good processes as well as good products. Or perhaps it is because companies that

Figure 1. *The Frameworks Quagmire*



* Not yet released

Copyright 1998, Software Productivity Consortium, NFP, Inc. Used with permission.

make good software also make money—enough money to support the overhead of process management.

The Rebuttal

Process Advocate: Unfortunately, we probably can not expect ever to have the kind of data that would satisfy a scientist. How could you ever really trace the *goodness* of a certain piece of software back to first causes with any certainty? You can only observe the statistical coincidences and draw inferences. Is that not good enough? If the two things go hand-in-hand, is it unreasonable to think that obtaining one of the two will increase your chances of obtaining the other? Remember Jay Forrester's warning that "intuitive judgments about cause-and-effect relationships may not be effective in complex feedback systems . . . with their multiple feedback loops and levels. Complex systems have a multitude of interactions, not simply cause-and-effect relationships. Causes may not be proximate in time and space to effects [6]." Certainly this comment applies to the complex relationships inherent in a development process.

Good software may be developed without a good process—*once*. That is an accident. Good processes help ensure that good software can happen more than once. A smart hacker can write a killer application, and undoubtedly makes some mistakes along the way. To an individual, those mistakes may not matter very much. Time and effort wasted are probably of little consequence. But in an organizational environment, mistakes *do* matter, because they consume valuable resources. Nobody ever sees software scrap, but it affects the bottom line as surely as hardware scrap. Can you imagine any manufacturing manager tolerating the amount of scrap coming off his production line as is routinely accepted in our software factories? Software scrap may not represent an expense in raw materials, but costs are incurred in terms of labor and schedule time, both of which are usually in short supply.

Devil's Advocate: I will acknowledge that software scrap is bad, but what makes you sure that process maturity will reduce it? Many companies are investing large amounts of money to raise their CMM maturity level, but do they really know if they will get the expected payoff? They will at least get bragging rights to a high SEI rating, which will probably be an indirect cause of increased revenue. But it may be that they would get a better value by using that money to hire the best engineers they can. It is hard to say, because we can not be sure which is the cause and which is the effect.

Process is merely a means to an end. It actually is a combination of ends: technical, quality, cost, and schedule performance, that will ultimately lead to customer satisfaction. There is evidence that companies with mature processes do indeed achieve these objectives. But even accepting the inference of a cause-and-effect relationship between the two, you must still consider whether there might have been another way to meet that goal. This is not to suggest that process maturity is bad, but that you should think about why each process step exists, and whether there is a better (i.e., easier or more cost-effective) way to achieve the same end. Furthermore, organizations can not afford to lose sight of the fact that achieving the highest levels of quality and customer satisfaction may not be the pre-eminent goals. They must also make enough profit to stay in business.

Process Advocate: You are right, process exists as a means to an end. Individual process steps exist for a variety of reasons —technical, cost, schedule, legal. Some steps are recommended as best engineering practices; they are there to promote technical quality. Other steps within the same process are there only for management reasons. If we pretend that we have infinite time and money to complete a project, we would not have to employ a lot of management controls. Since that situation never occurs, we have to find ways to optimize technical quality while maintaining some schedule and cost limits. Because these controls have to be embedded in the engineering tasks, our engineering processes contain a mixture of management and technical procedures. The advent of initiatives like Integrated Product Development have intertwined them even more, making it harder to separate the strands.

Software has given us the capability to build systems that are far more complicated than before its advent, and they are becoming more complicated all the time. Now that we are building systems containing millions of lines of code, development must be parceled out. Process is what makes this division of labor succeed; just being smart does not carry it off any more. Documented processes become the repository for an organization's collective wisdom and experience—lessons learned about what works and what does not.

Devil's Advocate: You agree, then, that less process is needed for small programs. Many organizations recognize this fact intuitively, and let smaller programs off the hook. That helps relieve the burden, but it does not help them figure out what *is* appropriate for programs their size. Even for large programs, one of the process mantras is that "the process must be tailored for your program." Practical tailoring guidance is hard to come by and difficult to apply.

Process Advocate: Yes, many of the sub-processes of systems engineering are really systems management. They exist, like the science of systems engineering, because of the need to manage large, complex efforts that involve many components, interfaces, people, sites, companies, etc. They are needed to keep a large project organization moving in sync towards a single goal at the right time. These subprocesses are more easily waived when you have a small team and a short schedule.

One mental exercise that can help is to assess each process step against the following questions:

- Would I do this if I were building this product by myself?
- Why is this process step here; is it for a technical or a managerial reason?

This will help to identify and segregate the steps that exist primarily to serve management goals. Those may be negotiable on small projects. For example, process steps that require completion of checklists, preparation of status reports, or multiple levels of approval to do things can probably be simplified or eliminated.

People fear that processes are going to tell them how to work. The most important processes for an organization to mature do not prescribe how to work but how to coordinate. No one minds being told how to get others to do what they want them to do. That is the function of most good processes. A newly minted engineer may not initially understand why he has to follow the process—he never had to do this in college. Eventually he sees the value when he realizes he is no longer working by himself, and his success is dependent on what others do.

When Bad Things Happen to Good Engineers

It must be understood that process maturity can not be realized by the efforts of engineers alone. As Watts Humphrey has noted, poor project management will defeat good engineering, and is the most frequent cause of project failure [7]. When managers insist that engineers shortcut the best engineering practice due to schedule or budget pressures, process maturity fails. These managers are often responding to inflexible contract demands to which their company committed in order to win a competitive bid. The procurement process encourages bidders to submit proposals that set unrealistic schedule and cost targets. Hence, while the customer community may be professing the desirability of a contractor's high level of process maturity, the incentive to industry promotes the exact opposite result. Thus, another prerequisite for a mature development process to thrive is the co-existence of a mature procurement process on the customer's part [8]. Even then, reality has a way of subverting the best intentions. Although everybody wants to accrue the benefits of capable processes, managers often experience 'sticker shock' that causes them to cut corners.

A hallmark of a mature development process is emphasis on early requirements analysis and up-front planning. This requires program schedules and budgets to be more heavily loaded on the front end. Despite having heard the caution, "Pay me now or pay me later," some program managers think they can get away with not paying at all. This includes managers in both customer and contractor organizations. Budgets and schedules are drawn up optimistically, trusting in a best-case scenario. This is rarely the scenario that unfolds, and the fact that it does not is largely a self-fulfilling prophecy.

Spending a lot of time and money up front is an expression of faith. It manifests the belief that heavy investment in thorough requirements analysis, trade studies, etc. will prevent problems later, and that the cost of fixing those problems would be greater than the initial investment. But it is difficult to quantify the cost of problems that never happen, so it is hard for program managers to commit to spending lots of resources early, when the payoff is so intangible. Despite numerous studies to support its validity, few programs are scheduled and budgeted this way. There is never sufficient time and money to do all the tasks that engineering best practice would dictate, i.e., to follow the process! Projects seem to be programmed for failure before they even start. Once the inevitable problems occur, projects revert to crisis management mode, which is not noted for its high level of process maturity.

While engineers working in the real world are told to follow best practices, circumstances often make it difficult for them to do so, and they are blamed when they fail. Is it any wonder they become annoyed and think that the Process Police issue lofty dictates from the ivory tower, while they struggle in the trenches to get the product out the door? Who can be surprised that engineers fighting daily fires do not want to hear the 'P' word? They want to do the right thing, but it seems as if their hands are tied when, for example, they have six months worth of requirements analysis activity to squeeze into the four weeks allotted in the schedule. Watts Humphrey also states that if engineers can reasonably defend their plans, management should respect these plans and not override them with schedule edicts. Too often, this caution is ignored.

Where Process Models Fall Short

Investing time and money up front should not require a leap of faith. We ought to be able to draw up a front-loaded schedule and budget with a reasonable degree of certainty that we can shorten the back end without undue risk. We can not do this today because of deficiencies in our process models and our base of collected measurement data. The reason we create models is to validate designs by varying operational parameters and conditions, and observing the results. Engineers trust their models because they trust the data that goes into them. Program managers, however, can not place the same level of trust in process models because we lack the abundance of hard data to substantiate them.

Many current process models are deficient because they do not always view the processes as systems. The process is the system an organization uses to generate all its other systems, i.e., its products. However, few organizations apply the same engineering rigor to creating their development processes that they apply to the systems developed for external customers. A properly engineered development process would be modeled in such a way that managers could accurately foresee the outcomes of various hypothetical actions. What happens to Integration and Test (I&T) time if we cut the Requirements Analysis time in half? Who knows for sure? We can probably guess correctly that it will increase, but by how much? Ideally, we could vary these parameters in a formula to find the optimum balance.

An example discussed previously provides a further illustration of this point. We have all seen empirical data to support the premise that the cost of correcting system defects increases as the system development progresses into later phases. But process models have not fully instantiated this data to the point where we can quantitatively predict the downstream effects of qualitative changes to the process. In an ideal model, you would be able to calculate how much time devoted to requirement reviews or design inspections is required to remove a certain percentage of defects, and at what point the law of diminishing returns indicates that the I&T savings no longer offset the cost of additional up-front reviews. The accuracy of such a model depends on having a substantial amount of validated measurement data, something that few organizations possess today. It is essential that we collect this data if we are to arrive at the point where our process models can be sufficiently calibrated to perform these sophisticated *what if* analyses.

Integrated Product Development notwithstanding, few process models in use today are truly integrated. Even with the best intentions, organizations tend to quickly decouple constituent subprocesses as they descend through a top-down decomposition of their development process. Except for time sequence dependencies, the system-wide effects of changing one subprocess are unknown. Practitioners still operate within their traditional stovepipe processes, even though their command media and their management structure may proclaim them integrated.

Another shortcoming in current process models is that the rationale for each process step is rarely captured. If it were, this type of data could be of great value when attempting to tailor or re-engineer a process. People are often afraid to change or eliminate a process step if they do not know why it is there. Worse, more aggressive individuals may rashly eliminate steps because they do not immediately see their value. If they were aware of the possible consequences, they could at least take the risk knowingly. While some organizations collect historical lessons learned information, it is often poorly organized, difficult to access, and not kept up-to-date. Most signficantly, it is not tied directly to the process model. A good process model would have links to these lessons learned to show when a process step was changed, added, or deleted as the result of a lesson learned. We often maintain rationale for other engineering decisions, why not for processes? Capturing rationale for technical decisions is another one of the tenets of a mature engineering process. This is just another example of how we need to engineer the development process with the same rigor we engineer other systems.

Taking the Long View

Continuous Process Improvement (CPI) sounds like a good thing, and it is. The point is not that CPI is an impossible or unworthy goal, but that like the process, CPI is a means to an end. There may well be alternative routes to the same end. Organizations pursue CPI because they believe it will increase productivity and quality while reducing the cost of doing business. However, even the authors and champions of maturity models admit, when questioned, that there is no hard data to quantify the return on investment (ROI) in process maturity. Unless the goal is to achieve a CMM Level 5 rating for its own sake, it is valid to suggest that money spent on CPI may be better spent elsewhere in pursuit of the same goals. We should not forget that the law of diminishing returns will apply to CPI just as it does to any other investment. CPI requires a significant capital investment, with a promise of return on that investment. As with many corrective actions, the biggest ROI on CPI is achieved by a few heavy hitters. This is embodied in the well-established theory of Pareto analysis. For this reason, many practitioners believe that the difference between a Level 3 organization and a Level 1 organization is probably greater than the difference between a Level 5 organization and a Level 3 organization.

Once an organization has achieved a high level of process maturity, it is valid to question whether the ROI on continued improvement is sufficient to justify the investment. The gains will become smaller and smaller, and there may be a point where the investment is larger than the payback. Most organizations are still at relatively immature levels of process capability, and there are many valuable gains to be made. In this current state, it is hard to imagine that someday we will be at the point where all the low-hanging fruit has been picked. When that day arrives, we will have to take a hard look at the received value of pursuing CPI.

Also, there is a danger of confusing continuous improvement of the process with improving the process deployment. Deploying any reasonably adequate process rigorously and uniformly is of greater value than having a perfect process on paper, but not enforcing it effectively. An organization's priority, therefore, might be to ensure that a majority of programs within the organization are performing at Level 3 (for example), before investing in advancing to Level 5 for a limited number of programs.

The bottom line is that process maturity is only one of many factors that contribute to the ultimate success or failure of any project. There is no doubt that personal attributes such as education, training, and work ethic of the individuals executing the process will also have an effect. Likewise, the finest engineers can not perform up to their potential if not given an adequate working environment with sufficient resources of time, money, and tools. Finally, even the best engineers with the most ample resources may still fail if the project is badly managed in other ways. Process is just one ingredient in the mix.◆

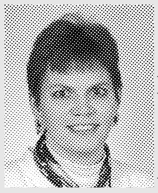
References

1. Brooks, Frederick, *The Mythical Man-Month*, Addison-Wesley, 1975.
2. Alder, Rudy, Instead of the Wrecking Ball, *CrossTalk*, May 1998.
3. Davis, Alan, Software Lemmingineering, *IEEE Software*, September 1993.
4. Sheard, Sarah, The Frameworks Quagmire, A Brief Look, *Proceedings of the Seventh Annual International Symposium of the International Council on Systems Engineering*, August 1997.
5. Hadden, Rita, How Scaleable Are CMM Key Practices?, *CrossTalk*, April 1998.
6. Hughes, Thomas P., *Rescuing Prometheus*, Pantheon, 1998.
7. Humphrey, Watts S., Three Dimensions of Process Improvement Part I: Process Maturity, *CrossTalk*, February 1998.
8. Courteney, H. and Ruston, S., Mature Procurement of Large Scale Systems: A Better Way to Buy, *Proceedings of the Seventh Annual International Symposium of the International Council on Systems

Additional Reading

Gundrum, Valerie, Architecture for a Process Meta-System, *Proceedings of the Seventh Annual International Symposium of the International Council on Systems Engineering*, June 1999.

About the Author



Lori Pajerek is an Advisory Systems Engineer at Lockheed Martin Federal Systems in Owego, N.Y. Her current assignment in the Systems Engineering Technology department includes surveying, evaluating, selecting, and deploying Systems Engineering tools. With more than 15 years experience in systems and software engineering for defense-related industries, her areas of interest and expertise is requirements engineering and management. She has a bachelor of science degree in mathematical sciences from Binghamton University, and is a member of the International Council on Systems Engineering (INCOSE).

Lockheed Martin Federal Systems
1801 State Route 17C
Maildrop 0210
Owego, N.Y. 13827
Voice: 607-751-6226
Fax: 607-751-6025
E-mail: lori.pajerek@lmco.com