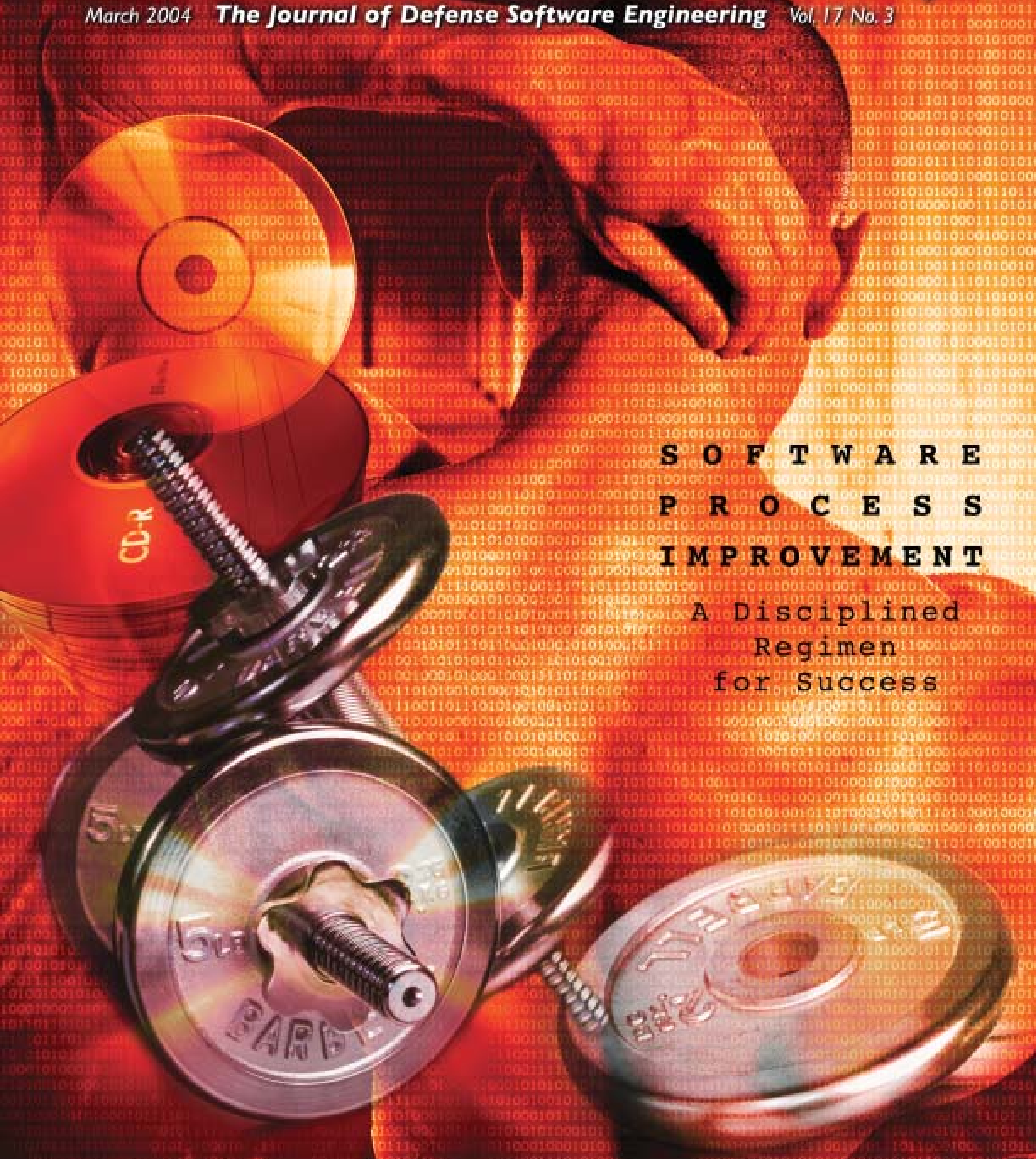
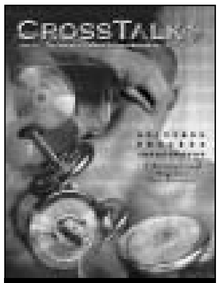
CrossTalk

March 2004 **The Journal of Defense Software Engineering** Vol. 17 No. 3



SOFTWARE PROCESS IMPROVEMENT

A Disciplined Regimen for Success


ON THE COVER
Cover Design by Kent Bingham.

CrossTalk

Publisher	*Tracy Stauder*
Associate Publisher	*Elizabeth Starrett*
Managing Editor	*Pamela Palmer*
Associate Editor	*Chelene Fortier-Lozancich*
Article Coordinator	*Nicole Kentta*
Creative Services Coordinator	*Janna Kay Jensen*
Phone	(801) 586-0095
Fax	(801) 777-8069
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/ crosstalk

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 29.

Ogden ALC/MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/ crosstalk/xtlkguid.pdf>. CrossTalk does not pay for submissions. Articles published in CrossTalk remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CrossTalk.

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CrossTalk Editorial Department.

STSC Online Services: www.stsc.hill.af.mil
Call (801) 777-7026, e-mail: randy.schreifels@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CrossTalk available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Buying and Building Systems and Software Better

A lot of us are talking about systems these days, and rightly so. But this month we are focusing on the pivotal aspect of any modern military system – software. Our theme this month is software process improvement (SPI). We have featured this theme many times in the past primarily because SPI is what CROSSTALK and it's parent organization, the Software Technology Support Center (STSC), are all about. CROSSTALK helps the STSC achieve its mission through publishing informative articles aimed at the software professional as they strive to buy or build software better. Whether you are designing, developing, testing, configuring, managing, sustaining, or even buying software, we aim to increase your awareness of the benefits and challenges of disciplined processes in all phases of the software life cycle.

CROSSTALK is also part of a Capability Maturity Model® (CMM®) Level 5 organization, the Ogden Air Logistics Center's Software Engineering Division OO-ALC/MAS at Hill Air Force Base, Utah. It is a privilege to be a member of a Level 5 organization as process improvement is embraced at all working levels. SPI is a major contributor to high workplace morale and customer satisfaction. MAS is currently in the midst of putting new processes in place as well as updating existing processes as they work to be in compliance with the CMM Integration^{SM} (CMMI®) and prepare for a CMMI assessment later this year. Ogden's MAS division is a great example of an organization that has learned to build software better through their application of SPI techniques in all facets of software development.

We begin this month's issue with *Accelerating Process Improvement Using Agile Techniques* by Deb Jacobs. This author discusses how an organization can get the most *bang for their buck* by putting processes in place quickly while still remaining agile in a business environment. The author presents a methodology that includes a common sense, simple, step approach to developing an organization's maturity. Although processes can be put in place fast, Jacobs emphasizes that the processes become a foundation for continued improvement over time.

Next, Dr. Miguel A. Serrano and Dr. Carlos Montes de Oca bring us *Using the Team Software Process in an Outsourcing Environment*. This is a good example of using the Software Engineering Institute's Team Software Process^{SM} (TSP^{SM}) model as a method of buying software better. The TSP techniques were used to estimate a legacy system upgrade project's cost and schedule even before signing a contract. The information used to produce the cost quote was an outcome of a TSP launch. The article describes the main problems and results of using the TSP and discusses lessons learned from the experience.

Unlocking the Hidden Logic of Process Improvement: Peer Reviews, by Marilyn Bush is featured next. This author discusses how the CMM and the CMMI define necessary process tasks but the models fail to describe the logic that sequences these tasks. Bush chooses peer reviews as an example and discusses the reasons for doing peer reviews as prescribed by these models. She also offers her insights and warnings if peer reviews are attempted too early in an organization's process improvement journey.

Process documentation can be looked upon as a necessary framework or foundation for any process improvement effort. Ronald A. Starbuck expands on the importance of process documentation when it is done right in *A Beginner's Look at Process Improvement Documentation.*

In our supporting articles this month is *Common Errors in Large Software Development Projects* by David A. Gaitros. This article is a good reminder of the common mistakes made while producing software and discusses practices that when implemented can increase your project's success at developing software. In our Open Forum section, Michael West addresses this month's theme with *Applying Systems Thinking to Process Improvement.* West reminds us that process improvement should not be done if an organization just wants to be glorified with achieving a certain CMM or CMMI level as this just creates more problems for them to deal with. The author presents how systems thinking can provide a bigger picture view of commonly occurring systemic problems in organizations and further helps with resolving the problems that frequently occur in model-based process improvement efforts.

I hope this month's collection of articles provides you with helpful information as we all work together to buy and build systems and software better.

Tracy L. Stauder

Tracy L. Stauder
Publisher

Accelerating Process Improvement Using Agile Techniques

Deb Jacobs
Software Engineering Services

In today's fast-paced, cost-conscious world, it is critical for companies to keep up with the Joneses while continuing to keep costs reasonable. The cost of process improvement efforts has proven hefty. Many companies have abandoned these efforts simply because they cannot afford the associated costs. By speeding up process improvement, companies get their bang for the buck.

Mark slugs down one more beer and figures it is about time to go home. This afterwork party is getting boring. As he is getting ready to leave, Mark sees a coworker who he has not seen in a long time.

"Hey, Joe!" Mark yells across the crowded room.

"Mark, my man, long time," Joe answers as he comes over to sit by Mark at the bar.

"How's life treatin' ya?" Mark asks, wondering why Joe, who is usually such a fun-loving guy, seems so miserable.

"Oh, okay I guess," Joe says, unconvincingly, ordering another beer for himself and Mark.

"So, how's that high profile project you were telling me about going?" Mark asks.

"Hey, between you, me, and the bartender, I've about had it!" Joe exclaims.

"So, what happened?" Mark asks. "You said this was the opportunity of a lifetime last time we talked. The promotion to software project lead is what you've been working for since college."

"This has definitely turned out to be the project from hell! I thought the last project was bad but this one beats them all!" Joe complains. "We're always behind schedule, and the costs are skyrocketing! We had to add three more engineers and you know what that's like, between training them and trying to get the real work done, we end up even farther behind."

Table 1: *Key Success Criteria in Accelerating Process Improvement*

Key Success Criteria
Executive Management Commitment
Mid-Level Management Commitment
Organizational Adaptability ->Flexible
Project Management Style ->Proactive
Training Style ->Proactive
Communications Style ->Open and Non-Inhibitive
Delegation of Authority
Process Improvement Model Familiarity
Process Acceptance Factor ->Positive

"Yeah, I know what you mean," Mark says, empathizing. "The project I'm on makes it hard to get out of bed in the morning, too."

"It just keeps going on and on with this company, all talk and no action," says Joe. "They tell us that they're working on it but nothing ever changes, same old thing every time. Get a new project, make unreasonable promises, and who suffers? We do."

"Yeah, I know what you mean!" Mark says.

"Well, I'm not going to take it any more; my resume was out the door a week ago. I'm just fed up now. They can't all be this bad," Joe says hesitantly, and adds, "Can they?"

"I hope not. I may be right behind you, Joe. Put in a good word for me when you find something," Mark says commiserating as he orders another beer to wash down the gloom that is now starting to overcome him, too.

Why Accelerate Process Improvement?

This scenario is played out in organizations all over the country every day. Good people are lost, money is lost, reputations are lost, and, ultimately, clients are lost because of immature organizations.

The bottom line is this: Companies cannot afford to wait while bureaucracy plays itself out. The consequences can be overwhelming with projects over cost and over schedule, extensive overtime, confusion, loss of staff, misdirection, distrust, and frustration. Nobody likes to be caught with their pants down, which is typical of immature organizations. By accelerating the process improvement effort and getting processes in place quickly using agile techniques, an organization can concentrate on improving their processes over time and still remain competitive in an agile business environment.

How to Accelerate Process Improvement

There are many models and methodologies available for improving an organization's failure or success quota such as the ISO 9000 series, Software Process Improvement and Capability dEtermination, Total Quality Management, Software Process Improvement in Regions of Europe, the Project Management Institute's Project Management Body of Knowledge, and Six Sigma to name just a few.

To date, the Software Engineering Institute's (SEI^{SM}) Capability Maturity Model® (CMM®), including the CMM for Software (SW-CMM) and the CMM Integration^{SM} (CMMI®), have proven to be the most successful at maturing organizations. It all depends upon your ultimate goals with the process improvement effort being undertaken.

Regardless of the model or methodology selected, by using the keep-it-short and-simple (KISS) method, many basic elements can be put in place quickly to kick things off, thus building a foundation for continued improvement. Sometimes, we tend to concentrate on the gory details and forget the big picture.

Key Success Criteria

The SEI recommends 18 to 24 months per level for the CMM. The basic KISS philosophy and use of many of the techniques discussed in this article were responsible for bringing an organization to CMM Level 3 in just over one year. The rest of the techniques are based on the lessons learned from this and other process improvement efforts.

There are several key success criteria that organizations should meet prior to attempting to accelerate a process improvement effort. Table 1 provides a list of important key success criteria.

The SEI developed the Initiating, Diagnosing, Establishing, Acting and Learning (IDEAL^{SM}) process improvement methodology in the mid-90s to guide process improvement adopters. The

^{SM} SEI, CMM Integration, and IDEAL are service marks of Carnegie Mellon University.

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

IDEAL model is based on the Deming Cycle/Shewhart Cycle, Plan-Do-Check-Act, which provides a mechanism for perpetual change. The Accelerating Process Improvement Methodology (APIM) uses these models as a basis.

For the APIM, time is the key word. The very successful agile programming methodology was also used as a basis for the APIM. Indeed, process improvement can be quite complex, just as software development is. However, the key to process improvement is improving the way you do business, and for the majority of businesses faster, better, and cheaper is the mantra of the day. Figure 1 illustrates the APIM.

The APIM has three phases: Pre-Maturity, Maturity, and Post-Maturity. Each phase consists of various steps required to develop an organization's maturity. As illustrated in Figure 1, the Maturity Phase is iterative. It is repeated until an organization is ready for a formal assessment.

This methodology takes an agile approach with simplicity and common sense as the magic words. Many times organizations tend to over-process with multiple forms, plans, and procedures that end up being meaningless. As is usually the case, the devil is in the details.

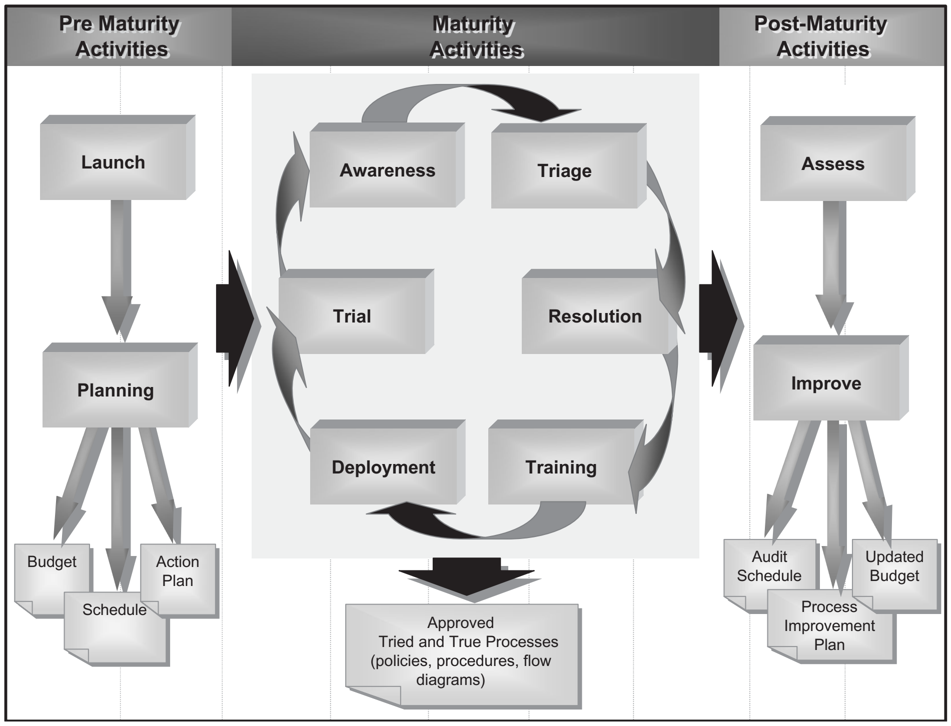


Figure 1: *Accelerating Process Improvement Methodology (APIM)*

Pre-Maturity Phase: Step 1-Launch

The launch step is key to having the appropriate resources and budget for execution of the APIM. This is where you get buy-in from executive management. Without executive buy-in, it will be virtually impossible to move ahead in any process improvement effort; however, for an accelerated process improvement effort it becomes even more critical to success. Each organization has its own method of authorizing tasks, but you should get official executive approvals and task authorizations before proceeding.

Initial resources and appointments to the process team will be an important activity during launch. Based on previous process improvement efforts, it is recommended that the core process team be kept fairly small depending upon the size of the organization and have enough hours assigned to the team to get the real work accomplished. Too many people can cause bottlenecks that prevent, or slow, real accomplishment. On the flip side, too few people, or too few hours for the people assigned, allows no room for accomplishing tasks. Additionally, assigning the wrong people with a negative attitude to the group can sabotage the effort.

The key will be to find the appropriate balance for the organization. It requires closely monitoring the group during start-up from the honeymoon stage (this is the time that people are most enthused) into the early start-up stages. It may take some trial and error to form the right group of people to accomplish the tasks required to accelerate process improvement.

An executive steering committee should be formed during launch. This committee can be very effective in providing needed resources and resolving issues throughout the process. Always remember to remain agile; do not let the bureaucracy typically associated with numerous committees bog you down. When you start putting in frequent, long, drawn-out review cycles and approvals, you stop being agile.

In order to make a difference, the process team should be armed with the authority to make decisions and changes needed to meet the process improvement goals. Arming the process team, or at least the lead, with authority, levels the playing field for them by providing an equal voice with project managers. By leveling the playing field within the organization, the process team will be better able to make reasonable changes. Even though there will, in all probability, be mistakes, the organization must be prepared to roll with the punches to ease a quick recovery.

The final and very important task during the launch step is the initial kickoff meetings. There should be at least two kickoff meetings for the organization during launch:

1. The initial kickoff meeting should be with the executive staff, as previously discussed. This may have already been accomplished at the very beginning of the launch step or even prior to launch.
2. Once the initial resources and appointments have been determined, a process team meeting will be the first activity for bringing the process team together and getting a feel for how they will work together. Remember, this is still the honeymoon stage, so it will be hard to really tell how the group will form until some time passes.

Pre-Maturity Phase: Step 2-Planning

There are many reasons why projects fail, but survey after survey has found that one of the top reasons for failure is lack of planning. Conversely, studies have found that a major reason for success is proper planning. Upfront planning is very important but, as with the very successful agile programming methodology, planning should remain as painless as possible and iterative.

A forward-looking philosophy should be employed to ensure that you are prepared for tasking but concurrently ensure enough flexibility to accommodate changing task priorities. In essence, the upfront planning should provide the basic strategy by which a process improvement effort operates. An action plan should be developed that includes the following:

- Process improvement goals.
- Major milestones and associated tasks.
- Measures to indicate status and effectiveness (remember KISS).
- Resources and appointments to the team.

- Team responsibilities.
- Initial risks and mitigation.
- Budget (this will depend upon how an organization handles budgeting).
- Completion criteria.

For each task, the responsible team member should develop a short and simple implementation/action plan. This will be discussed more fully in later phases. The major milestones should be defined in a master schedule, which is critical to the success of any project. The master schedule should always be maintained to include all current tasks, upcoming tasks, and potential tasks. Each current and upcoming task should be tracked to resolution to ensure appropriate resources are available to accomplish essential tasks as assigned. The master schedule should be developed and maintained by the process team to ensure soundness of the timelines. This will be a very important tool for success.

During the planning step, the following two additional kickoff meetings should be conducted for the organization:

1. A special kickoff meeting should be accomplished with the mid-level management staff. This is where a mid-level management commitment is received, which is as important to the success of the effort as executive management commitment. The mid-level management will ensure that things are done a certain way such as using the processes developed by the process team.
2. A kickoff meeting for the entire organization's staff will provide the first opportunity to advertise the process improvement effort. Advertising the effort will be important to getting the entire organization on board with processes and the process improvement effort. Further advertising should be done throughout the process. Remember, the ultimate goal of process improvement is to change the way the organization accomplishes its work. If the staff doing the work is not on board and aware of the effort, the changes cannot be accomplished. Some suggestions include regular e-mails, newsletters, posters, flyers, presentations, and announcements at other meetings by executive management.

Maturity Phase: Step 1-Awareness

A mini-assessment will determine where an organization is, and where they need to go. The focus is on finding and identifying the weak areas that need to be corrected or improved to meet best practices or the organization's process goals. The initial mini-assessment will set the baseline for progress mini-assessments, which should be accomplished for each iteration of the maturity phase as illustrated in the APIM diagram.

Various mini-assessment methods can be used but any mini-assessment should have a minimal impact to the organization's staff. At minimum, it should consist of a records analysis as well as interviews with the process users. The length, size, and scope of the mini-assessment will be dependent upon the assessor's knowledge of the organization and the process goals. For example if CMMI is selected, the assessor should have a good working knowledge of that model. A mix of internal and external staff, either within or outside the company, can be very beneficial to gain both organizational knowledge and model knowledge.

The mini-assessment results should be analyzed to determine the weak areas along with the level of weakness and an initial estimate of what it will take to strengthen the area. The final analysis will determine what actions need to be taken to meet the organization's process goals. The actions can be viewed much like the user stories in the agile programming methodology. The initial mini-assessment results should become the baseline for use in later iterations.

The baseline results should be maintained through use of a tool to depict the status of each project and the organization. One successful method is through maintenance of stoplight-type charts or other types of tables that indicate each project and the organization's status. These should be updated following each progress mini-assessment and regularly reported to the executive steering committee.

Maturity Phase: Step 2-Triage

Triage comes from an old French word meaning sorting or sifting. It has been used to describe the treatment of patients, especially battle and disaster victims, according to a system of priorities designed to maximize the number of survivors. For patients, the following three categories have been defined: 1) those who will not survive even with treatment; 2) those who will survive without treatment; and 3) those whose survival depends on treatment. By using triage, the treatment of patients requiring help is not delayed by useless or unnecessary treatment of those in the other groups. Triage originated in military medicine when limited resources faced many wounded soldiers and time was critical. Hence, triage decisions are made after relatively quick examination; patients in lower-priority groups are reexamined periodically.

This same triage or sorting method can be used to accelerate process improvement efforts. The prioritization categories would be a bit different with consideration for the level of weakness and the effort required to strengthen it. A second consideration is the needed actions of most value to the project or the organization. The key selection criteria should be based on three goals: business goals, project goals, and process goals.

Whatever prioritization criteria are used, this should be done swiftly in order to get the organization where it needs to go as quickly as possible. Based on his former Army experience, Christopher P. Higgins, Bank of America national manager currency Services, said, "Make a decision! Make a decision! People are dying all around you!"

Agile programming uses index cards to depict their user stories. This method could also be used effectively for process improvement with the actions needed as the user stories. Another method just as successful is to maintain actions needed in written form such as tables. For each iteration of the maturity phase, the index cards or tables must be updated to reflect the current actions needed.

Maturity Phase: Step 3-Resolution

Based on prioritization, actions are selected and assigned to process team members for resolution. For each action, the responsible team member should develop a short and simple implementation/action plan. The size and scope of this plan depends on the size and scope of the assigned task, but it should be kept minimal with the key information needed to accomplish the task. The following is recommended for inclusion: problem definition/objectives/purpose, team members, piloting strategy, desired results, issues and risks, timeline and high level tasks, and deliverables. A sample of an agile action plan can be found with the online version of this article at <www.stsc.hill.af.mil/crosstalk/2004/03/0403jacobs.html>.

During resolution, the process should be developed that may include process flows, policies, procedures, forms, and templates. The key again is simplicity: KISS. As in agile programming, do the simplest thing that will work. If previously developed complex processes are used, you should use a technique called refactoring in programming that means making the code clearer, cleaner, simpler, and elegant. This does not mean changing the functionality or rewriting processes but simplifying them for easier use.

A quick peer review method should be used to finalize processes for piloting as well as an approval process in order to ensure the integrity of the processes. As with the rest of this methodology, the key is agility and keeping it simple. Avoid bureaucracy unless highly warranted; it is time consuming.

Maturity Phase: Step 4 -Training

Training can play a pivotal role in the acceptance or rejection of a developed process. Special care should be taken with training to get buy-in from the process users. This duration should be used to tailor the process to meet any specific user needs as well as train the user on the process.

Maturity Phase: Step 5- Deployment

Processes should be piloted on a project or within the organization prior to being added to the organization's process repository. It is important to ensure that the process will work in a real situation instead of just in theory.

The process team should act as a mentor/coach for the project when piloting processes. Frequent checkups should be done to ensure that it is being used as developed and to ensure a complete understanding of each step, template, deliverable, etc.

Maturity Phase: Step 6 -Trial

Once the process has been piloted, the process team should assess the effectiveness of the processes developed for the selected action and either approve or reject them. Depending upon the severity of the findings, they may be immediately improved and approved, or improved and re-piloted during the next iteration. As always, collect lessons learned for making later iterations easier, better, and even more agile.

Post-Maturity Phase: Step 1-Assess

The final phase is the Post-Maturity Phase that starts with the formal assessment. A formal assessment should be accomplished when the progress mini-assessment indicates readiness. The method of assessment will depend upon the process improvement model or methodology selected, but it will be key to identifying strengths and weaknesses from an outside source. Some formal assessments use organizational staff and some use staff from outside the organization or company.

Post-Maturity Phase: Step 2-Improve

It is important to provide continuous improvement to an organization. Organizations change, staff changes, business goals change; many changes take place in organizations, sometimes very quickly, and processes must continuously keep up with these changes. The accelerated process improvement effort will put the initial processes needed in place as a foundation for further improvement.

In order to continue being agile and keep costs at a minimum, the accelerated process improvement method can continue to be used. Since needed processes will already be in place, it may require tailoring to meet future process needs as opposed to initial process needs. The following are some of the things that should be considered for the improvement effort:

- Improvement measures.
- Audit plans.
- Audit reports.
- Periodic progress assessments.
- Updated risks.
- Updated budget.
- Updated action plan.
- Task-completion criteria.

Phase	Step	Objectives
Pre-Maturity	Launch	• Executive Approvals • Task Authorization • Executive Steering Committee • Initial Resources/ Appointments • Kickoff Meeting(s)
Pre-Maturity	Planning	• Develop Brief Action Plan ✓ Goals ✓ Milestones ✓ Measures ✓ Resources ✓ Risks ✓ Responsibilities ✓ Budget ✓ Completion Criteria
Maturity	Awareness	• Mini-assessment to identify strengths and weaknesses • Maturity level measurement • Analyze current situation
Maturity	Triage	• Determine strategy and actions • Analyze return on investment for *each* action • Select actions using a triage approach (speed is the key here) based on importance to: ✓ business goals ✓ project goals ✓ process goals • Prioritize selected actions
Maturity	Resolution	• Select project(s) for piloting initial processes • Develop processes (remember KISS): ✓ process flows ✓ policies ✓ procedures ✓ forms ✓ templates • Process Reviews and Approvals
Maturity	Training	• Train pilot process users • Tailor, as needed
Maturity	Deployment	• Pilot processes to selected project • Mentor pilot project(s)
Maturity	Trial	• Assess effectiveness • Improve/approve • Collect and analyze lessons learned
Post-Maturity	Assess	• Determine maturity level • Identify strengths and weakness for future improvement
Post-Maturity	Improve	• Improvement Measures • Audit Plans • Audit Reports • Periodic Progress Assessments • Update Risks • Update Budget • Update Action Plan • Task Completion Criteria

Table 2: *Accelerating Process Improvement Steps Overview*

APIM Checklist

Table 2 provides a quick checklist/overview for each step in APIM.

APIM Truths

The SEI has long recommended allotting 18 to 24 months per CMM level. It found that the average is two years to get to SW-CMM Level 2. Watts Humphrey, founder of the Software Process Program at the SEI, recommends one to three years per level. Each organization must weigh the importance of the advantages and disadvantages based on their unique environment to determine whether to take it slow and easy or accelerate process improvement.

Table 3 (see page 8) compares some typical advantages and disadvantages of accelerating processes to taking it slow and easy.

Bottom Line

There are many lessons learned from both successful and unsuccessful process improvement efforts. The Internet is full of hard-learned lessons and provides a great tool for levying others' lessons learned. There are also many lessons learned from successful accelerated process improvement efforts. Use the tools and techniques developed and shared by others to help make your process improvement effort successful as well as the difficult lessons learned to help avoid making the same mistakes.

Coming Events

March 29-April 1
Defense Technical Information Center Annual Meeting and Training Conference
Alexandria, VA
www.dtic.mil/dtic/annualconf

March 30-31
3rd Annual Southeastern Software Engineering Conference
Huntsville, AL
www.ndia-tvc.org/SESEC

April 19-22
2004 Systems and Software Technology Conference
Salt Lake City, UT
www.stc-online.org

May 17-21
STAREAST
Orlando, FL
www.sqe.com/stareast/

May 23-28
26th International Conference on Software Engineering
Edinburgh, Scotland
www.jupiterevents.com

June 23-26
Agile Development Conference 2004
Salt Lake City, UT
www.agiledevelopment conference.com

September 27-29
4th ACM International Conference on Embedded Software
Pisa, Italy
www.emsoft.org

November 15-19
STARWEST
Anaheim, CA
www.sqe.com/starwest/

Accelerating	Slow and Easy
Quicker return on investment	Institutionalization more likely
Early success fuels improvements later	More time for improvement successes
Early failures jeopardize later efforts	Easier recovery from failures
Tendency to keep things simpler	Tendency to create bureaucracy
Less time	More time
Processes in place quicker	Processes more staggered
Requires research to levy lessons learned from other organizations	More time to learn from lessons and collect historical data
Process improvement staff needs to be both process savvy and have an agile temperament	Time to learn process improvement how-tos

Table 3: *Accelerating Process Improvement Advantages and Disadvantages*

Keeping costs reasonable and time optimal is the key advantage of APIM. This method has been proven to keep process improvement costs lower and time minimal. However, it is key to remember that you should have a small, agile process team and there should be very little project impact due to the process improvement effort. The only impact to projects should be in improving the way the projects operate. Otherwise, the cost savings will be minimal if seen at all. The bottom line is: Do not sacrifice productivity to meet process improvement goals. This will counteract any of the advantages achieved by using the APIM.

Finally, always consider, "What is the bang for the buck?" If the return from each step, form, or plan, is not worth the time it takes to do it, then it should probably not be accomplished. Do not let bureaucracy stop you from achieving your ultimate process improvement goal: Mature processes for a smoother, more effective working environment.

Whatever you do, always remain agile!◆

Additional Reading

1. Carnegie Mellon University. CMMI SM for Systems Engineering/Software Engineering, Ver. 1.1, Staged Representation. CMU/SEI-2002-TR-002. Pittsburgh, PA: Carnegie Mellon University, Dec. 2001.
2. Yourdon, Edward. Death March – The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects. Prentice Hall, 1999.
3. Humphrey, Watts. Managing the Software Process. Reading, MA: Addison-Wesley, 1990.
4. Humphrey, Watts. A Discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995.
5. Paulk, M. C., C. A. Weber, B. Curtis, and M. B. Chrissis. The Capability Maturity Model: Guidelines for Improving the Software Process. Reading, MA: Addison-Wesley, 1995.
6. McFeeley, Robert. IDEALSM: A Users Guide for Software Process Improvement. CMU/SEI-96-HB-001. Pittsburgh, PA: Carnegie Mellon University, Feb. 1996.
7. Project Management Institute <www.pmi.org>.
8. International Organization for Standardization <www.iso.ch>.
9. Software Process Improvement and Capability dEtermination <www.sqi.gu.edu.au/spice>.
10. Software Process Improvement in Regions of Europe <www.cse.dcu.ie/spire>.
11. Extreme Programming (aka, agile programming) <www.extremeprogramming.org>.

About the Author

Deb Jacobs is process improvement manager and principal engineer at Software Engineering Services. She has over 25 years experience in system/software engineering, project management, and process improvement, including helping organizations be more successful in development and management. Her notable successes include leading a successful Capability Maturity Model® (CMM®) Level 3 effort in one year, organizing struggling projects, and mentoring new managers. She is former SPINOUT newsletter editor/ originator, former CERT® [Computer Emergency Response Team] Conference chairperson, InfoTech deputy Software Tracks chair, and a Software Engineering Institute CMM IntegrationSM contributor. Jacobs has a Bachelor of Science in computer science.

Software Engineering Services
1508 JF Kennedy DR, STE 201
Bellevue, NE 68005
Phone: (402) 292-8660
E-mail: djacobs@sessolutions.com

Sample
Agile Action Plan

Monitoring and Control (Organizational) Action Plan Program Management Office Establishment

Problem Definition/Objectives/Purpose:
A consistent, repeatable method is needed for monitoring projects. The purpose of this task is to develop the processes, templates, and guidelines for management of all projects.

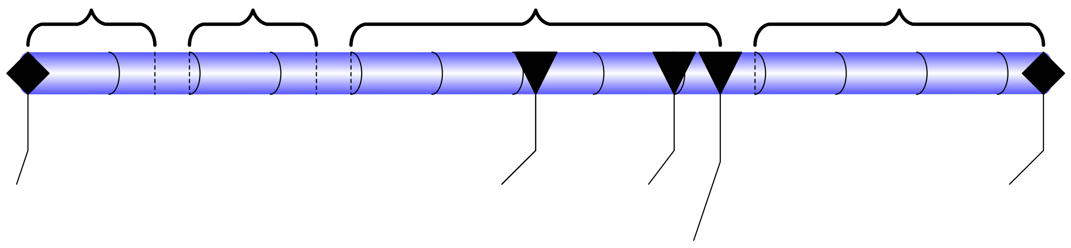
Team Members:
Rod Simpson
Jane Smith
John Jackman

Piloting Strategy:
Piloting will be accomplished by using the selected projects to accomplish the processes developed and by improving with lessons learned as we go along. The PLID project has been selected to try the process out due to their current maturity level and availability. They will try it once or twice, and then the rest of the selected piloting projects will attempt to follow the process. The processes will be tweaked along the way to make it work best for each project as well as give the management team the information they need to monitor projects.

Desired Results:
The key expected result is development of a Program Management Office with processes and templates that allow executive management to monitor projects. It also provides an opportunity to manage issues before they become issues for the customer.

Issues and Risks:
Executive Management buy-in.
Executive Management using process for statusing projects.

Timeline/Actions:



Deliverables:
Flow Charts (High Level and Detailed)
Processes
Templates
Specifics to be Determined

Using the Team Software Process in an Outsourcing Environment

Dr. Miguel A. Serrano and Dr. Carlos Montes de Oca
CIMAT

This article describes an experience using the Team Software Process^{SM} (TSP^{SM}) in an outsourcing software project. One characteristic of this type of project is that the company offering the service might have to face an economic penalization if the project is not delivered on time. This article describes how TSP techniques were used to quote the project, how TSP helped to control the project during development, and the lessons learned from this experience.

Small companies make up the majority of software development organizations; they face many challenges to maintain their business and to survive [1, 2]. In particular for outsourcing software development companies, fulfilling commitments is everything. Finishing projects on time is critical. Moreover, finishing projects within budget and with the expected quality is the whole business focus because delivering quality software on time means maintenance costs will be minimal. To be able to meet these constraints, outsourcing companies must excel in estimation, planning, project management, and quality assurance.

The Team Software Process^{SM} (TSP^{SM}) and the Personal Software Process^{SM} (PSP^{SM}) are well defined processes for software development teams and for software engineers, respectively [3, 4, 5, 6]. The TSP and PSP are designed to help teams and engineers improve their performance and to produce quality products on time and within budget.

This article describes the experience of QuarkSoft, a small, outsourcing, software development start-up company that used the TSP to run an outsourcing project. Specifically, this article describes how TSP techniques were used to quote the project before signing a contract, including estimating project cost and running and keeping the project under control. This article includes a description of the main problems and results of using the TSP, as well as a discussion of the lessons learned from this experience.

QuarkSoft offers consulting and outsourcing software development services. It was conceived as a company where quality software development is one of the main distinguishing characteristics. Thus, the company is committed to developing quality software on time and within budget. The strategy to meet these business goals includes following the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®) and its implementation with the TSP and the PSP [7]. In other words, the TSP and the PSP are used as the baseline processes in all software development projects.

The TSP provides an operational process to help software engineers do quality work. It also provides the mechanisms to maintain an effective team working environment [8, 9, 10, 11, 12]. The TSP provides team members with the forms, instructions, standards, processes, and scripts to do disciplined and effective teamwork. Teams working under the TSP start with a project launch in which the goals, strategy, risks, plan, and schedule for the entire project are addressed. Normally, the plan is decomposed into several cycles, and during the launch a detailed plan for the first cycle is defined. The project launch takes three to four days. Once the project launch is finished, the team executes the plan. The TSP teams have periodic status meetings during each cycle. At the end of the main phases and cycles, postmortem analyses are conducted. In addition, each new cycle starts with a project relaunch in which the detailed plan for the cycle is built [8].

"TSP techniques were used to quote the project before signing a contract, including estimating project cost and ... keeping the project under control."

The Project

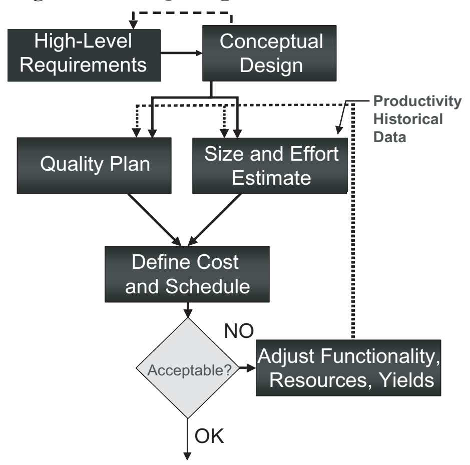
The QuarkSoft project was a major upgrade of a legacy system. It involved reengineering and a great deal of new functionality dealing with databases and compiler techniques over a distributed environment. The core part of the legacy system was the query engine (QE). The old version of the QE was developed for the DOS-Intel platform using indexed files. The client, a large international information and ratings service company, required development in less than nine months.

Getting and Negotiating the Project

The first important issue in this project was defining the development cost and schedule as well as specifying the functionality and quality that the final product would have. On one hand, it was required to quote the project before signing the development contract. On the other hand, since the TSP is used in all projects, it was necessary to perform a TSP launch to have a reasonable quote. In other words, the information to produce a quote is an outcome of a TSP launch. Unfortunately, the client did not want to start the project without the quote, and QuarkSoft could not start without a contract.

To address this deadlock situation, a TSP-based quoting process as depicted in Figure 1 was defined. This process is based on techniques used during a TSP launch. The advantage of this approach is that it requires fewer resources while keeping some of the TSP estimation practices. Clearly, some of the major disadvantages of the approach are the lack of a deep analysis of the problem and the lack

Figure 1: *The Quoting Process*



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

of a deep risk analysis.

First, a small team of two engineers was put together. One of these engineers was knowledgeable to some extent of the domain because he developed version one of the QE several years before. The other engineer was knowledgeable about technical issues such as the new platform required for the project.

Second, a conceptual design was developed; TSP estimation is based on product size. Then, using historical productivity data from previous projects, the estimate of time, effort, and quality is done. The TSP bases the initial size estimation on a conceptual design that is done during the project launch [8]. Thus, the TSP-based quoting process started by creating a conceptual design. The conceptual design was developed using the experience of the first engineer and a high-level requirements document provided by the client. Then, client domain and technical experts validated the conceptual design.

Once the conceptual design was agreed on, the estimating process started. First, the size of the product was estimated in source lines of code (SLOC). Each of the components defined in the conceptual design was estimated. The team started with the QE component. Then, other components were estimated using historical data from the client. Finally, best-guess estimation was done in the totally new components such as data base access, graphical user interface, reports, logs and security, and integration with other systems. A combination of the Wideband-Delphi [3, 13] and Standard-Component [14] estimation methods was used to get these estimates and to derive the size estimate of the whole system.

Next, the effort was estimated using estimated productivity data from QE version one and from a previous TSP project. Then, a quality proposal was developed. The quality proposal defines a target for defects/thousand SLOC during the product life. According to this objective, it is possible to define targets for the yield (i.e., percentage of defects in the program that are removed in a particular phase or group of phases [3]) of each of the phases of the development process (e.g., design and code reviews, inspections, compilations, testing). Having the yield for each of the phases, it is possible to estimate the amount of defects that will be in the product during integration test, system test, and product delivery. Thus, the team defined a target yield for each phase and estimated the defects in system and integration test. Using the TSP Quality Guidelines rates for defect removal on those phases, the team estimated the amount of time testing would take. This effort was added to the estimate.

The team came back to the client with the estimate. To meet the project time and budget restrictions, the quoting team and the client worked out a final proposal: Both sides agreed to cut about 30 percent functionality and decided that the development team would include one engineer from the client's staff. Finally, it was agreed that the project would be a nine-month effort, utilizing four engineers.

The Team

The team was assembled with the two senior engineers that made the quote, one junior engineer, and another junior engineer from the client's staff. It was agreed that the project would be developed in QuarkSoft's offices, which provided a TSP-friendly environment.

It was the first time the team members worked together. Also, this was the first TSP project for all team members except for one of the senior engineers who had participated in a one-year TSP project previously. In addition, everyone but the engineer from the client staff had been trained in the PSP. Since the TSP teams require PSP-trained team members, this engineer had to be trained before starting the project.

One practice QuarkSoft promotes is that every team/project has a distinctive name and logo, both selected by team members. The name is in the spirit of the NASA mission names. All documents are tagged with the team name and logo. The team decided to be called Maximus and designed their logo. This practice is aligned to one of the TSP objectives, that is, to form jelled teams [8]. It has been observed that the name and logo help to build team identity (and team pride). In this regard, having an engineer from another company in the team was seen as a risk in terms of forming a jelled team. However at the beginning, this risk was not considered to have high impact.

One deviation from the TSP guidelines had to be done. The team coach and the team leader roles were assigned to the same person. The reason was that the senior engineer that played both roles was the only trained TSP Launch Coach and he was the engineer who developed version one of the QE component.

Launch

After the team was formed and the PSP training was finished, the next step was to perform a TSP launch. Strictly speaking, at this point the TSP starts. The launch consists of nine meetings and lasts three to four days. The launch is the planning phase for the whole project. During the launch, the team follows strategic management principles, together with risk analysis and quality planning to produce a sound plan to develop the project.

The launch was a typical TSP launch. In meeting No. 1, client representatives gave an overview of the project and talked about its importance for the company as well as the impact on sales and customer satisfaction.

In meeting No. 2, Maximus set the project and team goals and assigned TSP roles to team members. Since the first cycle was planned to write the Software Requirements Specification (SRS), only five of the eight roles that TSP defines were assigned. The planning and quality manager roles were assigned to a junior engineer, the support manager role to one senior engineer, the process manager role to the other junior engineer, and the interface manager role to the other senior engineer. Design, implementation, and test manager roles were not assigned.

In meeting No. 3, the conceptual design developed to quote the project was used as a starting point to develop the project strategy. It was decided to implement the project in seven cycles. The first cycle would be devoted to writing the SRS and the Statement of Work (SOW). In the second cycle, a prototype would be developed to define the technology to build the project. In the third cycle, Maximus would build the High Level Design. The rest of the cycles would be for detailed design, implementation, and system integration and system test.

The rest of the launch (i.e., meetings four through nine) included the creation of the general plan for the seven cycles and the detailed planning of the first cycle (four weeks), the quality plan, and a risk analysis.

The detailed plan for the first cycle included activities to create the SRS and the SOW. To estimate the time required for developing these products, the team used historical data from a small TSP project that was finished by another team.

Two important deviations from a normal TSP launch occurred. On one side, the conceptual design, which is normally built in launch meeting No. 3, was already done while quoting the project. Nevertheless, during meeting No. 3, Maximus revisited the conceptual design and made some adjustments. On the other side, the schedule was committed before the launch, so the planning had to be done and adjusted accordingly.

During meeting No. 8 the team prepared the launch report and the presentation for the client. Finally, in launch meeting No. 9, the overall plan was presented to the client. The client was informed of the strategy to build the system, the overall plan, the deliverables, the schedule, the quality plan, and the risks. Moreover, during this meeting several issues were detected that were not considered by the client at first. These findings eased the requirements elicitation.

Running the Project

After the launch, Maximus started executing the plan for the first cycle. Maximus held weekly status meetings, and at the end of the cycle a postmortem meeting was held. Once the first cycle was finished, the rest of cycles followed the same general TSP process: a relaunch, weekly status meetings, and a postmortem [8]. During relaunches, the general plan is reevaluated using the historical data of previous cycles and the detailed plan for the new cycle is built.

Role assignment was the same during the first three cycles. On the fourth cycle, team members started to switch roles. Maximus' members explained that during the first cycles it was not easy to perform the roles. Since they were new to TSP, they required some experience before switching roles. In addition, the first cycles were devoted to building the SRS, documents, and the design. Some roles such as implementation manager do not play an important role in those cycles.

Some of the findings that Maximus got from the weekly meetings and postmortems were the following:

- During the first three cycles, Maximus had time estimation errors up to 300 percent. The most important factors for these estimation errors included lack of historical data, lack of experience in the development platform, and underestimation of the learning curve for both process and technology (e.g., Component Object Model, Distributed Component Object Model, parallel processing).
- By the end of the second month, the project was four weeks behind schedule according to the earned value prediction. This problem was discussed with the client.
- After some weeks into the project, data showed that the engineer member of the client staff was not performing well. His weekly data showed that his earned value was falling behind. Later in the project, postmortem data analysis of the fifth cycle showed that this engineer was still not working as expected and was putting the whole project in jeopardy. It was then decided to take this engineer out of the Maximus team. The decision was supported with data from the first five cycles. It was estimated that keeping the engineer working at his historical earned value rate would make it impossible to finish the project on time. In addition, it was estimated that in order to meet the delivery date, it was necessary to add two more engineers.

Consequently, Maximus had some reorganization. The team leader and coach became only the coach of the team. The other senior engineer became team leader, and two new PSP-trained engineers were added to Maximus.

The relaunch of the sixth cycle lasted one day more than normal. Maximus invested one and one half days in performing a detailed reestimation of the project. The detailed plan for cycle six included time to account for the learning curve of the two new team members. In addition, a strategy to mitigate the impact of the learning curve was devised. The strategy consisted in having specialists for each of the PSP phases. That is, a specialist in design, a specialist in coding, and specialists in testing. As a result of this relaunch, the original strategy and schedule of builds were adjusted but the delivery date remained the same.

After implementing these changes, the last cycles of the project improved considerably. Table 1 shows some final data from the project.

Lessons Learned

Getting the Contract

Using the TSP approach for project estimation helped to have a more realistic estimation of size and effort. It gave solid arguments for negotiations. The TSP-based quoting process produced the information to convince the client that to meet the time and budget restrictions it was necessary to cut down some functionality and to add more resources.

The client technical experts were sensible to the estimation process and participated on validating the conceptual design and some of the size estimates (i.e., some of their data was used). Thus, they did not have much room for trying to cut costs. The client was also sensible to the fact that the estimation was error prone. It was difficult to negotiate a 25 percent estimation error but using the data at hand, the client had no arguments to go against the proposed estimation error. In summary, the TSP-based estimation process provided the data to elaborate, support, and defend the quote for the project. In addition, it gave the client a sense that the project was estimated professionally as opposed to being obscure nonsense estimations.

Actual Size (in SLOC)	28,344
Size estimation error	2.26%
Effort estimation error	26.59%
Defect density (defects/KSLOC)	0.18
Productivity (SLOC/Hr)	6.14

Table 1: *Project Final Data*

Project Launch

One of the advantages of a TSP launch is that everybody knows and agrees to the plan. This common knowledge facilitates the communication among stakeholders and gives a common vocabulary for such a communication. According to the team leader, a major advantage of using the TSP was the planning. He said, "From the beginning, all team members know the activities that each one would perform, the sequence and dependences of them, and the time they would take."

According to one of the engineers:

> Having this detailed planning, it is possible to have better estimates, to plan time for researching the best technology and the best approach for the development of the project. In addition, each team member has a clear idea of the whole project as opposed to other projects where I have participated in which we had no idea of the context. In those projects, the good ideas start coming at the coding or testing phase, when it is very difficult to implement them. With TSP, the good ideas start coming from the launch.

The client liked the last meeting in which Maximus presented the result of the launch. The client was impressed by the TSP methodology (e.g., the level of detail of the planning and all the work products that the launch produced), and considered the team very professional.

Some of the problems reported by Maximus' members were the following:

- A formal method for doing the initial conceptual design is needed, since it is the most important part for estimating the project.
- Risk identification was not easy. Although the TSP contains a risk

analysis, it is not clear how to identify the risks.

Running the Project

Relaunch

Relaunches helped to keep the project under control and allowed detailed planning for the cycle. Relaunches were the perfect time to revisit the strategy, objectives, and risks. In addition, relaunches provided the time to make adjustments to the team and to the original plan in an orderly way. One of our favorite comments stated by one of the engineers was: "Relaunches are a fundamental part of TSP. They allow us to do detailed planning for short periods as opposed to doing detailed planning for a whole nine-month project."

Weekly Status Meetings

Weekly meetings were the best thermometer of the project, during which the team evaluated the status of the project. Weekly meetings were perhaps the most important activity to keep the project under control and to foster communication among team members. They were the best moments to identify and solve problems. Especially, a weekly meeting was a great moment to resolve dependences, misinformation, lack of information, and sort out personal issues. However, it took time to get in that type of mood.

According to Maximus team members: "At the beginning, we wanted to rush the meeting, and we left many open issues unattended."

In another comment, the team leader said:

> We checked risks and objectives weekly, but until cycle six we realized that the whole point of this activity was to generate activities to mitigate risks and to meet objectives. So, a mechanism/process to produce activities to this end is needed.

Other Lessons

Metrics and Data Analysis

One of the main advantages of using the TSP is that the team produces a great deal of information that is available at the right time, which is fundamental for decision making. Several metrics and indicators can be produced. They are really helpful for early detection of problems and for adjusting planning if necessary. For example, without the TSP data and processes, it would have been difficult to detect that the project was behind schedule and to do the analysis to take corrective actions. In addition, as data is collected, more realistic plans are built since they are based on historical data from previous cycles.

In practice, Maximus did not have the time to perform all the data analysis that the team wanted. From the beginning, the project was estimated without considering time for the roles' activities (e.g., planning manager, quality manager). Therefore, no time for data analysis was planned. Also, Maximus realized that they would have needed to develop a tool to help them do the data analysis.

Project Under Control

A major advantage of using the TSP was that the project was under control. Problems were identified on time and corrective actions were applied. Furthermore, corrective actions were backed up with historical data. For example, the decision to take one member out of the team and add two new members was backed up with data, and a strategy to do the switching was developed. The result was a relatively smooth transition. The new Maximus team worked very well from that change on. The team became more cohesive. Without the type of data and process that TSP provides, this kind of analysis and decisions would have been difficult to accomplish.

Another advantage of TSP is the visibility of the project. Every week all team members and other stakeholders know the project status, which includes aspects such as work done, problems detected, new issues, new action items, risks' status, schedule slippages, and goal accomplishment.

Although teams that follow the TSP produce a wealth of information, there is the need to improve the process and tools for issue tracking, action items tracking, and goals and risk tracking. For example, the team leader said that "although we agreed on many things during meetings, they are not done unless they are urgent."

Cultural Change

Another relevant aspect of the TSP is that it is an excellent medium to promote cultural change toward a disciplined process-oriented work method. As we mentioned before, the majority of Maximus' team members were TSP first-timers. Nevertheless, they willingly followed the process, collected data, and committed to finishing the project on time. They also committed to delivering a high quality product. They performed design reviews, code reviews, and inspections. None of them did that in the previous organizations they had worked for.

In addition, TSP promotes the formation of cohesive teams. This might be explained by the fact that they own the plan, and they own the process. While being external observers, we have seen how Maximus team members have developed a camaraderie that goes beyond the workplace. The team leader said he had noticed how the TSP fostered disciplined work in all areas.

Limitations

Maximus identified and documented several process improvement proposals (PIPs). However, Maximus members were unable to implement the PIPs due to the lack of a process to do so and a shortage of available time.

It was observed that not all of the day-to-day activities of the team members' roles were well defined. Thus, engineers had a difficult time performing effectively the roles assigned to them. The TSP provides general guidelines for each role. Also, some role activities are included in some of the TSP scripts. However, in the daily basis, the responsibilities of each role are not detailed.

The TSP does not include a deployment phase. This was a relevant issue for Maximus. After having guidelines and a process for all the development phases, going into deployment of the system without such support caused some project instability. In particular, the deployment phase was critical because the system was deployed in two different countries.

Conclusions and Recommendations

The TSP starts with size estimation. Then, the effort estimation is calculated using historical data such as productivity rates and defect insertion rates. It might be risky to share with the client the productivity rates used to estimate effort. If the client does not have a deep understanding of the TSP and the PSP estimation and planning processes, the client might assume that productivity rate is the only parameter used to estimate effort. With this belief, the client might push the estimation team to use a higher productivity rate that would lead to a reduction of effort estimation, which in turn derives in a reduction of the cost of the project. The client might argue that the productivity rates used are too conservative making it difficult to convince the client otherwise. This situation might lead to an unrealistic estimate.

If historical data is lacking, it is convenient to adjust the estimate to account for the estimation error. QuarkSoft did not have a definite number on that, but a number between 20 percent to 30 percent estimation error had been used for first-time TSP projects during initial estimation. The estimation is adjusted during each relaunch with the data collected in previous cycles. Thus, the estimation error tends to decrease after some cycles.

One important aspect is the TSP tool support. The SEI's prototype TSP tool is pretty helpful, but it lacks functionality that would make some tasks much easier. To put it in perspective, without the SEI tool, it would be very challenging to have a TSP team collecting and analyzing all the data they produce. However, the functionality for data analysis and reporting provided by the tool is still not enough.

Regarding the effect of the TSP on the client, it was observed that at the beginning, the client was excited about the TSP. The client was well impressed with the TSP process. But at the end, the only thing that mattered to the client was to deliver what was wanted on time.

In addition, it is necessary to help the client in becoming PSP/TSP literate. The TSP produces and uses a great deal of information that can be easily misunderstood and used against the development team. For example, TSP team members might be logging 30 hours/week in planned tasks. The client might demand 40 hour/weeks. However, team members might be working more than 40 hours/week if the time invested in overhead activities is accounted for.

Maximus' team members agree that the TSP is a powerful set of practices and processes. Many of the problems that they faced while using the TSP are covered by the TSP if applied completely. This suggests that team performance will improve with practice and that TSP coaching is particularly important in first-time TSP projects.

Finally, every project will eventually get into some sort of trouble. The important thing is to have the right information at the right time to make the appropriate decisions. This experience shows that the TSP can provide such information. Although TSP needs some adjustments for outsourcing projects, this experience suggests that the TSP is powerful and flexible enough to be used in outsourcing environments.◆

Acknowledgments

We want to thank Maximus' team members for sharing their experience and information. In particular, special thanks to Gerardo Borbolla (team leader), Ricardo Vazquez, and Ricardo Vidrio who provided most of the information to write this article.

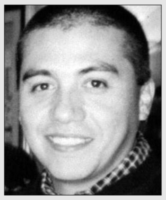
References

1. Fayad, M., M. Laitinen, and R. Ward. "Thinking Objectively: Software Engineering in the Small." Communications of the ACM 43.3 (2000): 115-118.
2. Ward, R., M. Fayad, and M. Laitinen. "Thinking Objectively: Software Process Improvement in the Small." Communications of the ACM 43.4 (2000): 105-107.
3. Humphrey, W. A Discipline for Software Engineering. Addison-Wesley, 1995.
4. Humphrey, W. "Pathways to Process Maturity: The Personal Software Process and Team Software Process." SEI Interactive 2.2 (1999): 1-17.
5. Humphrey, W. "Three Dimensions of Process Improvement. Part I: Process Maturity." CrossTalk 11.2 (1998): 14-17.
6. Kamatar, J., and W. Hayes. "An Experience Report on the Personal Software Process." IEEE Software 17.6 (2000): 85-89.
7. Davis, N. "Using the TSP to Implement the CMM." CrossTalk 6.9 (2002): 30-38.
8. Humphrey, W. Introduction to the Team Software Process. Addison Wesley Longman, 2000.
9. Humphrey, W. "Three Dimensions of Process Improvement. Part III: The Team Process." CrossTalk 11.4 (1998): 14-17.
10. Humphrey, W. "The Team Software Process (TSP)." Technical Report CMU/SEI-2000-TR-023. Pittsburgh, PA: Software Engineering Institute, 2000: 51.
11. McAndrews, D. "The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices." Technical Report CMU/SEI-2000-TR-015, ESC-TR-2000-015. Pittsburgh, PA: Software Engineering Institute, 2000: 53.
12. Webb, D., and W. Humphrey. "Using the TSP on the TaskView Project." CrossTalk 12.2 (1999): 3-10.
13. Boehm, B. Software Engineering Economics. Englewoods Cliffs, NJ: Prentice Hall, 1981.
14. Putnam, L. Measures for Excellence: Reliable Software on Time, Within Budget. Englewoods Cliffs, NJ: Yourdon Press, 1992.

About the Authors

Miguel A. Serrano, Ph.D., is a researcher in the Department of Computer Science at the CIMAT, Mexico. He is a SEI-authorized Personal Software Process^{SM} Instructor and Software Engineering Institute-trained Team Software Process^{SM} Launch Coach. He is currently working on the Capability Maturity Model® Integration (CMMI®)/Standard CMMI Assessment Method for Process Improvement track. His current research interests include software process improvement, statistical process control, and software quality. He has a Master of Science in information systems and decision sciences, a Master of Science in system science, and a doctorate degree in computer science from Louisiana State University.

Apdo. Postal 402
Guanajuato, Gto., 36000
MEXICO
Phone: +52 (473) 732 7155 ext. 49544
E-mail: masv@cimat.mx



Carlos Montes de Oca, Ph.D. is a researcher in the Department of Computer Science at the CIMAT, Mexico. He is a SEI-authorized Personal Software Process^{SM} Instructor and Software Engineering Institute-trained Team Software Process^{SM} Launch Coach. Montes de Oca has over 10 years experience in software development and management. He is involved in several TSP^{SM} and PSP^{SM} projects in both academia and industry. His current research interests include software process improvement and software quality. Montes de Oca has a doctorate degree in computer science from Louisiana State University.

Apdo. Postal 402
Guanajuato, Gto., 36000
MEXICO
Phone: +52 (473) 732 7155 ext. 49577
E-mail: moca@cimat.mx

Unlocking the Hidden Logic of Process Improvement: Peer Reviews

Marilyn Bush
Marilyn Bush Associates

The success of many of the most useful procedures recommended by the higher maturity levels of the Capability Maturity Model® (CMM®) and the CMM Integration℠ (CMMI®) depends on understanding the silent links connecting them to practices associated with Level 2. Certain activities that can seem peripheral to Level 2 goals during an assessment, in fact, prepare organizational foundations for what will become fundamental improvements. In trying to get through Level 2 as quickly as possible and neglecting the global logic of the CMM and the CMMI, companies sidestep the momentum-builders that can catapult them to world-class status. Perhaps the most significant illustration of this phenomenon concerns the function of peer reviews and the procedures associated with detecting defects early.

The success of using peer reviews can vary widely. Many companies start to use peer reviews and then abandon them. These companies introduced formal training, but it did not help – within a year or two, very few if any projects were still actively doing peer reviews. Other companies swear by peer reviews, and boast that "they have saved us millions of dollars." So why do peer reviews work in some companies and not in others? And what difference does it make? The answers have to do with organizational culture.

All of this could be gleaned from reading the Software Engineering Institute's Capability Maturity Model® (CMM®)/CMM Integration℠ (CMMI®). But you need to read between the lines, since the CMM and CMMI enumerate necessary tasks but do not always describe the logic that sequences them. Part of this hidden logic involves the fact that moving toward and achieving Level 2 involves a general organizational discipline that recognizes and deals with problems early, accepts independent quality reviews, and promotes discomfort when quality procedures are missing.

This cultural change affects far more than the tasks specifically identified in Level 2, and it is essential for getting the most out of, for example, peer reviews. Although some companies seem to be able to assemble this kind of discipline without having all the pieces of Level 2 in place (for example, if they have previously employed other kinds of quality programs such as Crosby), this kind of discipline amounts to having all the practices of Level 2 in place. Level 2 has to do with management discipline. Most Level 1 managers cannot protect their workers because their process is too chaotic. Without such protection, workers do not have the freedom to perform peer reviews effectively.

The Inner Logic of the CMM/CMMI: Technical Progress Conditions Cultural Discipline

The CMM and CMMI enumerate necessary tasks but do not always describe the logic that sequences them. Both models assume the importance of detecting defects early and then preventing them, yet these goals are not spelled out in the early stages of the maturity scale. At Level 2, all the process areas/key process areas ostensibly concern project management activities – risk assessment, creating consistent processes, etc.

Technical Culture and Organizational Culture

Technical culture in this article is defined as the sum of an organization's technical practices and methodologies. Changing the technical culture involves instituting better practices, or organizing them in a different sequence. It involves *what* employees do, not why they do it.

Organizational culture deals with the underlying values that motivate individuals as they relate to an organization and its present and future goals. In the words of Harrison Trice and Janice Beyer, cultures consist of:

> … shared sets of beliefs, values, and norms that both impel people to action and justify their actions to themselves and others. With the passage of time, [these sets of beliefs] tend to move away from the forefront of people's attention and become implicit and taken for granted. [1]

Why Maturity Levels Cannot Be Skipped

Process capability grows in stages. Key processes are only effective after prerequisite processes are stabilized. Engineering processes usually do not improve, for example, before management stabilizes the way it makes decisions. If management changes the work conditions day to day or week to week, the best processes in the world do not have a chance to succeed. Therefore, Level 2 is largely concerned with management decision processes. Further, as management discipline solidifies, so does a more general quality discipline [2].

In technical terms, at Level 2, managers learn to prepare estimates with their team rather than by themselves and methodically track actual estimates against original estimates. These actions constitute an important increase in technical rigor. They also, however, involve changing perspectives as well as changing practices. Managers are empowered by enhanced information to take corrective action early rather than late, and get into the habit of doing so when they need to. When costly problems are found in reviews, and fixed early and easily, teams start to see the benefit of independent and methodical reviewing and begin to feel discomfort when consistent processes are missing.

In other words, changes at Level 2 not only alter what people do but also how they think about it, that is, they substantively alter organizational culture. The big picture looks like this: When all members of the project are involved in planning activities, the whole team has to come to a consensus about goals and necessary quality standards in the product and the process. This can be a slow and bruising process, but it can happen in a timely manner with the proper culture and team participation. Reaching a consensus means the entire team buys in. People are less afraid to raise problems because they have survived an open discussion, and because their first priority is to fix things before it is too late. Quality comes before blame. (This is sometimes called *decriminalization*.) A culture in which problems are handled early – without the fear of blame – also accepts independent review without defensiveness.

How does all this happen? Where does it lead?

Changing the Technical Culture for Defect Detection[1]

One of the most powerful payoffs of the CMM and the CMMI has to do with the savings made possible by finding and fixing defects before testing (the technique that is used is called *peer reviews* in both the CMM and CMMI). Peer reviews offer *huge* savings when done correctly. For example, to find and fix a defect before testing costs only *one percent* of what it would cost to fix the same defect in operations. That is, to find and fix one defect found early in development might cost $100. To find and fix the same defect during integration and test can cost $1,000 to $2,000. And to find and fix the same defect post-delivery costs an organization anywhere from $10,000 to $20,000. (Fixing critical defects found post-delivery can cost much, much more [3]).

Today, software developers expect to find at least three to six post-delivery defects per thousand lines of code[2] [4]. That means a loss in the range of $50,000 to $100,00 in unnecessary costs per thousand lines of code. For a typical system that contains millions of lines of code, the unnecessary costs that can be saved are astronomical.

An operational defect, defined as a defect encountered by the product user as a failure, causes the product to malfunction relative to a product specification. (If it does not do what the user wants, it does not work.)

Since many projects now consist of anywhere between 250,000 and millions of lines of code (with programs getting more complex every day), the real costs of quality are enormous, and the real benefits of process improvement are equally high – the potential saving of hundreds of millions of dollars of preventable costs.

Levels 4 and 5 build naturally on the technical and organizational cultures conditioned by Levels 2 and 3. But peer reviews require not only a management discipline but also a cultural discipline. This kind of developing cultural discipline forms an essential element of the CMM /CMMI program of maturity.

Why Defect Detection Does Not Work Without Level 3 Maturity: Doing Peer Reviews in an Immature Organization

Even when they seem to be doing so, teams without a culture of cooperation do not really look for operational defects.

Managers who have not internalized the culture of quality scare staff away from discussing problems by quickly assigning blame. Lacking the technical discipline of planning and monitoring and control procedures, they too often rush to drop quality procedures when schedules are slipping.

Their subordinates pick up these signals and act accordingly. Finding too many defects might mean that they will be blamed for not doing their job properly. Rifkin states,

> If we fear for our jobs then we are less likely to take the chances that are inherent in performing some new action, making the inevitable mistakes. We would fear that such mistakes would count against us, and we may form a basis for poor performance and then we could lose our jobs. [5]

An operational defect, encountered by the product user as a failure, causes the product to malfunction relative to the product specification. Of course, not all defects are equally disabling. *Critical* defects render a product unusable and require immediate attention. In the case of *serious* defects, the customer's use is severely restricted. Defects of medium severity involve limitations that are not critical to overall operations.

Low severity defects permit users to circumvent the problem and use the product with slight inconvenience. For peer reviews, a determination must be made between major defects and minor defects. Critical and serious defects would be considered major defects, medium severity could be either major or minor depending on the nature of the defect, and low severity defects would fall under the category of minor defects [6].

In a Level 1 organization, finding a few defects (no matter how many more there were or whether the critical ones were caught) is *good enough*. Finding them will *impress the boss*. If too many are found, the boss will think that "we are not very good at our jobs."

The same attitudes shape the way teams in a Level 1 organization conduct peer reviews.

Managers who do not understand the culture of quality scare staff away from discussing problems. They then do not raise them because they are afraid they will be blamed for them – or for slowing down the schedule by raising them.

Without real changes in organizational discipline, peer reviews typically result in only one operational defect detected per review[3].

Consider this example. On one project, the project manager of organization X (which was Level 1) mandated that every development team use formal inspections. One team manager was sure he really did not need to do this since his team did not make mistakes (and even if they did, no one would know about them). But since he had to do it anyway, he decided to implement the order in the following way. He told the project manager his team did not have time to be trained. They would have to read the material on their own and would be directed to perform the formal inspections on Saturdays.

Needless to say his team inspection defect rate was very low. But since there was no one on the project regularly reviewing the inspection data as it was produced, this team manager was allowed to continue with his practice. When the system went into testing, over 60 percent of the defects found were from this one team's modules. It slowed the entire project schedule down by over three months, and cost the project $200,000. (Remember, the difference between one defect found in inspections versus testing is approximately $100 versus $1,000.)

Worse yet, after peer reviews produce disappointing results, companies may get discouraged and adopt a resistance to improvement that places them farther away from improved productivity than they were before.

The Cost of Doing Peer Reviews Without a Mature Organizational Culture

Organizations quickly understand how doing peer reviews can save vast amounts of money. They less frequently see the point of the intermediate activities that really make peer reviews work. Getting to Level 2 can seem to take forever, and instituting Level 2 activities can often seem pointless. In the meantime people wonder why they cannot implement more radical techniques right away. "Why wait for Level 3 to do peer reviews?" they ask. "What's the use of all the *tinkering* the CMM/CMMI requires beforehand?" "Why do you need to progress through

Name of Level 5 Assessed Company	Quality Improvement	Productivity/ Profit Improvement	Predictability Improvement	Customer Satisfaction
Telcordia Technologies Assessed Level 5 May 1999 [10]	**1992/93:** 48 Faults/Thousand Function Points (KFP). **1997:** One Fault/KFP.	**1992** Cost to customer 35-40 percent higher than in 1997 and profit margin substantially higher. Cost of testing a line of code is less than 1/3.	**1992:** Projects took two years. **1997:** Projects take six to nine months.	**1992:** 60 percent **1997:** 95 percent Link to satisfaction is that Severity 1 and 2 defects halved over two years.
Onboard Space Shuttle Software (IBM Houston) Assessed Level 5 November 1989 [11]	Two orders magnitude reduction in defects delivered/kloc.	300 percent improvement since early 1980's.	Consistently predicts costs within 10 percent of actual expenditure. (Missed one deadline in 15 years.)	No information available.
Motorola India Assessed Level 5 November 1993 [12]	50 percent of software delivered had no known defects (defect levels running at 30 defects/million lines of code).	Increased 3.5 times going from Level 3 to Level 5.	No information available.	No information available.
BAESYSTEMS, CNI Division Assessed Level 5 March 2002 [13]	Post-delivery defects at Level 5 amount to less than 0.26 defects per thousand lines of code.	CNI's process improvement costs averaged 3-5 percent of the software engineering directorate staff. Productivity has improved by 16 percent moving from Level 3 to Level 5.	On-time schedule commitments have risen to over 90 percent from Level 2 to Level 5. Cost performance has remained at or above 1.0 moving from Level 3 to Level 5 (a 26 percent cost improvement).	External Customer Satisfaction improved 9 percent in the past year.

Table 1: *Results From Companies That Have Changed Their Culture*

Maturity Levels?" "Why not just do those things that provide a real payoff?"

Alternatively, organizations often try to get away with attaining Level 2 with only a cursory independent quality review in place. Consequently, their organizations do not develop a culture that is aware of the importance of finding problems early. Organizations also sometimes delay putting in place substantive measures to aid tracking critical elements at critical times. (For example, they estimate lines of code before they have done requirements and then collect no actual data.) Hence, their numbers give no early warning during the requirements and design phases, and the organizations delay developing a culture in which identifying issues early is positively reinforced. The result: management remains in crisis management mode rather than in a proactive mode.

But teams without a culture of cooperation cannot see operation-critical defects because they do not really know how to look for them. More precisely, they are not equipped as a culture to look for them. Without the management and cultural disciple conditioned by Level 2 activities, peer reviews produce derisory results. Hewlett Packard, we know, took 10 years to reach a 25 percent adoption level [7] because they did not have a Level 2 cultural discipline in place. The resulting discouragement usually leads to companies bowing to ever-present resistance to continuing process improvement. *Once staff turns against peer reviews, they will not attempt them at all, and they hesitate to adopt further improvement measures.* Nor do most managers understand enough to explain what has happened. All that anyone sees is wasted effort and unrealized savings.

Implications for Senior Managers: Assessments

Senior managers as well as mid-level managers need to be aware of the technical and the cultural implications of the CMM/CMMI. They need to understand the value of a process and measurements that gives them real visibility early in the development process and allows them to be proactive rather than reactive. Being proactive is the key to quality.

Trice and Beyer state that,

> In order to manage cultures of work organizations successfully, managers must (1) be culturally aware – that is, they must understand and take into account what culture is and how it works; (2) know the cultures they are managing; (3) recognize and use the levers they have available to influence their organizations' cultures; (4) resolve the ethical dilemmas involved in managing cultures; and (5) be clear about whether they seek to maintain existing cultures, change existing cultures, or establish new ones. If managers understand the nature of culture, they will be better able to recognize the opportunities and constraints it poses for managerial action. [8]

In other words, executives are key to the success of implementing change (which is never merely technical change). *Without executive vision, positive change is unlikely to occur.*

Assessments are an effective method for management and practitioners to get expert insight into the organization's maturity and culture. Senior managers should not put too much stress on the numerical *grade* of an assessment and should stress instead an assessment's salutary stimulation. Organizations sometimes try to *game* an assessment, which sidesteps the self-reflection that leads to real change and hinders the growth of cultural discipline that will generate the major benefits to come.

Pretending things are better than they are does not improve things; only laying firm foundations helps. An organization's emphasis should not be "How badly did we do?", "Are we still at Level 1?", or "Who is to blame?" Rather it should ask: "What aren't we doing right and how can we fix it?" Asking these questions is already a big step toward higher maturity. Always, the greatest payoff is in heightened self-consciousness and the self-discipline that goes with it. With self-discipline the big payoffs later on are easy. Without it, they are nearly impossible [9].

Going from Level 1 to Level 2 the right way forces everyone in the organization to be more self-conscious. Middle and project managers begin to understand what to look for, what to ask for, and what the answers mean. Managers have real information and people become less frustrated. Once managers can see the road ahead, their expectations become more realistic, and so developers stop feeling they are being asked to do the impossible.

At that point, and not before, an organization is ready to do peer reviews properly.

Implications for Senior Executives: The Bottom Line

Without the hands-on and technically informed input of executives, the big decisions (not to game an assessment, to take the recommendations that come out of an assessment seriously) do not get implemented. And unless they are implemented, nothing changes.

To make this kind of decision with confidence, senior managers need to understand the huge payoff in profit margin when defects are caught or prevented before testing, and how these payoffs are tied to a changed organizational culture. Unless they personally understand the way a culture of cooperation and discipline evolves through the levels of process improvement, their organization will inevitably take all the easy ways out.

Being world class brings enormous

rewards. The charts in Table 1 illustrate four Level 5 organizations and their reported benefits. It should be noted that these organizations are in different businesses, have different product lines, and in general have different business concerns. Yet the numbers show the same order of results.

Conclusion

Peer reviews as prescribed at Level 3 by both the CMM and the CMMI are immensely profitable, but only if they are done right. When attempted too soon in a process improvement program, they not only can disappoint but also prove counterproductive. (The disappointment is so great that it interferes with further process improvement.)

Knowing when to perform peer reviews depends on understanding the silent links between practices associated with Level 2 and Level 3 of the CMM/CMMI improvement programs. Certain activities that seem peripheral to Level 2 goals, in fact, prepare the cultural foundations for more sophisticated activities. In trying to get through Level 2 as quickly as possible and neglecting the global logic of the CMM/CMMI, companies sidestep the momentum-builders that can catapult them to world-class status. This paper has discussed what some of these submerged links are.◆

References

1. Trice, H.M., and J.M. Beyer. The Cultures of Work Organizations. Englewood Cliffs, NJ: Prentice Hall, 1993: 75, 76.
2. Humphrey, Watts S. Managing the Software Process. Addison-Wesley, 1989.
3. Bush, M. Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory. Proc. of the IEEE 12th International Conference on Software Engineering, May 1990: 96-199.
4. Bush, M. Getting Started on Metrics: Jet Propulsion Laboratory Productivity and Quality. Proc. of the IEEE 12th International Conference on Software Engineering, May 1990: 133-42.
5. Rifkin, S. "Why New Software Processes Are Not Adopted." Ed. Marvin Zelkowitz. Advances in Computers Vol. 59 (2003): 22.
6. Grady, R., and Caswell, D. Software Metrics: Establishing a Company-Wide Program, Englewood Cliffs, NJ: Prentice-Hall, 1987.
7. Rifkin 12.
8. Trice 355-56.
9. Bush, M. Do CMM-Based Assessments for Internal Process Improvement Help Companies Stay More Competitive? Proc. of the European SEPG Conference, London, 1998.
10. Ahuja, Sanjiv. Process Improvement in a Rapidly Changing Business and Technical Environment. Proceedings of the European SEPG Conference, Amsterdam, 1999.
11. Billings, C., et al. "Journey to a Mature Software Process." IBM Systems Journal 33.1(1994): 46-61.
12. Pellegrino, J. "Birds of a Feather Session." SEPG Conference, 1995.
13. Howard, Peter. "Operating at Level 5." Internal BAE SYSTEMS Conference, Nov. 2002.

Notes

1. Peer reviews in the CMM and CMMI concern the defect detection, removal, correction, and verification process carried out by small groups during the *pre-test* phases of the development life cycle. The primary objective of peer reviews is to remove defects *early* in the development process. Peer reviews supplement, not substitute for major milestone reviews. A trained moderator and a group of developers (limited to about four to six people) draw from the area of the life cycle being completed to carry out peer reviews. Everyone participating should have a vested interest in the work product. Peer reviews should never be used as a tool to evaluate workers or assign blame for defects. Team members, after undertaking special training, are assigned specific roles (for example, author, reader, recorder, moderator, etc.) Checklists of questions derived from previous experience are used to fine-tune defect finding. The checklist is regularly updated. Afterwards, statistics on the number and types of defects found and the time expended by engineers on peer reviews are kept as a historical database for later trend analysis. Peer reviews enhance the development life cycle by creating shorter feedback loops. They are not tied to any specific methodology or tool. They are usually done at the end of the following phases of the development life cycle: system requirements, functional design, software requirements, architectural design, detailed design, source code, test plan, and test specification. Peer reviews in the CMM constitute a Level 3 Key Process Area. By the time an organization has achieved Level 3 maturity, one would expect it to perform peer reviews on every project. In the CMMI, peer reviews are included not in a separate process area but as a Level 3 activity found in the process area called Verification, Goal 2 [5, 6].
2. Although these figures come from a 1990 study, and no more recent study is available, their accuracy has been informally confirmed in work with dozens of companies through 2003.
3. Presentation by A. Warman at GEC Marconi SPIRE 99.

About the Author

Marilyn Bush is an independent management consultant who specializes in working with executive managers. She is one of the authors of "The Capability Maturity Model: Guidelines for Improving the Software Process" and the Software Engineering Institute's (SEI) Capability Maturity Model® (CMM®) Introductory Course, and is currently co-authoring a book on assessments. As a visiting scientist at the SEI, she helped revamp its CMM-Based Appraisal for Internal Process Improvement Assessment Method and Lead Assessor Course. She is a transition partner for the CMM Integration^{SM} (CMMI®) and is authorized to lead CMMI assessments (SCAMPI) and to teach the SEI Introduction to the CMMI. Bush, who divides her time between the United States and Europe, is an expert on how European and American software practices overlap and diverge.

Marilyn Bush Associates
8037 Seminole AVE
Philadelphia, PA 19118
Phone and Fax: (215) 248-5391

United Kingdom:
39 ST Giles
Oxford OX1 3LW
Phone and Fax: 44 1865 512289
E-mail: m.w.bush@ieee.org

A Beginner's Look at Process Improvement Documentation

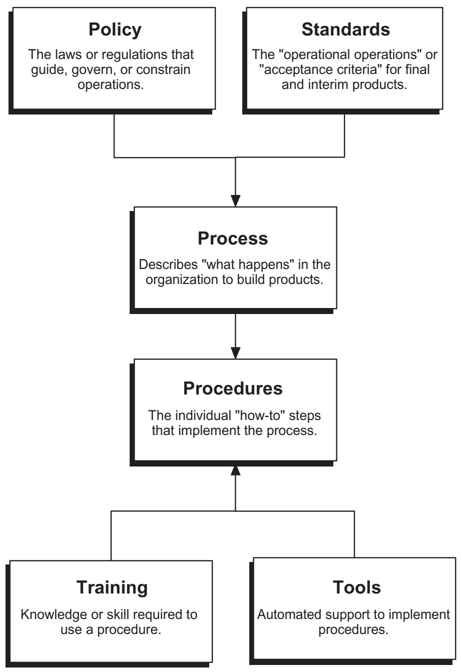
Ronald A. Starbuck
MetaVista Consulting Group

While working to get our software processes in place for process improvement, we ran across the documentation underpinning used by the Software Engineering Institute (SEI) as a guideline to implementing the Capability Maturity Model® for Software (SW-CMM®) [1] and its successor the Capability Maturity Model® Integration for Systems Engineering and Software Engineering (CMMI®-SW/SE). This underpinning is based on a very powerful yet simple structural concept useful to those just beginning the process improvement journey. By following its outline, my organization found the rewards for its improvement effort resulted in well designed and executable software processes. These are the types of processes that provide repeatability for the implementation of solid process improvement. This article looks into the fabric of this underpinning to provide a beginner's reference to understand how documentation, when done right, models process improvement for your organization.

Good processes are not just an accident! They are based on the internal associations and relationships defined and linked together by a documentation framework that can be read, interpreted, and executed by people. The framework that anchors the Capability Maturity Model® for Software (SW-CMM®) and the Capability Maturity Model® Integration (CMMI®) is derived from the structural concepts outlined in the Software Engineering Institute's (SEI) Software Process Framework (SPF) [2]. This model gives organizations the foundation they need to recognize and document the different positions and tasks used in the SW-CMM [3], and establishes the architecture needed for doing process definition.

The SPF, as we discovered, embodies what many in software development consider the foremost in *best practices* for implementing software process improvement. The architecture used in it for process definition is built on an operational framework of *process information types* comprised of policy, standards, processes, and procedures. We will explore what constitutes this process definition and the operational framework more closely in the following sections.

Figure 1: *Operational Framework*


Policy
The laws or regulations that guide, govern, or constrain operations.
Standards
The "operational operations" or "acceptance criteria" for final and interim products.
Process
Describes "what happens" in the organization to build products.
Procedures
The individual "how-to" steps that implement the process.
Training
Knowledge or skill required to use a procedure.
Tools
Automated support to implement procedures.

Defining Architecture

The process definition model used in the SPF is comprised of two very important and interrelated structural concepts. These have been around mature software development organizations for a long time and considered by many to be world-class software practices. They are the following:

- First, there is an ordering hierarchy for process information types, which defines and constrains the process activities to be performed. This is identified as the *operational framework*.
- Second, there is a standard format used for consistent construction of the process information based on a defined set of process attributes, known as *process definition criteria*.

Let us look at each of these architectural concepts in more detail.

Operational Framework

The operational framework forms the relationships and dependencies between what is to be done, by whom, and how to do it. It is a very powerful yet simple hierarchical documentation concept. The relations of this concept are shown in Figure 1. They are comprised of the different process information types, which each describes a distinct set of information about the overall process. Collectively, they provide a complete operational description of how the process is done and implemented.

At the top level of the operational framework shown in Figure 1 are the controls and discipline constructs. These prescribe the organizational policies that govern operations and establish the acceptance criteria for the developed products.

The next level of abstraction is the *what-happens-and-how* constructs. They provide detail into what processes are used to build the product and the corresponding procedures that describe how to do it.

Following these at the lowest level of abstraction are the *support constructs of tools and training*. They enable the process and procedures into action in the organization through training and tools. These support information types are not process definition constructs, but the means by which processes can be put into operation by the organization.

The following are the definitions of the process information types:

- **Policy:** The laws or regulations that govern, guide, or constrain operations. They dictate the organizational approach that governs operations. Usually these laws enforce using the organization processes by identifying the required or acceptable processes or approved ways of doing work.
- **Standards:** The operational definitions or acceptance criteria for developed products. These are the operational definitions of organizational work products constraining organizational processes by setting acceptance criteria on the output of the processes.
- **Processes:** Provide the context of *what happens* over time to build products conforming to standards in accordance with the organizational policies. Processes are constrained by the organizational policies and standards; they must specify ways to develop products that conform to organizational standards in accordance with its policies. Processes are implemented by specific procedures.

- **Procedures:** Describe the *how-to* or *step-by-step* instructions that implement the process. There are many procedures to processes. They focus on how to perform a certain task identified in a process.

The following are support types the organization uses to import processes into action:

- **Training:** The knowledge/skills required to use a procedure. Training is used to support the use of processes and procedures in the organization.
- **Tools, Automated:** Provide the support needed to implement procedures, policies, standards, processes, and training needed to build software products. Tools like training are used to support the use of processes and procedures in an organization.

Using the SPF format to construct process information types removes any confusion about where the information is found for processes or where it is supposed to go, and eliminates the bad practice of mixing process information types (i.e., policy and procedure) within the same document for an organization. This is a problem that the author's organization had to clean up in its implementation effort, and unfortunately exists in many immature process documentation implementations that you should try to avoid.

Process Definition Criteria

The criteria used for process definition supports the operational framework by defining a standard format to construct process descriptions. These descriptions are comprised of the attributes that define a set of definition elements that tie together complete process descriptions. This process information is essential for doing processes and procedures. The process attributes used by the SPF establish a solid format for consistent definition of process descriptions collectively satisfying the *what*, *who*, *when*, and *how* of processes. This concept ensures that processes and procedures are completely defined and that they convey all of the information needed to enact them by people. The basic process elements that establish the criterion are shown in the flowchart in Figure 2 and described in the following process elements descriptions:

- **Roles:** What has to be done by whom to perform the activities required for the process?
- **Entry Criteria:** The conditions that must be in place to start doing the process, i.e., software requirements must be approved before starting a design and coding process.
- **Input:** Description of work products used by process, such as allocated requirements or a developer-sanctioned standard.
- **Activities:** Description of actions that are done by the process to transform the input data into a product.
- **Outputs:** Description of work products (i.e., code, documentation, lists, etc.) that result from performing the process.
- **Exit Criteria:** How we know we are done doing the process, e.g., the responsible party approves the baseline submitted by configuration management.

In addition to the basic process elements, there are several other pieces of information useful to include in process descriptions. These aid in transforming the data into work products produced from the process. They are the following:

- **Reviews and Audits:** List of reviews and audits performed during the process. What type of audits, i.e., Physical Configuration Audit, or reviews, i.e., Preliminary Design Review, are required for the process?
- **Work Products Managed and Controlled:** List of work products (code, documentation, etc.) to be managed and controlled.
- **Measurements:** Description of process measurements, i.e., lines of code.
- **Documented Procedures:** List of activities to be completed according to a documented procedure.
- **Training:** List of training for the process, formal or on-the-job training.
- **Tools:** List of tools to support the process, i.e., automated configuration management tool or spreadsheet used during the process.

Successful Considerations for Documentation

The most important consideration we found to the success of process documentation depends on the level of detail provided to the process elements. The usability of process documentation comes down to how well these process elements are described. This will determine if they will be easily understood or are too detailed for people to use. No matter how good the architecture of the SPF, the bottom line comes down to how effectively people are able to interpret, translate, and execute the documented processes.

Process information is only as good as the process elements are described. If there is too much detail, or conversely not

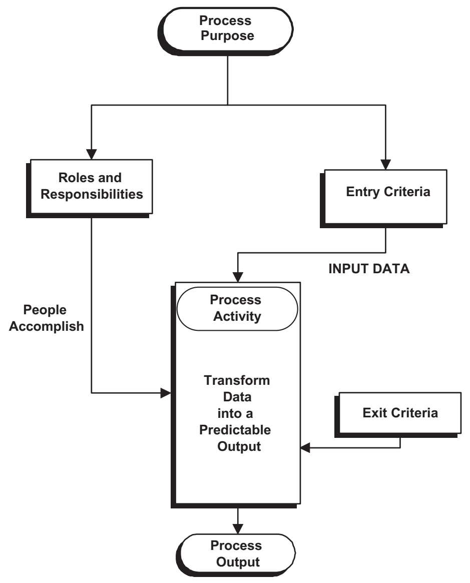


Figure 2: *Process Definition Criteria*

enough, people will have difficulty comprehending and following the processes. To ensure the success of your process documentation, the writing goal is simplicity. Process elements must be simple and clear enough to use – and not too detailed to be useable [4]. An example of this style of writing is shown in Figure 3 (see page 20). This illustration example is a configuration management process.

In this very simplistic process for configuration management, all of the process elements shown in Figure 2 were embodied. It provides readers clear, plain, and unconfused statements that inform them of the *what*, *who*, *when*, and *how* for process execution. The language used is distinct and uncomplicated. Additionally, it requires only one page to describe the process, further underscoring that being clear and uncomplicated does not require a lot of narrative pages for descriptions.

The goal is not to do one-page processes and procedures, but to determine what is appropriate and important for simple and clear process descriptions. This consideration for documenting should be given the highest priority by your organization for writing process documentation.

Factors Effecting What Is Documented

Lastly, there are a number of factors that influence how organizations finally document their processes. These factors establish the need, organizational tone for doing processes, and the funds available for who does the work. How they come together is different for every organization along with the *bottom line as* to what end the processes are documented. Organizations start out with good inten-

Configuration Management Baselining Process

Purpose of Process: To Establish Software Baselines

Roles and Responsibilities:

Configuration Manager:

- ♦ Identify what is to be baselined.

Quality Assurance:

- ♦ Verify what has been baselined is correct and current.

Entry Criteria: Life-cycle configuration milestone has been reached according to Project/System Configuration Management Plan.

Input Data: Design Progress Review, Preliminary Design Review, Test Readiness Review, or Post Deployment Support.

Process Activity: Configuration manager does following procedures:

Baseline procedure to create baseline identification in database, configuration log, and version release records.

Verification procedure to coordinate with quality assurance to ensure correct configuration has been baselined.

Status Accounting procedure to ensure project and others baseline visibility of evolving configuration is under control.

Exit Criteria: Release or identification of the following:

- ♦ Baseline Identification
- ♦ Entry into configuration log
- ♦ Database Identification completed

Process Output: Baseline established or updated

Figure 3: *Configuration Management Baselining Process*

tions for doing processes, but each of these factors has an impact on the outcome of how the documents are finally done and must be considered in your process development. The following factors are the ones the author's organization found to be important to documentation:

- **Size of the Organization:** Larger organizations have the need for more communications due to their size and the number of people involved in processes and projects. They are generally provided with a budget that supports doing processes. Conversely smaller organizations do not have the same communications problems. Because of a smaller size and fewer employees, everyone is easily informed to be able do his or her part of the job. Generally, there is little or no budget for doing processes.
- **Organizational Culture:** This is how the organization thinks and has evolved over time. Its founders have deeply seeded what this is over time by their convictions and philosophies. Organizational culture changes and evolves very slowly over time.
- **Available Budget:** This is the bottom line – funds have to be there for the organization to do anything.
- **Degree of Support for Process Definition:** This is the resource provided to do the processes in the organization. It indicates whether an organization is dedicated to full-time positions, or those extra duties in addition to a full-time job. The outcome will be different based on who is doing the process documentation implementation.

Conclusions

The author's organization found software process improvement is predicated on documentation that is defined by standard definition criteria and modeled by a structural framework. This framework forms the relationships and dependencies of operation among what is to be done, who will do it, and how it will be done.

The software process model used for the SW-CMM and the CMMI is the SPF. It is enabled by the process information types of policy, standards, processes, and procedures that are defined by definition elements that enable construction of consistent process descriptions. These concepts work together to achieve processes that are interpretable, translatable, and executable by people who do them.

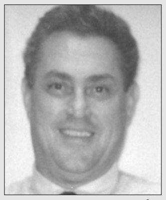
The level of detail provided to the process definition elements (criteria) is the key to the success of the process documentation. Documentation is only as good as the information provided for doing processes. There are a number of factors that influence how organizations finally document their processes. These various factors establish the need, organizational tone for doing processes, and the funds available for doing the work. How they come together is different for every organization and the *bottom lines* as to what end the processes are documented.

These are the concepts the author's organization found when doing process implementation. Hopefully, they will provide beginners with the foundation they need to understand how to model process documentation that results in successful software process improvement.◆

References

1. Paulk, Mark, et al. Capability Maturity Model® for Software, Ver. 1.1. TR CMU/SEI-93-TR-24. Pittsburgh, PA: Software Engineering Institute, 1993.
2. Software Engineering Institute. SEI Software Process Framework (SPF). CMU/SEI-94-HB-1. Pittsburgh, PA: Software Engineering Institute, Sept. 1994.
3. Caputo, Kim. CMM Implementation Guide. Addison-Wesley, 1998: 70.
4. Software Engineering Institute. Questions and Answers on the CMM. Issue No. 2. Pittsburgh, PA: Software Engineering Institute, Aug. 1994: 16.

About the Author



Ron Starbuck is a consultant with MetaVista Consulting Group. He has more than 24 years experience in all aspects of configuration management, and also in software quality assurance, process engineering, and programming. Starbuck has implemented and maintained software configuration management for large- and small-scale software efforts in both the Department of Defense and the private sector. While at the Sacramento Army Depot, he received the U.S. Army's Civil Service Achievement Medal for implementing and managing configuration management. Starbuck authored the Configuration Management Section for the Army's Test Program Set Procedures Manual. He was responsible for putting together the architecture for software configuration management in conjunction with the business rules for developing pre-processors used by the Output Technology Systems. He is cofounder of the El Dorado Hills Chapter of the Software Process Improvement Network and believes in furthering professional software development in greater Sacramento, Calif.

Meta Vista Consulting Group
Phone: (916) 933-2398
E-mail: ronstarbuc@email.msn.com

Common Errors in Large Software Development Projects

David A. Gaitros
Florida State University

During the past 40 years, numerous techniques for improving software reliability and efficiency have been developed. These techniques, when used properly, can contribute to the success of a major software development effort. However, despite these new techniques, software projects continue to fail. This article attempts to explain a few of the reasons why, despite advances in technology, a project can fall flat on its face, and what management can do to prevent these problems.

Developing software is a relatively new area of enterprise that bears little resemblance to other engineering disciplines. Although the term software engineering is widely used throughout the business, the act of creating a new piece of software can hardly be compared to the design and construction of a new building or bridge. Computer scientists are still struggling after 30 years [1] to define *software engineering* and to find the right combination of techniques, procedures, and tools that assure success in development of large complex systems.

The closest comparison I can make of software engineering to another creative process is writing a book. Give two authors the exact same subject matter; they will almost certainly generate different works. Although the subject matter may be the same and perhaps the outcome of the book similar, other aspects such as number of pages, references, organization, writing styles, and even the number of chapters would be all different.

As a creative process, writing a book shares many things in common with writing software. [2] A great deal of research as well as planning is required. Knowing the outcome of the book or the desired effect is essential for success. The book, like software, will evolve over time going through many stages of development and modification. Software, like a book, once created need not be created again, just replicated for whoever wishes to use it.

Although development of software is a complex and mysterious process to most, there are some basic management techniques that you can apply that may greatly reduce the risk of failure. In this article, I provide some general guidelines that I have gathered over the years in dealing with large and complex software development projects that have remained constant over several decades. While many books and articles focus on techniques that you should use, they often ignore the fact that mistakes can still be made during a project that will cause it to fail if certain aspects are ignored. This article will focus on some of the more common errors I have observed over years of participation in large software development projects.

Not Knowing What You Want or Need

It is not uncommon for companies and organizations to believe that the creation of a piece of software in itself will reduce costs and increase productivity. The more accurately the needs of a proposed system are defined the greater the chances of success. One of the most critical pieces of information is the proposed cost savings or benefits of the new system. Projects that involve hundreds of thousands to millions of dollars and several years in the making should not be undertaken unless there are clear and measurable objectives with provable benefits to the organization.

"To avoid the perception of bias, the project manager should not originate from any of the specific groups but should be brought in from a neutral agency."

Information management (IM) or information technology (IT) can be very expensive to develop, purchase, and maintain. Management must not only consider the cost of the software but also the hardware, infrastructure, software maintenance, facility maintenance, additional technical support, recovery procedures, alternative processing, etc.

Introducing this technology into an organization is usually done for one or more of the following reasons:

1. The resulting technology will reduce manpower/labor requirements and reduce cost by automating what was once a manual process and prone to errors. In other words, the cost of developing the system in conjunction with yearly maintenance cost is offset by the reduction of the labor force.
2. The resulting technology will increase the productivity of the current manpower/labor force. The cost of developing the system in conjunction with the yearly maintenance fees is recouped through increased profits. Since federal, state, and local government agencies do not track profits, they would be looking for an increase in their ability to service the public by offsetting the cost of system development and maintenance with the cost savings in hiring additional labor to accomplish the same tasks.
3. The software would offer a capability that previously was not available. Early in the development of computers, scientists realized their potential to perform extensive calculations. It was not until computers became reliable and fast enough that certain mathematical problems could be solved.
4. The new IT and IM technology would increase the decision-making capabilities of upper level management. There are instances where more accurate, detailed, and properly manipulated data gives senior management better resources to make critical business or strategic decisions. The cost of development and maintenance of the system is offset by increased profit/productivity through improved decisions or through avoiding costly mistakes.

Managers must understand the intended purpose of the new system before investing time, money, and other resources in the development of a completely new product. Such endeavors are extremely expensive and there are many risks involved with developing a new piece of software. The spiral software model takes into account that as a project progresses, risks are assessed at each step to ensure the product being developed is not only of good quality but will be the right project [3].

Fractured Development Teams

Often, a company or organization will involve several other organizations in the actual requirements analysis and product development. This is done to involve all those who have a stake in the outcome and to prevent alienating potential users. Several textbooks and journals I have read have never addressed this particular problem although it is more common than one might realize.

Modern software engineering techniques all focus on the assumption that the development will be accomplished by a single entity under the control of a strong management structure. The division of work and responsibilities among several organizations is often done for *political reasons*. Although one particular group may have the title of program manager, they may not have any direct authority over the other development teams. This presents several roadblocks to the successful development of a software product.

Without clear lines of communications and authority, the requirements and eventually the product development are accomplished in an inefficient fashion using different standards, different approaches, and sometimes even different technical standards. Even if all teams adhere to written standards, common interfaces, duplication of code, different interpretation of those standards, and other factors usually lead to integration problems late in the program development life cycle. The results are almost always unsatisfactory and at best it is inefficient.

One particular example typifies this problem. Several government agencies were tasked to develop a standard software package to be used by all branches of the federal government and military [4]. Although one branch was titled as the project lead, each branch that would use the end product was permitted to identify their own requirements, establish their own development teams, propose technical standards, identify existing modules to be used as an interim solution, and maintain their own cadre of contractors for support. After several years and several millions of dollars, very few lines of code had been successfully delivered.

Each different government agency perceived its requirements to be different from the other agencies, resulting in the very basic requirements of the system to be different. Different requirements dictate different specifications, which lead to different designs and even different implementation strategies. Unless great care is taken to ensure there are sufficient lines of communication among the different agencies, the products will almost always be incompatible.

To increase the chance of success, the development team should be geographically located in the same town, in the same building, and if at all possible, on the same floor [5]. Also, there must be clear and undisputed lines of authority even if some of the team members originate from other companies or groups. This clears up any ambiguity for both the development group and the customers who now have a single focal point for feedback. The customers identified with each of the groups must have equal say in the outcome of the new product, and great care should be taken to include representatives from each group in all critical phases of the project. To avoid the perception of bias, the project manager should not originate from any of the specific groups but should be brought in from a neutral agency.

"Through years of experience and mistakes, I have discovered there are no such things as generic managers capable of managing any type of project."

Lack of Experienced and Capable Management

Organization and proper planning are absolutely essential elements of any project. You would not think of hiring an experienced chef to run an automobile assembly line or likewise a software development project. Through years of experience and mistakes, I have discovered there are no such things as generic managers capable of managing any type of project. Other large software development firms have learned the same lesson.

Although the customer calls the shots as far as functionality, cost, and schedule, the person in charge of the actual software production should be an experienced data automation manager with technical knowledge in the area being developed. Hire someone with experience in dealing with the kind of system you are trying to develop. The following are some reasons for hiring such a person:

1. IT managers must be competent enough to identify and hire a qualified work force.
2. Unqualified managers too often must rely on less experienced technical staff or, worse yet, vendors on project decisions.
3. Managers must be able to assess accurately the capabilities of other shops and sub-contractors assigned to the project. Shop managers or contractors who hire inexperienced or the wrong type of personnel will find out late in the project that they must hire expensive consultants to make up the differences. An experienced manager will have a good idea what capabilities are required to complete an assignment and be aware of the risks of peer organizations that hire sub-standard personnel.
4. The manager is usually the face-to-face customer contact. He or she must converse intelligently on the impacts of alternatives and be able to address technical questions without drawing from other staff members. Customers must have confidence in management.
5. Decisions on cost, schedule, and performance cannot be delegated and must be made based upon a combination of education, experience, metrics, and the interpretation of those metrics.

Lack of Proper Work Environment

Imagine a hospital with a staff of highly trained and skilled surgeons that lacked proper operating rooms, instruments, and other medical staff for support. Operating on a patient would be hazardous at best. Likewise, software development organizations will sometimes hire very skilled software engineers and purchase expensive computer-aided software engineering (CASE) tools but ignore establishing the right development environment needed to ensure a successful project.

Having modern automated software products in conjunction with proper organization, a skilled support staff, and the right work environment can greatly increase overall productivity and improve software quality at the same time. Organizing your staff is just as important as the tools they use. Here are a few hints on setting up the proper environment:

1. Be sure to separate your development staff from computer operations. You do not want your programmers and analysts spending their time troubleshooting networks, installing servers, maintaining a database environment, creating Web pages, etc. They should spend their time using the services of the system to satisfy customer requirements. It is well worth the money to hire a few dedicated help-desk/operations staff to alleviate the burden from the rest of the workers. Part time or on-call services usually benefit smaller projects by not requiring full time employees.
2. Do not purchase top-of-the-line equipment for the development phase of your project. If software is designed on state-of-the-art equipment, chances are it will only work properly in that environment. Developers/coders/analysts should try to develop systems that run efficiently and will operate across a broad spectrum of capabilities using the worst-case scenario for standard testing and development. Provide computer programmers with the minimum configuration you would expect most of the users to possess. Be sure to test it on all target platforms, memory ranges, and operating systems.
3. Set up a separate configuration and control group that keeps close tabs on quality control and version maintenance. Individual programmers usually do a poor job of policing themselves. It is equally important that the configuration and control group not be managed by anyone in the development chain but report directly to the IT manager. This will avoid any attempts by development staff to fudge on schedules and skip important reviews and test cycles.
4. Although your hardware should be minimal in nature, do not scrimp on sophisticated tools or software engineering environments. Automatic code generators, development environments, and extensive library setups with automated configuration and control tools increase productivity tremendously. Their only drawback is the extensive amount of training required for their use.

Inexperienced or Mediocre Technical Staff

The last thing a manager needs on a project that requires creativity and ingenuity is mediocre people. A few highly experienced and educated developers can outproduce any number of average programmers. A painful lesson that I keep learning repeatedly is to hire top-notch people and empower them to do the job. Here are some hints on hiring good people:

1. Be very specific on technical experience and education when advertising for empty positions. If you need experience in specific languages, operating systems, machines, and networks with specific versions then specify them on your advertisement and do not accept less. A common mistake some companies make is to advertise for an employee with a wide range of experience never really expecting to find such a person but rather to find someone as close as possible.
2. Confirm all applicable classes/schools attended along with grades, degrees awarded, companies worked for, as well as contacting all references. Do not hire someone who has lied on his or her resumé.
3. Test the applicant. If the applicant claims to be an experienced C++ programmer, give him or her a test. Have him or her write a small complex program in the environment they claim to have experience in. You will be surprised how few are capable of passing these kinds of tests.
4. Technology changes rapidly. Look for experienced developers who continue to update their skills. A good, prospective employee will keep up with technology and be willing to prove it.

"Set up a separate configuration and control group that keeps close tabs on quality control and version maintenance. Individual programmers usually do a poor job of policing themselves."

Getting a Slow Start

The easiest place to make up for lost time is during the beginning of the project. Too often, slow and methodical project starts result in a panic race near the end to finish and deliver the software. All too often, this results in a shoddy product. This is where experienced management will be the most visible and aggressive. The first half of the project should not be devoid of meaningful deliverables. Too often, high-level models, minutes of meetings, funding expenditures rates, and personnel reports take up most of the first half of a project.

An experienced manager should be looking for results of requirements analysis, architecture plans, development strategies and schedules, technical impacts, preliminary database design, detailed models, timing diagrams for real time systems, etc. These time-consuming and important efforts cannot be put off until the latter half of the project.

Communication Requirements

Some of the most non-productive time spent on a project can be meetings. Take a tip from professional business process engineers on conducting meetings. First and foremost, each meeting must have meaning other then just a routine schedule [6]. Always have a published agenda with the specific meeting purpose stated. If upper-level management or the contract requires routine meetings, they should follow these simple guidelines:

- Have a fixed agenda and purpose.
- Always appoint someone in charge that can minimize extraneous talk.
- Track and publish decisions made during the meeting and deliver to appropriate parties.
- Adjourn the meeting when business has been conducted.
- Do not go beyond the scope of the meeting's original intent. Rather, you should conduct a smaller meeting with only the interested parties to avoid taking other individuals' time.

When coordinating project technical requirements, minimize the number of people in the meeting to essential personnel only. Make sure the people attending can satisfy all the requirements that arise during the meeting. Nothing is more frustrating than attending a meeting where key technical personnel were replaced by the next level of management who were unable to discuss the project details thus requiring another meeting. If key personnel cannot attend, postpone the meeting.

There is a huge debate going on in the industry concerning using e-mail and the Internet in the office [7]. Proponents of e-mail claim it improves communication within an organization by allowing people to transfer information anytime during the day or night ensuring that the person gets the information. Opponents claim it wastes time and productivity through abuse of the technology. Companies have restricted some office personnel to receive internal e-mails only while reserving external e-mail privileges for individuals that have bona fide requirements to do so. My recommendation is to allow e-mail throughout the organization since I believe the benefits outweigh the bad. The Internet can be a very useful tool, but it can also be a costly waste of time in both manpower and network bandwidth. Since this is primarily a research tool, I would limit access to those individuals who truly need it.

Relying Only on User Interviews for Requirements Definition

A good computer systems analyst will extensively explore other avenues of information on gathering requirements. A common error in most failed projects is using only a selected number of individuals from the user community to define accurately functional requirements. It is rare to find individuals that possess all or most of the knowledge required to develop fully a complete information technology software system. On the contrary, corporate knowledge is usually spread among several individuals and groups. Many times, crucial business processes are contained in notebooks, briefcases, a PC spreadsheet, or in a person's desk. Here are some recommendations to help resolve this:

1. Obtain copies of any legacy software system that is being used. This will give some insight into documented business practices.
2. Obtain and thoroughly read all operations manuals, pamphlets, brochures, regulations, or laws that pertain to the organization.
3. Hire your own full-time *functional experts* to assist in filling any gaps in users' requirements. Recently retired individuals are particularly useful. Companies will seldom allocate an employee full time to your development team.
4. Perform as many of the above functions as possible before your first official meeting with the customers. You will find the advanced preparation will make your first and subsequent meetings very productive.

Not Involving the User at All Stages

The most elaborate and efficient system in the world is doomed to failure if the targeted users are not enthusiastic about its arrival. A good deal of public relations work should be done at all stages of the project to ensure that users eagerly anticipate the new system. The best way to accomplish this feat is to involve as many of the users as possible in the software development stages. It is absolutely necessary to make a little bit of time and effort during each stage of the project to keep the user community informed and obtain their feedback.

"A common mistake some companies make is to advertise for an employee with a wide range of experience never really expecting to find such a person but rather to find someone as close as possible."

Cutting Short Testing

The computer industry calls its mistakes *bugs*. Rarely does a software package make it to market without bugs, which are a result of three basic acts: (1) an omitted or improperly stated requirement, (2) errors in computer logic, and (3) a performance or timing error usually associated with hardware or networks [6]. A good software development organization will have strict testing requirements to discover as many defects as possible. Each test should examine utility, correctness, robustness, and performance. Here are the usual definitions of the different testing cycles [1]:

- **Programmer Test:** Individual programmers test their specific modules against the specifications they were given. Interfaces to external modules or systems are usually simulated.
- **Software Inspections:** No one likes to have their work criticized in an open forum, but scientists have always been accustomed to having their work reviewed. Software inspections come under the category of *fault-avoidance* techniques. Code reviews, software inspections, and walkthroughs have proven to be very effective in detecting errors [6]. Experiments have shown that up to 85 percent of software faults have been detected by such techniques [8].
- **CASE Tools:** Although we mentioned CASE tools before, they can also be used to enforce programming standards. Capability Maturity Model® Level 3, 4, and 5 organizations often will introduce strict coding standards into their projects to prevent common programming mistakes.
- **Configuration and Control Test:** The configuration and control group tests whether the delivered modules meet specific programming standards set down by the company and whether the software specifications match the documentation.
- **Alpha Test:** This test is usually done by the company either at their site or at a user's location. The following features are tested: (1) deployment features, (2) interfaces to other systems, (3) upload/download procedures, (4) requirements compared against design specifications, (5) basic user functionality, and (6) performance.
- **Beta Test:** This is the first truly operational test where the new system replaces the legacy or manual system in its entirety at limited locations or sites. This is not only an extension of the Alpha Test but also a measure of whether the initial feasibility studies on the usefulness of the new system were accurate. These test results are used in determining if the new system is acceptable to the user.
- **Extended Beta Test:** After corrections or modifications on the Beta Test, the new software is released to a larger community of users before organizational or worldwide deployment. Tests should endure past several milestones such as end-of-day, end-of-week, end-of-pay cycle, calendar date rollovers, etc. to ensure adequate numbers of scenarios are tested.

Conclusion

Other engineering disciplines have the benefit of many decades if not centuries of refinement on their processes. The building of software is still in its infancy and may take several years to fully mature; we can help it along by practicing a few basic management techniques that have proven successful.

There have been many attempts by individuals to identify which step of the development process is the most critical. Some say the early stages of requirements definition, others say testing, while some conclude it is the actual development itself. I believe that all stages are critical to the success of any project. Failure can and does occur at any stage while success can be claimed only after the completion of all phases and successful product delivery to the customer. Software development is extraordinarily tedious and time consuming; minute details have been known to bring project personnel to their knees. A successful project will have top-notch management, a healthy work environment, expert technical staff dedicated to specific tasks, clear lines of communication and authority, involvement of the user community, high standards, modern development tools, and clear goals.◆

References

1. Sommerville, Ian. Software Engineering. 6th ed. Addison-Wesley, 2002: Chap. 1.
2. Hamming, R. "Mathematics on a Distant Planet." Invited Talk, 1996.
3. Boehm, B. "A Spiral Model of Software Development and Enhancement." Software Engineering Project Management, 1987: 128-142.
4. Office of the Inspector General. "Audit Defense Environmental Security Corporate Information Management Program." Project No. D2000AS-0207.000. 7 Dec. 2000.
5. Jensen, R.W. "Lessons Learned From Another Failed Software Contract." CrossTalk Sept. 2003: 25-27.
6. Bruegge, B., and A.H. Dutoit. Object-Oriented Software Engineering: Conquering Complex and Changing Systems. Prentice Hall, 28 Oct. 1999.
7. Coetzer, Dudly. "Cut Unnecessary Communication Costs." Accountancy SA Oct. 2003 <www.accountancysa.org.za/archives/2002oct/features/Limiting.htm>.
8. Fagan, M.E.. "Design and Code Inspections to Reduce Errors in Program Development." IBMB System Journal 15.3 (1976): 182-211.

About the Author

David A. Gaitros is the associate chair of the Computer Science Department at Florida State University, Tallahassee, Fla. He spent 22 years in the U.S. Air Force as a software developer and manager of large software projects as well as the associate chair for the Naval Postgraduate School, Monterey, Calif. He has worked on the Airborne Warning and Control System, the Air Force Data Systems Design Center, and various other development projects for Air Force Material Command (former Systems Command) and the Air Force Civil Engineer.

Florida State University
Department of Computer Science
261 James J. Love BLDG
Mail Code 4530
Tallahassee FL, 32306-4530
Phone: (850) 644-4055
Fax: (850) 644-0058
E-mail: gaitrosd@cs.fsu.edu

THIS COULD BE YOUR LAST ISSUE!

CrossTalk has lost its funding.

Our ability to provide CrossTalk to you at no cost and with no advertising or commercial influences may no longer be possible. However, your answers to 12 simple survey questions on our Web site will help us determine the future direction of CrossTalk and even if it continues to exist.

Also, you can easily enter our contest to win a **Nikon Coolpix 3100 3.2 MP Digital Camera.** The Nikon Coolpix 3100 creates beautiful photos up to 8" x 10" for professionals and amateurs with auto-exposure functions.

Go to <www.stsc.hill.af.mil/crosstalk> or call (801) 775-5555 or DSN 775-5555 and take five minutes to help determine CrossTalk's future. Take our survey and tell us how important CrossTalk is to you, and what subscripton price you would be willing to pay to keep it coming to your mailbox each month. The Nikon Coolpix 3100 could be yours for simply answering 12 survey questions.
**Hurry – the deadline to respond is Mar. 12th.*

* Many readers did not receive the advertisement announcing this survey attached to their February issue, so we have extended the deadline to respond to the survey and enter the Nikon Coolpix contest.

Applying Systems Thinking to Process Improvement

Michael West
Natural Systems Process Improvement

In our day-to-day work in process improvement using the Capability Maturity Model® or the Capability Maturity Model® Integration, it is easy to lose sight of the big picture. Applying systems thinking can generate breakthrough approaches to effectively improving systems development, integration, and maintenance.

The concept of *systems thinking*, which was definitively described by Peter Senge in his seminal work "The Fifth Discipline" [1], has been used by many people to investigate and resolve deep organizational problems and to achieve higher states of operational excellence. You can use systems thinking to effectively resolve many of the barriers and problems that commonly plague process improvement initiatives based on the Capability Maturity Model® (CMM®) or the CMM Integration^{SM} (CMMI®).

Losing Sight of the Forest

Working day to day in a process improvement role, it is very easy to lose sight of the *big picture* quickly. We tend to see individuals around us making independent decisions and taking seemingly unrelated actions. We get caught up in trying to deal with every separate event using a different approach and frame of reference than the last seemingly dissimilar event. It is also quite easy to lose sight of the relationships between your process improvement work and system or service delivery. Sometimes, process improvement – or worse, the model we are using, i.e., the CMM or CMMI – takes on a life of its own, and we end up doing process improvement for its own sake.

In systems thinking, Senge described two systems archetypes that provide a way of gaining a big-picture view of commonly occurring systemic problems in organizations: *fixes that backfire* and *shifting the burden*. These two archetypes are particularly useful in understanding and resolving problems that frequently plague CMM- and CMMI-based process improvement efforts.

Fixes That Backfire

In the fixes-that-backfire systems archetype, the obvious solutions are applied to problems. However, because the perceived or obvious solution is frequently applied hastily and without a thorough understanding of the problem, the result is often unintended consequences, including a worsening of the problem. One of the most pronounced examples of a fix that backfires is corporate downsizing to improve profits. In one 1991 study of 850 companies that had cut staff drastically, only 41 percent had achieved the savings they hoped for [2].

The diagram in Figure 1, known as a *causal loop diagram*, illustrates the dynamics of the fixes-that-backfire archetype as it relates to software and systems process improvement. Because the net, long-term, negative effects of the *fix* are greater than the short-term, positive effects, the reinforcing loop is the prevailing influence in the system.

"One of the most pronounced examples of a fix that backfires is corporate downsizing to improve profits. In one 1991 study of 850 companies that had cut staff drastically, only 41 percent had achieved the savings they hoped for."

In the figure, the inner loop (or the core loop) represents the organization attempting to address the problem of poor software or systems delivery by implementing model-based process improvement. The outer loop – also known in systems thinking as an *addiction loop* – represents the long-term effects of the *fixes*. When it turns out that the CMMI is not the panacea for all the organization's problems, there can be unintended negative consequences.

The fixes-that-backfire archetype has three primary manifestations in CMM and CMMI based process improvement:

1. The race to achieve a maturity level (the perceived fix) causes widespread cynicism, which in turn leads to a grass-roots resistance – a backlash – to the process improvement initiative (the unintended consequence).
2. Process discipline and improvement (the perceived fix) is done for its own sake, and the consequence is that the organization's software/systems development performance gets worse (the unintended consequence).
3. The unintended consequences of business and model disintegration occur when the cost and value of process improvement activities are not integrated with the price of the things such as products or services that are sold to a customer.

These three fixes that backfire in model-based process improvement are described in the following subsections.

The Race to Maturity Levels

Frequently, executive and senior level managers are sold on the idea that the CMM or CMMI are *the* vehicles for improving software or systems development and delivery. However, they get fixated on achieving maturity levels under the belief that maturity levels are concrete evidence that processes have improved. The maturity level becomes the evidence that the organization has improved its efficiency, effectiveness, and quality. Such beliefs are as faulty as assuming a student has actually learned something because he received an *A* in class.

Acting on such beliefs, executives will sometimes construct incentives such as bonus programs for senior and mid-level managers to achieve maturity levels in their respective sub-organizations. Now, both level-envy, and the ensuing race to achieve levels, is on! The focus on maturity levels drives people to look for quick

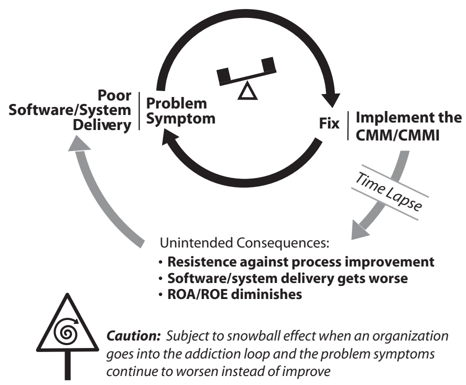

Poor Software/System Delivery
Problem Symptom
Fix
Implement the CMM/CMMI
Time Lapse
Unintended Consequences:
• Resistence against process improvement
• Software/system delivery gets worse
• ROA/ROE diminishes
Caution: Subject to snowball effect when an organization goes into the addiction loop and the problem symptoms continue to worsen instead of improve

Figure 1: *Fixes-That-Backfire Archetype in Process Improvement*

fixes that will enable them to *pass* an assessment. The symptoms – observable behaviors and artifacts – most commonly associated with a race to maturity levels are the following:

- Deadlines are established for achieving a maturity level without any estimating or project planning that supports the deadline (which, by definition, is a low maturity behavior).
- Slogans appear such as "Level 5 in '05."
- People read the CMM or CMMI, and learn to recite the model's terms and phraseology.
- Policies and procedures are rapidly created, and frequently mimic the practices in the CMM or CMMI instead of defining the organization's processes.
- Project managers, under pressure from upper level management, create elaborate project documentation that satisfies the letter but not the intent of the model. However, they promptly shelve the project documentation and do not use it to manage their projects.

The problem worsens in a large organization in which various sub-organizations are trying to achieve maturity levels independent of each other. The sub-organizations' respective managers too often let their egos and competitive natures get the better of them, and try to outdo each other to be the first to achieve the targeted level.

Once organizations have become entrenched in the level race (or perhaps more accurately, *level wars*), they are in the addiction loop, and you can count on rationale and reason often being abandoned. The organization takes a short trajectory to the maturity level, which it more often than not achieves – one way or another.

What is the result of the unintended consequences? Those incentives for achieving maturity levels usually stop at the mid-level manager, and almost never make it down to anyone doing the work, including software engineering process group members and project managers who have done the bulk of the work in the death-march process improvement project. For a brief time following *passing* the assessment, everyone in the victorious organization is exuberant. But once the assessment *high* wears off, people start looking around for some lasting and meaningful results of their work. Yet for reasons they cannot always comprehend, everything looks and feels the same as it did before the maturity level race.

Even in extreme command-and-control work environments where the primary motivation is fear (usually of losing your job or being marginalized), fixes that backfire eventually take their toll on the morale and momentum of process improvement initiatives. Smart, skilled, process-oriented people start to look outside the organization for more meaningful, rewarding work. Project managers and engineers become jaded on the whole idea of model-based process improvement. They will continue giving a gratuitous *salute* or lip service for fear of not doing so, but they will be burned out on process. Worse yet, they may perceive – perhaps accurately – that their organization's CMM/CMMI effort is a waste of time and money.

The organization that chases maturity levels for their own sake, and fails to set business goals for process improvement, will spend hundreds of thousands or even millions of dollars on model-based process improvement, and can end up with nothing more to show for their efforts than a few gratified egos.

Process for Its Own Sake

Sometimes, organizations do not get too wrapped up in CMM/CMMI maturity levels, yet process still becomes the be-all and end-all to fixing every issue plaguing the organization. For many of my years in Xerox, *it's the process*, or *fix the process* became the only politically acceptable approach to any problem. The primary fallacy of this approach is that it ignores the observable, measurable fact that there really are people and accountability problems, or technology problems that cannot be resolved by addressing only the process.

A classic example of a backfire from applying a fix occurs when organizations attempt to apply the entire CMMI to traditional information technology, systems maintenance, or engineering services shops. With intelligent interpretation and tailoring, many of the CMMI practices can be applied to improve work in these environments. But the implementation of processes and procedures that are nothing but a regurgitation of the CMMI in these environments results in burdening the organization with process overhead that does not add value, thus making the organization less effective and less efficient than they were without the CMMI.

Unintended Consequences of Business and Model Disintegration

Businesses incur unintended and negative consequences when process improvement (and probably other internal *improvement* or quality initiatives, a.k.a., *solutions*) are treated as infrastructure or overhead. As these internal initiatives grow, so does the percentage of the organization's employees whose work does not directly produce something that is sold to or adds value to what is sold to a customer (the consequence). Thus, as William Bergquist noted in his book "The Post Modern Organization" [3], it becomes increasingly difficult for the expanding organization to achieve and maintain its profit goals because operating and overhead costs grow at a faster rate than that of realistically achievable revenue.

The approach some organizations take to keep the cost of process improvement down is to hide it. People in charge of process improvement or CMMI efforts in large organizations have actually presented at conferences in which they proudly announced that people worked unpaid overtime on nights and weekends to achieve a maturity level. While this may seem heroic and laudable on the surface, even a layperson can easily see that the maturity level was not truly deserved because of the obvious Level 1 behaviors exhibited in Organization Process Focus.

Strategies for Fixes That Backfire

Here are some strategies you can employ to prevent or mitigate the effects of implementing fixes that backfire:

- In planning the process improvement effort, ensure the plans include achieving measurable or observable business goals in addition to achieving maturity levels. Make sure that reporting progress or success includes status against all the process improvement goals and not just the number or percentage of practices satisfied.
- Increase awareness, especially among senior and executive managers, of the unintended consequences of chasing maturity levels. In 2001, I was involved

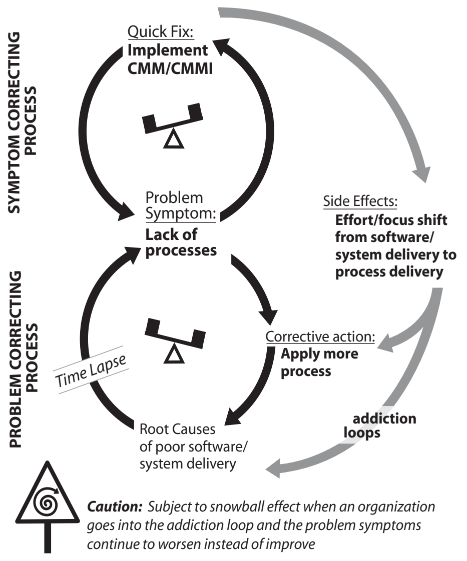

SYMPTOM CORRECTING PROCESS
PROBLEM CORRECTING PROCESS
Quick Fix: Implement CMM/CMMI
Problem Symptom: Lack of processes
Side Effects: Effort/focus shift from software/ system delivery to process delivery
Corrective action: Apply more process
Time Lapse
addiction loops
Root Causes of poor software/ system delivery
Caution: Subject to snowball effect when an organization goes into the addiction loop and the problem symptoms continue to worsen instead of improve

Figure 2: *Shifting the Burden Archetype in Process Improvement*

in an email conversation with senior level managers and marketing people in Computer Sciences Corporation who were trying to come up with strategies for countering the *CMM level race* approach of one of our biggest competitors. I sent out a note thinking that it would be career limiting because it went directly against the prevailing beliefs. Much to my surprise, a senior marketer read and understood my note, and invited me into further conversations on how we could market real process improvement benefits without getting into level wars with our competition.

- Spend the time understanding the problem. You do not necessarily have to conduct time-consuming formal root-cause analysis, which can often lead to *analysis paralysis*, but you cannot afford to continue applying solutions to symptoms, only to have the root problem perpetuate or worsen.
- Establish alliances or relationships with people in your marketing and sales organizations. Find ways to defray some of the cost of process improvement activities by selling (and charging for and collecting) the benefits of process improvement. With the right approach, a customer or client will be willing to pay a higher price for the goods or services so long as you have convinced them that there is greater value.

Shifting the Burden

According to Senge's "Systems Thinking,"

> Shifting the burden … usually begins with a problem (symptom) that prompts someone to intervene and solve it. The solution (or solutions) are obvious and immediate; they relieve the problem symptom quickly. But they divert attention away from the real or fundamental source of the problem, which becomes worse as less attention is paid to it. This forces the perception that there is no other way out except the symptomatic solution. [1]

Shifting the burden to process improvement is illustrated in Figure 2.

In this systems archetype, the perceived problem is a lack of process or process discipline in the organization, or that the organization does not have a CMM or CMMI maturity level. So the obvious, or solution, is to implement the CMM or CMMI. In the short term, the fix does appear to address the perceived problem: lack of process is replaced with process.

The unobserved yet insidious effect is that effort and focus shifts from product or service delivery (or integration) to the symptom of inadequate processes that in turn, either does nothing to improve product/service delivery, or hurts it by burdening the existing delivery processes with overly bureaucratic standards and procedures.

Table 1 identifies some common systems development and delivery problems (in the left column). The center column identifies how these problems often are perceived as process (or lack of process) problems. The third column identifies other possible root causes of the problem, which may have little to do with process discipline or maturity levels.

What are the unintended conse-

Table 1: *Common Problems Attributed to Lack of Process That May Have Other Causes*

Software/Systems Delivery Problems	Perceived as a Process Problem	Potential Real or Root Problems Not Addressed due to Shift
Projects experience scope-creep.	• Inadequate requirements management process.	• Poor business relationship with customer. • Lack of discipline in engineering staff. • There is no release strategy. • No strategy/process for insertion of new technology. • Lack of standards for acceptance or decline of requirements. • The Organization does not understand the market it is in.
Projects overrun cost and schedule.	• No estimating or planning process. • Project plans are not adequately documented.	• Culture encourages *low bid*; accurate bidders don't get work. • *Schedule* is synonymous with *plan.* • Poor business relationship with customer. • There is no release strategy. • Staff has inadequate skills to do the work. • The term *Project* is not defined. • Management does not perceive planning activities as work/progress.
Product quality (e.g., defect density) is poor.	• No quality process. • No people assigned to inspect/audit the quality.	• The organization has no defined standards or criteria that define *quality*. • Management and the culture rewards fast and cheap; good is not encouraged or rewarded. • Staff has inadequate skills and resources to produce quality work.
No amount of process improvement activity seems to improve the *bottom line.*	• People will not buy into the PI initiative. • Not enough in-house CMM/CMMI expertise. • Clients do not value the process improvement efforts.	• Misalignment between the chosen model and the organization's core business and business goals. • There are no baseline performance or capability measurements with which improvement could be ascertained. Improvement is anectdotal.

quences of shifting the burden to process? In many cases, doing so can have a compounding, double negative impact on the organization. With resources diverted from the real problems, the real problems get obscured, are given a lower priority, or are ignored, which diminishes the chance of them being resolved. Worse yet, the process improvement efforts – which can be quite expensive – may not only have no discernable effect on the symptoms, they may exacerbate the root cause.

Take the situation in which the outsourced contractor has a poor relationship with the customer or prime contractor. Shifting the burden to process by throwing the CMMI, ISO, Six Sigma, Theory of Constraints, or some other currently popular initiative at the perceived problem can irritate the customer even more, further worsening the relationship.

Strategies for Dealing With Shifting the Burden to Process

If you are fortunate enough to get involved in a process improvement initiative at its inception, make every effort to get everyone involved in discussing the business problems they want to address, and how the perceived process solutions will address those problems. As with fixes that backfire, spend time first understanding the problem to be resolved or the business goals to be achieved. If the correlation (or better yet, the causation) between business problems and process solutions cannot be clearly established, encourage people to consider pursuing alternative solutions such as organization-wide skill improvement, new technology, or fundamental shifts in business strategy.

If you get involved in a process improvement initiative after it is already underway, be persistent in questioning people around you about their proposed process solutions. Constantly ask questions such as, "Why are you doing this?" "What problem will this solve?" You may become quite annoying to some people, but after a while, you will have them asking the same questions at least quietly to themselves if not overtly.

Again, do not presume that your organization is the first ever to try to apply process solutions to its problems or goals. Conduct benchmarking activities with other organizations to find out what has worked and what has not. Process people are just as susceptible to the pitfalls of the *not-invented-here* syndrome as are engineers and other technical people.

Why Systems Thinking?

Modern software or systems organizations are themselves a system of systems. There are people (social systems), tools and technology (environmental systems), and policies and processes (process systems). The three systems – people, tools, and processes – are inextricably interwoven, and changing one without considering the interrelationships can cause fixes that backfire, do not resolve the original problems, or inadvertently make the problems worse.

The greatest unintended consequence of applying a CMMI or process solution to a non-process problem is too often the vast waste of resources used for the *faux fix*. If you really want to improve things in your organization, start by improving the process of process improvement. You can save your organization money and aggravation by using this systemic approach.◆

References

1. Senge, et al. The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization. Currency-Doubleday, 1994.
2. Boroughs, Don. L. "Amputating Assets: Companies That Slash Jobs Often End Up with More Problems Than Profits." US News and World Report 4 May 1992.
3. Bergquist, William,. The Post Modern Organization: Mastering the Art of Irreversible Change. Jossey-Bass, Inc., 1993.

About the Author

Michael West is co-founder of the consulting firm Natural Systems Process Improvement (Natural SPI). He has more than 22 years in software and systems engineering management and model-based process improvement. Natural SPI has employed non-traditional, highly effective approaches to helping clients use the Capability Maturity Model® or the Capability Maturity Model® Integration (CMMI®) to achieve measurable business results. This article is excerpted from West's book "Real Process Improvement Using the CMMI."

Natural Systems
Process Improvement
Phone: (866) 648-5508
E-mail: michael@naturalspi.com

When Is It Cost Effective to Use Formal Software Inspections?

Bob McCann
Lockheed Martin Integrated Systems Solutions

The purpose of this article is to present a way quantitatively to determine the parametric limits to cost effectiveness of software inspections based on a previously published model. This analysis leads to the conclusion that it is cost effective to inspect both original code and most modifications to the code after initial coding. Any exceptions should be carefully considered based on quantitative analysis of the projected impact of the exceptions.

In this author's experience, the two following issues are discussed qualitatively when program management decides what to inspect and what data to collect:

- Deciding whether or not to inspect work products based on a qualitative understanding of various limiting parameters such as work product size, preparation rate, or expected defect density.
- Deciding whether or not to collect and to analyze inspection data based on a qualitative understanding of the return on investment (ROI) on collecting, analyzing, and using inspection metrics.

This article improves on this practice by estimating quantitatively, from the perspective of an existing quantitative cost model [1], both the parametric boundaries of inspection cost effectiveness and ROI for collecting and analyzing inspection metrics.

It is important to note that there are several different budgetary strategies that may be applied when making process implementation choices. Here are three examples ordered from long term to short term in the planning horizon:

- Minimize total cost of ownership, including post delivery maintenance costs.
- Minimize overall development cost excluding post delivery maintenance costs.
- Assure that inspection overhead costs are exactly balanced by reductions in test rework costs (this does not minimize costs).

A simple condition is derived that explicitly calculates whether or not inspections, if performed well, will be cost effective. Specifically they are cost effective if:

$$2 \times X \leq Y + C/(S \times Y)$$

(See the on-line article for symbol definitions.)

Due to space constraints, CrossTalk *was not able to publish this article in its entirety. However, it can be viewed in this month's issue on our Web site at <www.stsc.hill.af.mil/crosstalk> along with back issues of* CrossTalk.

STC
Systems & Software Technology Conference

19 — 22 APRIL 2004 • SALT PALACE CONVENTION CENTER • SALT LAKE CITY, UT

SSTC is pleased to feature the following guest speakers in the 2004 agenda:

Opening General Session	**Sue C. Payton**, Deputy Under Secretary of Defense (Advanced Systems and Concepts) **Jon S. Ogg**, Director, Engineering and Technical Management Directorate, AFMC "A Command Perspective: The Need for Speed"
Luncheon #1	**Steve McConnell**, Chief Software Engineer, Construx Software "Code Complete 2: A Decade of Advances in Software Construction"
Luncheon #2	**LTG Keith Kellogg**, USA (ret.) (invited), Senior Vice President, Homeland Security Solutions, Oracle Corp. "Tales from the Front Lines"
Luncheon #3	**Gregory S. Shelton**, Vice President, Engineering, Technology, Manufacturing, and Quality, Raytheon Co. "Bringing Key Advanced Software Technologies to America's Defense"
Tuesday Plenary	**Co-Sponsors Panel Discussion**
Wednesday Plenary	**U.S. Government's Top 5 Quality Software Projects for 2003** Sponsored by the Secretary of Defense
Thursday Plenary	**Dr. Charles J. Holland** Deputy Under Secretary of Defense (Science and Technology) "Challenges in Delivering Complex Warfighting Functionality in Computer-Based Systems and Related Technology Investments"
Closing General Session	**Bill Neugent**, Chief Engineer for Cybersecurity, The MITRE Corp. "Cyberterrorism: We're Toast"

IEEE Computer Society CSDP Preparation Course and Examination

SSTC once again is partnering with the IEEE Computer Society to offer the preparation course and examination for the Certified Software Development Professional (CSDP) program at SSTC 2004. The CSDP is the Computer Society's certification program for software professionals developed by industry experts. The CSDP credential is intended for software engineers, software developers, software program managers, and other professionals. The CSDP is the only certification for computing professionals that carries the brand, reputation, and standards of the IEEE Computer Society. Complete details about CSDP are available at http://www.computer.org/certification. Register for the CSDP course online at www.stc-online.org.

Trade Show

Don't miss SSTC 2004's accompanying trade show, providing 180+ exhibitors the opportunity to showcase the latest in systems and software technology, products, and services. This year's schedule has been adjusted to allow participants more time to interact with the vendors without conflicting with conference presentations.

For additional conference, registration, and exhibitor information, visit our Web site
www.stc-online.org

General Information stcinfo@ext.usu.edu 435-797-0423	**Technical Content Inquiries** stc@hill.af.mil 801-777-9828
Trade Show Inquiries stcexhibits@ext.usu.edu 435-797-0047	**Media Relations** stcmedia@ext.usu.edu 435-797-0089

Who Moved My Job?

In reading information technology publications lately, it is hard to find any technology in the chatter. Most talk is around globalization: H-1B and L-1 visas, offshore outsourcing, and jobs and the lack thereof. If Chicken Little were a software engineer, she would feel right at home. Jobs, like the sky, are falling. Dot-com, happy-go-lucky, techno-entrepreneurs have transformed into puerile quibblers.

Somebody not only moved their cheese, they took it offshore, sliced it up, and served it on crackers, rice, and curry. Many feel offshore outsourcing will send the country to hell in a data dump. Yet their solutions seem to bite the hand that feeds them.

What is going on here? Have you forgotten which side of the parity bit you reside? I suggest before you add more whine to your cheese, you take a good look at yourself, your country, and the freedoms you enjoy. To help, I offer a little ode called "Who Moved My Job?"

First, there was hardware with not much to share and a bit too unbending. Software came along, saw what was wrong, now changes are never ending. It worked like a dream, or so it seemed until modifications became expensive. You got upset about all the debt, and we just got defensive.

We asked for grace and a little more space to house our growing child. Memory filled, the disk over spilled and the infant was now teen wild. The coffers were plump with data you dumped and asked for its safe keep. We stored it away but to our dismay, it was lost in a collosal heap.

We gave you a voice on system choice as long as it was built by Bill. It was not the best but who would know to test the system with much skill? After you paid, your voice did fade and your choice evaporated. We had you hooked on the windows look and did not wish to be debated.

We thought not to annoy when we started to deploy languages by the score. I guess we were wrong and projects prolonged as our languages went to war. Then there was the CASE of automated haste and promised investments returned. It never left crate, crushed by its weight, leaving you again, burned.

As the dust settles, we started to mettle and found out an error we created. Panic and fear spread with good cheer for time had become outdated. We fleeced your stockpile and sent rank-and-file to solve the Y2K pimple. Quick to the task, adding a two-digit mask and saying, "Gee, that was simple?"

Our projects were late, budgets overweight and projections often unreliable. Yet our average intrigued if applied to big leagues would be all-star and very pliable. It is not our fault that the wants you exalt lack detail and clarity. We do what you say and mold your clay then look at it in all hilarity.

We gave you extreme, the CMM regime and the manifesto of agile; also came environments paperless, networks wireless, and applications that were blue-screen fragile. We have wi-fi, cubical sci-fi, and a plethora of spam. Jobs reborn, lots of porn, and a case of identity scam.

We were the butt of your jokes, drank all your Cokes, and programmed with Java Beans. Put forth virus fears, multiple hits on Britney Spears, and raised money for Governor Dean. We know we play and seldom display the air of proper decorum. Our maturity is obvious, while slightly acclivitous, sustained by pervasive cockalorum.

Just give us our jobs decorously robbed; I know we can do much better than Mohandas Gandhi, Sean O'Leary, or even Eddie Vedder. If it's maturity you seek, we can be cool and chic, and get you justification. For the market is there, and they will declare that we have the right certification.

When business was good and venture misunderstood, we loved your capitalist schemes. We were in big demand, six-figure offers were bland, and we bought the house of our dreams. Now why would you turn on your pals hard to learn and force us to compete with another? I know they are cheap but why would you leap when we treated you like a brother?

I knew the free trade would soon fade; can you help me with a tariff? Workers will unite to give you a fright and elect a union sheriff. Yet, it does not seem right to rig the fight and plunder the dreams of the future, by forcing your hand and squelching demand with a protective dissolvable suture.

What hypocrisy, when we export over sea, truths that got us here. Democracy, capitalism, and work ethic, over there we stand to fear. A job guaranteed was not in the creed I truly must confess – just life, liberty, and the pursuit of happiness.

– Gary Petersen
Shim Enterprise, Inc.

Can You BackTalk?

Here is your chance to make your point, even if it is a bit tongue-in-cheek, without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CrossTalk, we also accept articles for the BackTalk column. BackTalk articles should provide a concise, clever, humorous, and insightful article on the software engineering profession or industry or a portion of it. Your BackTalk article should be entertaining and clever or original in concept, design, or delivery. The length should not exceed 750 words.

For a complete author's packet detailing how to submit your BackTalk article, visit our Web site at <www.stsc.hill.af.mil>.

SSTC Systems & Software Technology Conference

19 - 22 April 2004 • Salt Lake City, UT

SOFTWARE ENGINEER PROGRAM MANAGER
COMPUTER SPECIALIST CONSULTANT
PROJECT MANAGER SYSTEM ENGINEER
SOFTWARE MANAGER DIRECTOR

The goal of SSTC is to provide a forum for systems and software professionals in the Department of Defense (DoD), related industries, and academia to:

Learn
by increasing the understanding of scientific and technical issues relevant to the mission of the DoD

Connect
with over 2,500 attendees to exchange lessons learned in the acquisition, development, support, and management of software intensive systems.

Discover
effective system and software technologies

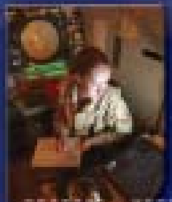
Register today!

See our advertisement inside this issue for guest speaker line-up

For registration, conference, and trade show information visit our Web site or call

www.stc-online.org

800-538-2663



ACQUISITION HOMELAND SECURITY TESTING METRICS
SYSTEMS ENGINEERING PROCESSES SECURITY REQUIREMENTS
EFFECTIVE DEVELOPMENT & METHODS NET CENTRIC ARCHITECTURES
SOFTWARE INTENSIVE SYSTEMS ADVANCED METHODS

Source Code: CT5

The premier systems & software technology conference co-sponsored by:

United States Army	United States Marine Corps	United States Navy	Department of Navy	United States Air Force	Defense Information Systems Agency

S T S C

Published by the Software Technology Support Center (STSC)

CrossTalk / MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737