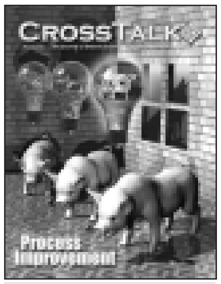
CrossTalk

March 2001 The Journal of Defense Software Engineering [illegible]

Process Improvement



On the Cover: Kent Bingham, Digital Illustration and Design, is a self-taught graphic artist/designer who freelances print and Web design projects.

Departments

CrossTalk

Sponsor	*Lt. Col. Glenn A. Palmer*
Publisher	*Tracy Stauder*
Associate Publisher	*Elizabeth Starrett*
Managing Editor	*Pam Bowers*
Associate Editor/Layout	*Matthew Welker*
Associate Editor/Features	*Heather Winward*
Graphic Designer	*Abby Hall*
Voice	801-586-0095
Fax	801-777-5633
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/Crosstalk/crostalk.html
CRSIP Online	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 31.

Ogden ALC/TISE
7278 Fourth St.
Hill AFB, Utah 84056-5205

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: at 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.

Publisher Leaves a Process Improvement Legacy

Process Improvement has always been a key interest for Reuel "Rudy" S. Alder. In fact, process improvement was so important to him that in May of 1988, he was one of the founders of the Air Force's Software Technology Support Center (STSC). There were only five members when the STSC first started, but these five people sat down together and did some planning on how to achieve their goal of sharing process improvement ideas with the rest of the software community. Their successful ideas helped create one of the leading organizations for software process improvement. One of those initial ideas was to publish a newsletter with process improvement information. CrossTalk started with about four pages of articles and 200 subscribers, but the useful information combined with the great price [free] generated a much larger audience during the years. CrossTalk now has more than 18,000 subscribers, and we estimate that we have an additional 20,000 on-line readers.

Rudy has been involved with CrossTalk through this entire effort. While he will continue to help lead STSC efforts, he is moving out of the position of publisher. As a result, Tracy Stauder is moving into the publisher's position. Tracy is not new to CrossTalk; she was the associate publisher from 1996 through 1999. She also believes in process improvement; one of her first tasks as associate publisher was to work with Rudy to lead CrossTalk's own process improvement effort. They started this effort by restructuring and documenting the journal's development process. Before this process was created, we never knew when the next issue would be published until it was actually sent to the printer. Now with a clear process that starts months in advance for each issue, we know that CrossTalk will be published each month and are confident that the articles will contain useful information. Our process improvement efforts continue with planning new and exciting issue themes and implementing color covers.

By the way, I'm not totally new to CrossTalk myself. I have had three articles published, written a Publisher's Note, and helped with other supporting functions to the CrossTalk staff.

We start this month's issue with *Perspectives on the Software Engineering Process Group.* Change is not easy; it takes exceptional people to effectively lead a process improvement effort. This article includes several perspectives of and from the Software Engineering Process Group at the Defense Finance and Accounting Service. *Raytheon Stands Firm on Benefits of Process Improvement* is an interview that delves into process improvement successes gained on the path to Level 5 while undergoing three mergers.

Process improvement can take many avenues, and Mike Evans discusses a process improvement method developed by the U.S. Navy's Software Program Manager's Network in *SPMN Director Identifies 16 Critical Software Practices* and Don Lucero and Fred Hall discuss their practical approach to software measurement in *The Best Measurement Tool Is Your Telephone.*

In our December issue, we promised you a contrasting point of view when we presented Don Reifer et al.'s, *Is Ada Dead or Alive within the Weapons System World?* Now we deliver on that promise with Ben Brosgol's *Ada in the 21st Century* and a few letters to the editor.

We always welcome readers' comments on the content of the journal and how we can improve. We will continue to improve under CrossTalk's new leadership.

Elizabeth Starrett

Elizabeth Starrett
Associate Publisher

Perspectives on the Software Engineering Process Group

This article looks at the Software Engineering Process Group (SEPG) and its role from several different perspectives within an organization. The article centers on the SEPG at the Defense Finance and Accounting Service, Technology Services Organization (TSO), Kansas City, Mo. The authors represent the director, vice deputy director of the TSO; the project officer of the Marine Corps Total Force System; two experienced SEPG members; and two brand new SEPG members. Each discusses his or her view of the SEPG's role and its activities within the organization.

The Software Engineering Process Group (SEPG) in any organization serves numerous roles from educator to organizational counselor. Its roles change with circumstances and audiences. The SEPG must listen to concerns and successes at all levels of the organization. From these conversations, the SEPG must become a communication channel between different groups and different levels from senior management to practitioner. In performing their duties, the SEPG interacts with people of wide ranging backgrounds. Its success and the success of process improvement efforts depend on its effectiveness with these different groups and activities. How the SEPG is perceived by the organization and its interactions with the organization are vital to process improvement success.

In this article, individuals from the Defense Finance and Accounting Service (DFAS), Technology Services Organization (TSO), Kansas City, Mo., will discuss their perception of the SEPG and the role it fills within the organization.

Experienced SEPG Member *by Paul Kimmerly*

First, an experienced SEPG member writes about the SEPG's role as a communicator and a salesperson.

I was once asked to draw a picture depicting my job. While I am no artist, I tried to represent a chameleon wearing a used-car salesman's plaid jacket and a priest's collar. The SEPG's job may be easy to define in academic terms, but it becomes difficult when observing the practical day-to-day efforts of the group.

I chose a chameleon because the SEPG's job is always changing. The SEPG does not work for a single project or division within the organization; it must work with all of them. To successfully accomplish its goals, the SEPG must know the entire organization and how it works. This goes beyond the formal and overt organizational structure. We must understand the underlying culture and unwritten rules. This helps the SEPG choose which colors to present when working with different areas of the TSO.

To serve as a communication link and coordinator for the TSO's process improvement efforts, the SEPG must be able to find the right ways to approach each layer of the organization. Those layers separate line divisions from support divisions, and managers from practitioners. The SEPG deals with and across all divisions. Just as the chameleon must adapt its colors to its environment, the SEPG must adjust its approach to the organization.

The salesman's plaid jacket helps with the sales aspect of the SEPG's job. As stated above, we need to be persuasive. Our organization has a long history of successful software development. It is sometimes hard for the managers and practitioners to understand why change is necessary or even desired.

After analyzing our audience and knowing which color jacket to wear, the SEPG must work with "clients" to identify their areas of need and find a solution that will fit. One division may need a family cruiser process that is steady and dependable over the long haul. Another may need a sportier model to respond to rapidly changing road conditions. The SEPG must find the model that makes sense to its client. Then, it closes the deal and offers extended service to support the sale.

The insincerity implied by the used-car salesman's jacket is balanced by the priest's collar. Since the SEPG deals with all levels of the organization, establishing an atmosphere of trust is extremely important. The SEPG must listen to managers' confessions and comfort the practitioners' fears. In its communication role, the SEPG brings the concerns and accomplishments of practitioners up to managers, then takes explanations and guidance from the managers back down. To successfully accomplish this, both groups must trust the SEPG to deliver its message honestly and without attribution. By fostering an air of open communication, the SEPG can coordinate improvement efforts throughout the organization.

I have always felt the best description for an SEPG's efforts can be found in the title of a Jimmy Buffett song, *Quietly Making Noise.* The SEPG moves quietly through the organization changing approaches as needed to sell, educate, assist, listen, support, coordinate, and facilitate. By building a series of little connected sounds, the SEPG can build process improvement successes into bigger organizational fanfare.

Director, Vice Deputy Director *by Major Joel Ogren*

The SEPG's activities must align themselves with the organization's goals. To do this, the SEPG must work with senior management. In this section, the TSO's director, vice deputy director talks about his view of the SEPG's efforts.

It is not possible to know everything that you do not know about software process improvement—that is where the SEPG comes into play within our organization. Within the structural concept of our organization, the SEPG works for me; but in reality, it is a support group for the whole organization. At the most abstract level, I hold these people responsible for the organization's software process improvement activities. Once you get beyond the abstract, you realize the level of effort, personal commitment, and professionalism associated with this august group of individuals. The SEPG is a driving force for facilitating the definition, maintenance, and improvement of the software processes used within our organization.

Within the confines of my organizational role, I have watched the SEPG implement and facilitate organizational goals discussed with and defined by senior management. I have also witnessed them gently nudging senior management back inside the box of a defined software process.

The SEPG, by its very nature, is aware of every facet of our organizational process. This is a necessary component of its job. Without this, an SEPG is not empowered to facilitate the necessary changes within its organization. This is similar to the Japanese philosophy of *Kaizen,* which involves your whole organization on a daily basis in its quest to find continuous, measurable improvement in all (business) areas. The SEPG attends many of the management meetings, both at the senior management and project management level. In this role, they clarify the organization's mission and values within the organizational roadmap for process improvement. The SEPG is a tool to help guide the organization along the defined path of the Capability Maturity Model® (CMM).

Key to our view of the Kaizen philosophy is the way we choose our SEPG members, and the way that each member influences the organization. This process starts by selecting credible members of the organization and providing increased education in CMM-related topics. They are more apt to be in tune with the changing current at the grass-roots level of the organization, and can therefore influence the entire organization on a day-to-day basis. If they find that they truly believe in CMM, and what it can do for our organization, they become its staunchest supporters and the greatest influence on each and every member of the organization. Once this occurs, the SEPG naturally takes on a leadership role within the organization, in many cases mentoring their knowledge of CMM processes.

The SEPG's role in project implementation walks hand in hand with the key process areas (KPAs) associated with a CMM Level 3 organization. It compares and contrasts our current practices against the CMM's goals and key practices. While there is no magic formula for this, the SEPG helps the organization manage our processes through measurement. Establishing an implementation plan is done in an ascending hierarchy. This hierarchy begins with a vulnerability assessment, followed by resource analysis, analysis of priorities, education and awareness, a basic measurement of expected outcomes, inserting our current organizational software development plan framework, and finally the senior management decision support for program selection.

The SEPG plays a key part in establishing an enabling structure for process improvement. This is realized through the SEPG's enhanced support functions, which are readily available for the organization's stakeholders. These functions allow the organization to adequately capitalize on the available structure, allowing the organization to manifest through a process of change. The SEPG manages this process of change through metrics. These metrics continue to spell out the strengths and weaknesses of our structure, enabling us to benefit from an informed, proactively managed process improvement methodology.

The SEPG has built a partnership environment within our organization. By practicing strong communication techniques, the SEPG minimizes conflict and empowers itself at the same time. The SEPG has created a trusted environment that is necessary for widespread information sharing. This also ensures that sanctuary and anonymity are readily available to the stakeholders of the organization, when necessary. These managed efforts are key to the SEPG's success and the organization's process improvement efforts.

A change effort needs a guiding force to begin the change process and enhance the organization's capability to accept the change. Political sponsorship helps improve the acceptance of this change. The SEPG helps facilitate this change through a causal relationship it has developed between the CMM's KPAs and the software developers themselves. It effectively provides assistance in restructuring our processes and policies. It is an effective role model, reinforcing why new behavior is needed. It communicates the vision in a manner that is attainable and provides sustainable successes, both individually and organizationally. The metrics it maintains provide the results that reinforce these new behaviors and values. These new behaviors and values become our baseline for our organization's core competencies. With a strong SEPG in place within our organization, we have realized many tangible as well as intangible benefits.

Project Officer *by Carol Mullins*

After establishing a role in the organization, the SEPG must work with the projects to assist in implementing KPAs within the CMM. This requires support and buy-in of the lead project managers. In this section, a project officer discusses the SEPG's support of her project.

The SEPG in the TSO is more than a group of individuals devoted to working on software process improvement (SPI) activities. This group provides "**s**upport, **e**xpertise, **p**ersuasion, and **g**uidance" as the Marine Corps Total Force System (MCTFS) software developers and managers go about their day-to-day jobs. MCTFS is the largest automated information system maintained and modified by the Kansas City TSO. MCTFS achieved CMM Level 2 in January 1997. Since then, the staff has completed four software releases and a large Year 2000 conversion effort. In December 2000, MCTFS and four other systems within the TSO conducted a CMM-Based Appraisal for Internal Process Improvement; their process capability was rated at CMM Level 3. These efforts would not have been successful without the SEPG's support.

This group collects all the metrics from a variety of sources that are used during the Post Implementation Review (PIR) held at the conclusion of each release. The SEPG also facilitates three separate PIR sessions with practitioners, section managers, and branch managers. It presents the final results and recommendations to me as the project officer. The findings from these sessions combined with the metrics reports help me identify areas in need of improvement. The SEPG's support and impartiality combined with its knowledge of MCTFS has made the PIR a very successful event.

The SEPG provides a level of expertise about software process improvement that cannot be found elsewhere in the TSO. As MCTFS moved along the path to a Level 3 assess-

® The Capability Maturity Model and CMM are registered trademarks of the Software Engineering Institute and Carnegie Mellon University.

ment, the SEPG was instrumental in identifying the actions that needed to be taken, the processes that required documentation, and the areas that must be addressed to successfully move forward in our process improvement efforts. The SEPG worked closely with MCTFS management staff reviewing work products to ensure that they were accurate and that they addressed the applicable KPAs. The SEPG relied on SEPG members in other parts of the DFAS organization for alternative ways to approach issues raised by the MCTFS development staff.

The SEPG's metrics expertise, and the support it has provided to MCTFS developers were instrumental in formalizing the methods used to estimate the size and effort of each change to MCTFS. The SEPG started with the estimating spreadsheets built by MCTFS programming staff and modified them to include estimating and tracking features for both size and effort. From these spreadsheets, the SEPG worked with MCTFS managers to develop management summary reports. It has continued to make minor modifications to these spreadsheets, which are now used without exception in all of the development areas within MCTFS.

SEPG members are the first to tell anyone that they are strictly advisors—they help process improvements happen. While that is generally true, there are occasions where their subtle but persistent persuasion leads to some significant changes. The SEPG had been suggesting that the use of burn rate metrics would assist in identifying possible problem projects before they got out of hand. At first, branch and section managers were lukewarm to the idea. Undaunted, the SEPG developed burn rate reports and updated them regularly. They continued to do this and continued to remind the staff of their availability, and even noted items of interest. Slowly, these reports have become a part of the standard tool-set used by MCTFS managers.

As the manager in charge of MCTFS, I must continually strive to balance the software development activities of the MCTFS staff with the need to continually access how we do business, and how we can develop software more effectively and efficiently. The SEPG helps me achieve this balance. Having been software developers before joining the SEPG, each SEPG member provides me with a unique perspective of the impact that new ideas, procedures, or initiatives will have on the staff. This "having-been-there" perspective provides valuable input and often times guides me in making decisions. The SEPG frequently offers me a variety of alternatives as I contemplate new things to try. Again, their guidance has proven to be invaluable.

New SEPG Member *by Shannon Morten*

As part of their efforts, the SEPG must periodically rotate members. This allows the experienced members to return to projects bringing their new process knowledge with them. It also allows the SEPG to keep a project-level focus by bringing in new people from the development staff. In this section, two new SEPG members talk about the transition from being a member of the development staff to being the new person on the SEPG.

"So, what does the SEPG do again?" That was the question I asked after one day in their ranks. I came to the SEPG after serving as a test analyst for MCTFS. During that time, I spent my days searching for needles in haystacks. My first day in the SEPG was a major eye-opener. Suddenly, I found myself no longer searching for needles, but looking at every haystack in the field!

Based on my experience from having served on the SEPG for several months, I have learned the answer to my initial question. We support the organization by providing assistance in SPI activities, meeting facilitation, and in identifying and addressing its needs. That sounds so general and expansive, but that is what we do. The number of little details that goes into our job every day constantly surprises me. This job did not turn out to be what I expected, but it has proven to be everything I wanted, and everything I did not know I wanted.

I find that my knowledge and experience as a tester along with college courses I have taken provided me with a strong base of knowledge about software development. I can apply this knowledge when we work with each system within the TSO. The attention to detail that I brought with me from test has proven to be a strong asset. I tend to approach my duties from a *techie* point of view instead of a management view. It takes more effort for me to think like management and look at the big picture. I tend to focus on all of the individual pixels on the screen. However, since the SEPG works with all levels of the organization, I have found that my skills complement those held by other members of the group.

After joining SEPG, I discovered that I was involved with SPI activities daily as a practitioner and did not know it. As a tester, whenever a CMM-related acronym was mentioned my flags immediately went up, and I dismissed it as a waste of my time. I had a product to get out after all. SPI should be management's concern, not mine. My first week with the SEPG was a massive eye-opener. I started wondering how I could have been so blind for so long. By stepping back, I can see how process improvement efforts related to what I was doing. Now, I find myself trying to explain this revelation to my fellow practitioners. I often wind up fighting the same battle SEPG fought with me. Some people listen, and some do not. I think that my background gives credence to what I am saying now, and that I can put SPI terms into language that practitioners can understand.

As I learn more about CMM and how to apply it, I am realizing that I am also applying the principles to myself. As a practitioner, I maintained a Level 1 attitude. Funny how it happens this way, but now as an SEPG member, I find my own behavior maturing.

New SEPG Member *by Pamela Yancey*

During my career, I have always sought opportunities to grow professionally, contribute, and make a difference. This led me to become an advocate for Total Quality Management (TQM) in the early 1990's. I knew the concepts were viable, and it seemed like a practical solution to address inefficiencies in an organization, as well as a means to foster teamwork.

When I joined SEPG, I understood CMM principles. I saw how it related to TQM, and I supported the concept wholeheartedly. I was prepared to take an active role as an SEPG member; however, I was unprepared for the emphasis and level of dedication within TSO to improve their software development processes. It was obvious to me from the beginning that

TSO equated implementation of the software-CMM to the success of their business objectives. Time and again, I saw how the TSO was willing to invest the time, effort, and money required to implement a flourishing SPI program.

My first month or so on the SEPG was overwhelming to say the least. The TSO is a collection of widely diverse projects, each with different needs. Quite frankly, after meeting with most of the project managers, I had difficulty seeing a standardized approach to process improvement within such a diverse organization. Besides, I wondered how I would ever become proficient when I felt that I had so much to learn. Though I felt disheartened at times, I continued to remind myself of the apparent success achieved within our organization. I knew that there had to be a connection between the SEPG's efforts and that level of success.

I took more than a few months to fully appreciate my role on the SEPG, and the group's role in the success of our organization's SPI efforts. I discovered that my success on the SEPG depended upon my ability to understand and embrace the diversity of our organization. I also recognized that one of the SEPG's greatest strengths lies in its ability to listen to the needs and concerns of our projects. In turn, the SEPG works with the projects to develop solutions. While these solutions address the projects' needs, they also bring about subtle organizational changes that enable future process improvement efforts. We celebrate these small changes as successes, recognizing that change cannot be approached abruptly or aggressively.

I saw that my role, and that of the SEPG, involves constantly working behind the scenes to promote heightened awareness of SPI and CMM-based process improvement initiatives. The SEPG allows the projects to resolve their unique issues while weaving a consistent thread throughout the organization. To be effective, my contributions are not only based on my expertise, but my ability to support the project managers and their staff. I had to maintain a delicate balance between my ability to provide guidance, direction, and training with my ability to foster support, encouragement, and open communication in all daily activities.

More than a year has passed since I joined the SEPG. I now feel at ease with my role, and feel like I am part of a cohesive team. We share a common commitment and belief in SPI. As a SEPG member, I have the unique opportunity to make a viable contribution to our organization's success in implementing or encouraging continued CMM-based process improvement initiatives. Additionally, I have the opportunity to grow and make a difference in our organization.

Experienced SEPG Member *by Capt. Wendell Bazemore*
Experienced SEPG members have seen the good and the bad. They have brought their project-based knowledge to SEPG and supported the organization and the projects. In this section, an experienced SEPG member talks about what it takes to succeed in supporting process improvement efforts.

Of the many things I've learned during the past two years as an SEPG member, three are most important: aligning SPI initiatives with business goals, the value of metrics for influencing culture change, and the importance of taking a project approach to process improvement initiatives.

The challenge for SEPG is aligning process improvement efforts with the day-to-day and strategic business objectives. Everyone at TSO can relate to the activities involved in developing a quality software product. It is the main thing. Concepts like SPI, CMM, or a certain maturity level may not be as relevant as software baselines, estimates, quality reviews, or defects to the software developer. This is not to downplay the importance of the CMM as a framework for process improvement; however, we (SEPG) have been most successful when we can show the business impact of process improvement initiatives. After all, the main thing is keeping the main thing the main thing.

In terms of influencing behavior change and reinforcing process improvement concepts, we have gotten the most mileage out of our metrics program. There is a perception among some in the SPI community that "you must wait until the organization is well on its way to Level 3 before implementing a metrics program." Indeed, CMM starts talking about the software process database in the Organizational Process Definition KPA. However, based on what I have observed in this organization, I would encourage SEPGs in any organization to begin collecting measures as soon as possible. There is a minimum set of data—effort, schedule, and defect numbers—that will prove useful to any organization. Our metrics program has grown and continues to grow as managers gain confidence in the existing measures and consider other measures that will provide insight into key project issues and concerns.

To enhance the success of the process improvement effort we must treat it like a project. Our organization is like most in that we think in terms of projects. Like company commanders in a rifle battalion, project officers are trusted with carrying out those most important functions of the organization. The TSO's structure supports the project officer's ability to ensure the production of quality software, on time and within budget. That structure includes a strategic plan, a chain of command, milestone reviews, and meetings with senior management. Resources are constantly being evaluated and frequently tradeoffs are made. Recently the TSO capitalized on the existing structure to implement key process initiatives, culminating in a successful assessment for CMM Level 3.

Summary

As you can see, SEPG represents different things to different people. However, there are common themes that run throughout. Communication, coordination, and support appear to be the main SEPG duties. All of the parties above talk about the SEPG's ability to relate to the different levels of the organization in language they understand. This helps the SEPG establish and foster process improvements throughout the organization. In order to do that, the SEPG must coordinate the activities of different areas to ensure the organization is moving in the right direction. This coordination comes from the different forms of support the group provides. The SEPG stands ready with services ranging from metrics analysis and reporting, to meeting facilitation, to counseling. All levels of the organization can benefit from the SEPG's efforts without being able to specifically say what they do. "It depends," is a favorite answer given by the SEPG when a question is asked. It also applies when trying to describe what they do.u

About the Authors

Paul Kimmerly has 13 years of experience in software development for the Defense Finance and Accounting Service (DFAS) Information and Technology Directorate in Kansas City, Mo. Since 1993 he has been a member of the Software Engineering Process Group, and its chair for the past five years. He also chaired a group that addressed DFAS-wide process improvement issues. He is a member of and presenter to the Software Process Improvement Network, and a past CrossTalk contributor. He presented at the 1997 SEI Symposium and participated on a panel discussion there in 2000.

Paul J. Kimmerly
DFAS-TKZ/KC
1500 E. 95th Street
Kansas City, Mo. 64197
Voice: 816-926-5364 DSN 465-5364
Fax: 816-926-6969 DSN 465-6969
E-mail: paul.j.kimmerly@.dfas.mil

Maj. Joel Ogren joined the Defense Finance and Accounting Service in Kansas City, Mo., in June 1999 and is currently director, vice deputy director for the Information and Technology Directorate (TSO). He also serves as a software process improvement champion for the TSO, and is a reviewer for the Institute of Electrical and Electronics Engineers on topics relating to software processes, information assurance, and communications engineering. He is a member of and presenter to the Kansas City SPIN, and is a graduate of the Naval Postgraduate School in Monterey, Calif.

Maj. Joel Ogren
DFAS-TK/KC
1500 E. 95th Street
Kansas City, Mo. 64197
Voice: 816-926-7374 DSN 465-7374
Fax: 816-926-6969 DSN 465-6969
E-mail: joel.ogren@dfas.mil

Carol Mullins has been director of the Marine Corps Total Force System (MCTFS) division since August 1996. She is responsible for the design, programming, and testing of all changes to MCTFS, as well as all maintenance functions that support MCTFS. Previously Mullins worked for more than 13 years at the Navy Personnel Research and Development Center, where her final position was supervisory operations research analyst.

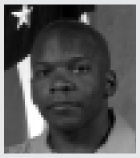
Carol Mullins
DFAS-TKT/KC
1500 E. 95th Street
Kansas City, Mo. 64197
Voice: 816-926-5360 DSN: 465-5360
Fax: 816-926-6969 DSN 465-6969
E-mail: carol.m.mullins@dfas.mil

Shannon Morten joined the Software Engineering Process Group (SEPG) at the Defense Finance and Accounting Service in Kansas City, Mo., in December 1999. Previously she worked three years in software development in the Test and Evaluation Division within the Information and Technology Directorate to the SEPG. She is a member of the Kansas City SPIN and presented to them in April 2000.

Shannon Morten
DFAS – TKZ/KC
1500 E. 95th Street
Kansas City, Mo. 64197
Voice: 816-926-5364 DSN 465-5364
Fax: 816-926-6969 DSN 465-6969
E-mail: shannon.morten@dfas.mil

Pamela Yancey joined the Software Engineering Process Group (SEPG) at the Defense Finance and Accounting Service in Kansas City, Mo., in December 1999. Yancey has held a variety of positions with DFAS and brings experience from both the functional and software development communities to the SEPG. She was active in the total quality management efforts within DFAS during the 1990s. She is a member of the Kansas City SPIN and presented to them in April 2000.

Pamela Yancey
DFAS-TKZ/KC
1500 E. 95th Street
Kansas City, Mo. 64197
Voice: 816-926-5364 DSN 465-5364
Fax: 816-926-6969 DSN 465-6969
E-mail: pamela.a.yancey@dfas.mil



Capt. Wendell Bazemore joined the Software Engineering Process Group (SEPG) at the Defense Finance and Accounting Service (DFAS), Kansas City, Mo., in October 1998 after completing his studies at the Naval Postgraduate School in Monterey, Calif. His master's thesis focused on the process improvement efforts at DFAS-KC. He has been involved in local and corporate levels for DFAS, is a member of the Kansas City SPIN, and has made several presentations to that group. He recently transitioned out of SEPG to become the lead of DFAS-KC Software Quality Assurance efforts.

Capt. Wendell Bazemore
DFAS-TKZ/KC
1500 E. 95th Street
Kansas City, Mo. 64197
Voice: 816-926-5364 DSN 465-5364
Fax: 816-926-6969 DSN 465-6969
E-mail: wendell.bazemore@dfas.mil

Shortly after the turn of the 20th Century, the Wrights flew the first plane.

Now it's your turn.

Design and Fly a 21st Century (Paper) Plane ... Details in the April issue.

Raytheon Stands Firm on Benefits of Process Improvement

CrossTalk Managing Editor Pam Bowers interviewed Robert L Keys, senior principal software engineer, and Ginger Tonkin-Sugimoto, principal software engineer, Software Engineering Process Group, about benefits they have seen and experienced during Raytheon Missile Systems' advance from Capability Maturity Model® Level 2 to Level 4. This interview is part two of an investigation into the relationship between acquisition reform and process improvement that was initiated in November 2000 CrossTalk in the AMRAAM and JASSM programs article. The final part in this series will appear in a later issue.

`CrossTalk:` What was Raytheon's motivation to initiate a software process improvement program?

Tonkin-Sugimoto: Our motivation for software process improvement is to produce higher quality products at lower cost and within schedule. Achieving these goals will ensure we remain the contractor of choice. The Software Engineering Institute's (SEI) Capability Maturity Model® (CMM) provides a worldwide-accepted model for achieving these goals. A secondary benefit of the assignment of a CMM rating is that it represents a significant competitive advantage. It's an observable rating, far more concrete than simply saying, 'We have great processes. You just have to believe us.'

Keys: Since we began progressing from CMM Level 2 in 1994 to Level 4 in 1998, we have concrete evidence of an increase in financial performance and of a 144 percent productivity improvement. So we believe that progressing to the next level of CMM maturity will continue to reap benefits for us.

`CrossTalk:` What were the hardest internal obstacles you overcame in moving from CMM Level 3 to Level 5? What was necessary to overcome them?

Tonkin-Sugimoto: The hardest internal obstacles in the move from Level 3 to Level 5 have been misconceptions and a lack of understanding. We have heard "We have Level 3, why go farther?", "Is Level 5 worth it? Is it really value-added?" [and] "We didn't sign up to do that!"
The major factors in overcoming these attitudes are:

- Senior management buy-in.
- SEPG support for process improvement across the organization.

But this is not all that process improvement requires. Many activities support process improvement. Training that facilitates the implementation of new processes and metrics is necessary. These new processes and metrics must be used consistently across the organization. We require monthly program review packages with both reports and metrics. Our software quality engineers monitor both product and process. Metrics are monitored across the organization and senior management measures projects via these metrics. With this, process improvement takes on a new importance. To further support process improvement, our SEPG mentors projects in using their metrics to define the root causes of problems. We get project personnel involved in developing and analyzing their process improvements.

`CrossTalk:` What are the most difficult obstacles you overcame when dealing with process improvement resistance from your clients? How do you overcome these struggles?

Tonkin-Sugimoto: Few of the customers we work with are obstacles to process improvement. We have occasionally heard that they don't care about process improvement ("It's a contractor problem.") or they don't see the value in process improvement ("We aren't paying you to do that. Process costs too much.") or it doesn't apply to their projects ("We're special."). Primarily, our customers are concerned with cost and schedule. When our customers understand that process improvement has a positive effect on productivity and quality, they no longer resist. Here is how these potential obstacles are overcome:

- Providing real data that demonstrates a positive return on investment resulting from process improvement.
- Training to show our customers that process improvement improves productivity and product quality.
- Offering assistance to projects (developing metrics and analysis as needed).
- Senior management support of process improvement across projects.

`CrossTalk:` Can you provide an example of how a higher rating has helped you to deliver a better product, or other benefits?

Tonkin-Sugimoto: Our Advanced Medium Range Air-to-Air Missile (AMRAAM) projects are operating to the Level 5 practices and procedures of the Raytheon Missile Systems Software Directive System (RMS SDS). Early in the AMRAAM Foreign Military Sales Tape 8 project, both budget and schedule were deemed at risk. With our continuous improvement culture, project management developed the Design Impact Assessment Process to reuse previous work while training less experienced engineers. Project metrics showed that this process allowed them to quickly create a basis for the detailed design, to build confidence that schedule and budget constraints could be met, and to enable less experienced engineers to be effective within the time constraints. This process is currently being piloted on another project.

Keys: The corporation developed a set of standard operating instructions (SOIs). We used these as a starting point then instantiated them into the RMS software development process. In some cases we used the SOIs as they were. In other cases, they required some modification. For example, we augmented them for our metrics program with very specific templates for each metric so that we have a common look and feel to all of our program review packages.

`CrossTalk:` Do higher CMM level teams consistently perform better? How do you measure this?

Tonkin-Sugimoto: We have found that higher CMM level teams consistently perform better. Our organizational productivity has shown improvement in our journey from Level 2 to Level 4 . Our organizational Cost Performance Index (CPI) and Schedule Performance Index (SPI) have shown improvement between 1998 and the present. Organizational defect containment has shown improvement since 1995. These metrics measure both product and process performance.

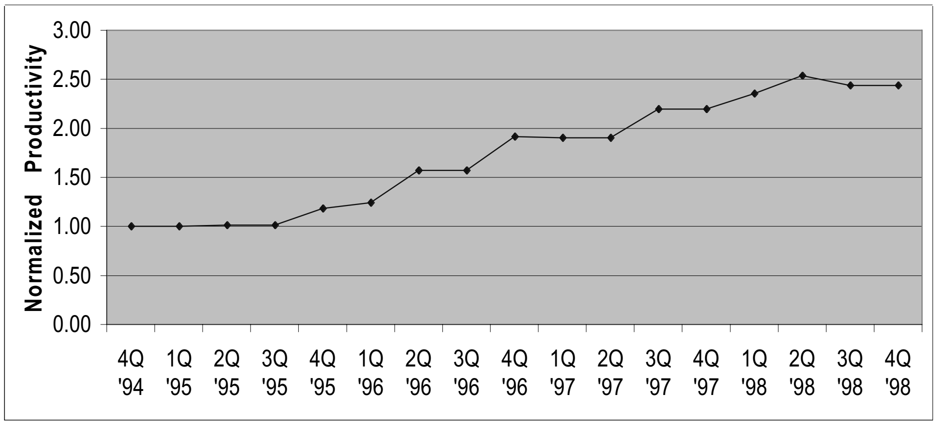
Keys: The measurements we generate, which include productivity, show improvement as an organization. While our CPI and SPI numbers as an organization have been getting better, they also are less volatile than they once were. There are contributing factors such as defect containment that have a significant impact on productivity and financial performance. We're detecting our defects during peer reviews within the current phase (in phase) as opposed to finding them in later software development phases where they are much more expensive to find and fix. We've been showing more than a 10 percent improvement per year.

`CrossTalk:` Can you provide hard data on defect containment or productivity improvement between Levels 2 and 5?

Tonkin-Sugimoto: The data in Figures 1 and 2 show that our organization's productivity has improved 144 percent from 1995 when we were a Level 2 to 1998 when we were a Level 4. During the period that productivity was increasing by 144 percent, we expended 6 percent of the annual budget on process improvement. This yields a 6-to-1 return on investment. Our overall defect containment during the same period increased from 32 percent in phase to 72 percent in phase.

`CrossTalk:` Is productivity translated into faster coding?

Keys: Yes, we are producing more lines of code per staff-hour. Our 144 percent productivity improvement in the rate of creating lines of code is because we're doing it right the first time. There are lots of parameters to increased productivity. The learning curve decreases dramatically as engineers move from project to project and the same for managers. All these things add up to improve productivity, which we measure starting with the design activity and ending prior to formal qualification testing. Requirements analysis is excluded from our productivity number.



Date	4Q '94	1Q '95	2Q '95	3Q '95	4Q '95	1Q '96	2Q '96	3Q '96	4Q '96	1Q '97	2Q '97	3Q '97	4Q '97	1Q '98	2Q '98	3Q '98	4Q '98
Productivity	1.00	1.00	1.01	1.01	1.18	1.24	1.57	1.57	1.91	1.90	1.90	2.20	2.20	2.35	2.54	2.44	2.44

Figure 1. *Organizational Productivity*

`CrossTalk:` How do you measure project success? Is it different for teams at different CMM levels?

Tonkin-Sugimoto: We measure both product and process status monthly via 14 metrics and 14 reports on each software project. These include the Enterprise Metrics: Software Size, CPI, SPI, Equivalent Productivity, and Fault Density that are collected throughout Raytheon. Our CMM Level 5 frontrunner projects monitor and report all of these metrics. Lower CMM level projects are not able to report all of these metrics or reports and must tailor some metrics. This inhibits their ability to manage their projects but it provides improvement opportunities that are addressed in their software process improvement plans (SPIPs).

Process measurement begins with the definition of project goals. Thresholds are established and actual project performance is monitored via the metrics. This provides fact-based management with early insight into unexpected results and allows for early root-cause analysis and mitigation planning.

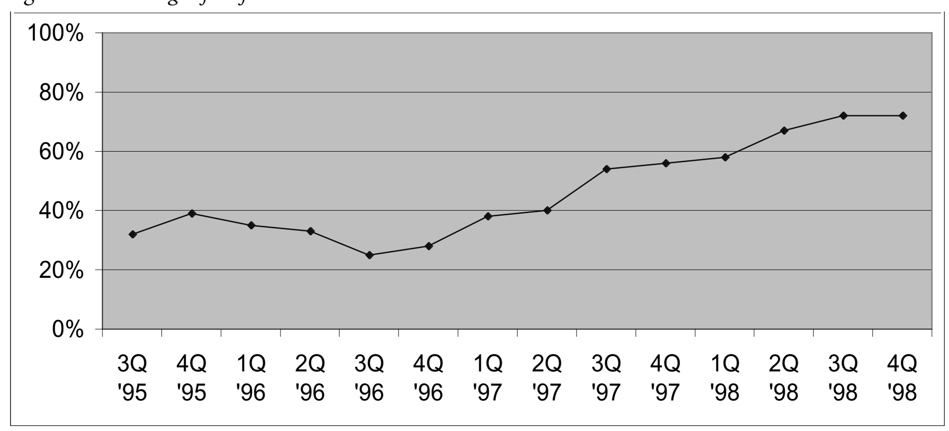
Keys: Each program fills out a blank monthly metrics template. The templates are all Excel, Word, and PowerPoint files that automatically link all the underlying data into the presentation package. Less mature projects tailor the templates while a Level 5 project will adopt the full process. Less mature projects might tailor out the cost of quality metric because it is very advanced and very mature. All reviewers—the software lead, the department managers, the SEPG and the Team of Five (TOF)—are looking for negative trends or unexpected results, which exceed control limits in either a positive or negative direction. We like to investigate either situation.

`CrossTalk:`How did the merger between Raytheon and Hughes impact your software process improvement effort?

Tonkin-Sugimoto: The Tucson facility has undergone three recent transitions:

1. The merger of General Dynamics and Hughes Missile Divisions.

Figure 2. *Percentage of Defects Found in Phase*



Date	3Q '95	4Q '95	1Q '96	2Q '96	3Q '96	4Q '96	1Q '97	2Q '97	3Q '97	4Q '97	1Q '98	2Q '98	3Q '98	4Q '98
Defects in Phase	32%	39%	35%	33%	25%	28%	38%	40%	54%	56%	58%	67%	72%	72%

2. The transition of the Hughes Canoga Park, Calif. work to Tucson.
3. The merger of Raytheon, Texas Instruments, and Hughes Missile Divisions.

The Tucson software process effort was greatly impacted when projects from Massachusetts (legacy Raytheon) and Texas (legacy Texas Instruments) were moved to the Tucson facility in a consolidation effort. It was at this point that the corporation developed the Raytheon SOIs from the three sets of best practices.

"We have found that higher CMM level teams consistently perform better."
–Ginger Tonkin-Sugimoto

The SOIs were instantiated for use at RMS in the form of the RMS SDS, which all frontrunner projects have adapted. Others are in transition as documented in their project SPIPs. This has been a major effort and took up much of our software process effort. We had originally planned to prepare for a Level 5 CMM-Based Appraisal Internal Process Improvement in 2000 but that has moved to 2001 (see Figure 3). We completed a Process Baseline Assessment in 2000 to measure our preparation for the 2001 assessment. It showed that we have maintained our quantitative process management/software quality management capabilities in spite of the merger of the three organizations.

It was a difficult process because it was such a massive undertaking. A large team reviewed every procedure or SOI that we developed with various program representatives. They each had the opportunity to provide input. If you don't have all the various parts working together to develop your best practices, someone is not going to go with you.

Keys: Communication was our greatest problem during the transitions. I would be talking to someone from Texas about what we called "earned value" in Tucson, which is a measure of cost and schedule performance; but it would mean something completely different to him. It took about 12 grueling months to complete the procedures to define the process, but it was key. If you talk to any of our folks now, you can't tell which legacy organization they came from. We're all talking the same language now.

`CrossTalk:` Does a higher level of maturity help with modifications? Is current code at Level 5 easier to modify in the future?

Keys: Yes, the code we're developing now will be much more maintainable then perhaps some of the code from a long time ago. That is because the CMM emphasizes the peer review process. We also emphasize having clear, well-defined work instructions and standards, which may not have always been the case in the past. The process causes us to more carefully evaluate our approach to doing the job, documenting it more clearly, and establishing clearer exit criteria.

`CrossTalk:` When you come up against a brick wall in funding process improvement, how do you overcome that?

Tonkin-Sugimoto: A brick wall in funding process improvement is brought about by a lack of understanding. Process improvement is not expendable. It improves productivity and quality of products. If funding is an issue, the first step is to demonstrate its importance to the organization's future. From there we get buy-in from either senior management or from program management. They are the source of the funding. How well you demonstrate process improvement can determine future funding. We use real data and actual success stories from projects to demonstrate the value of process improvement.

`CrossTalk:` Do CMM rankings provide a level playing field to compare contractors?

Tonkin-Sugimoto: CMM rankings only provide a level playing field when they are performed to the same standard and are performed regularly. The Raytheon Software Assessment Team has held to strict standards when assessing the various Raytheon facilities. We are aware that some outside assessors do not necessarily hold the same standards. When you look at an organization's assessments you need to know who did the assessment.

When comparing contractors, confidence is greatest when the project is part of the assessment. Confidence is reduced if the project is not part of the assessment.

Keys: Another important factor on this level playing field is to take into account the date of the assessment because there is no expiration date on the SEI ratings. So an assessment performed five years ago, especially considering industry mergers and transitions, could be invalid even within two or three years.

`CrossTalk:` What effect does a performance level rating have on past performance evaluations?

Tonkin-Sugimoto: It depends upon the organization that you are evaluating. Past performance evaluations for lower process level companies do not readily forecast how that company will perform on a new contract. When a company has been rated at least CMM Level 3, its processes are institutionalized. The tools, procedures, and training are in place to enable personnel to perform at least as well as they did in the past.

`CrossTalk:` What steps do you take to ensure you maintain your capability level?

Tonkin-Sugimoto: The RMS SDS is CMM Level 5 compliant, and we assure our capability by having all projects tailor their processes from the RMS SDS.

We maintain our capability through monthly TOF meetings that include the project software lead, software configuration management representative, software quality engineer, product line SEPG representative, and department manager. The TOF evaluates software project metrics, accomplishments, risks, and problems, and can initiate mitigation efforts. Our capability is assured at three different levels:

1. Organizationally, a software process engineer is assigned for each product line and process training is required for all engineers. At the project level, SQE audits of project software processes are performed throughout the life of each software project. At the directorate level, engineering process reviews are held each month to evaluate and report process status across all disciplines.
2. Organizationally, we develop a SPIP every year that details our improvement goals for the year. We measure our efforts against this plan monthly.
3. SEI suggests that CMM assessment ratings be reviewed every 18 to 30 months. As part of Raytheon, we are assessed on this schedule. The Tucson

facility was assessed in 1996 as a CMM Level 3 and in 1998 as a CMM Level 4.

4. Recently, we underwent a Process Baseline Assessment to determine our weaknesses in reaching CMM Level 5. Our next CMM assessment is in 2001.

CrossTalk: What do you see in the future for process improvement? What would you like to see? Do you plan a transition to CMMI? Why?

Tonkin-Sugimoto: Software process improvement at this facility is institutionalized. This was brought about by Raytheon's software process improvements specifically but also across the board by Raytheon Six Sigma. This program encourages every employee to improve their activities, to make them more efficient, faster, less costly, and improve quality. We're beginning to feel empowered to make improvements.

The Engineering Directorate for the enterprise is actively moving to CMMI by building on our software successes. Raytheon has had two pilot CMMI projects: one in Bedford, Mass., and one in St. Petersburg, Fla. Our CMMI activity will represent another competitive advantage for Raytheon in the same way that the software CMM maturity has. As all engineering disciplines reach a culture of continuous process improvement, we will be able to produce better and better products.

Keys: The value of software CMM process maturity is diluted by the fact that other disciplines don't have the same type of controlled processes and maturity that we do. This is particularly true in the area of systems engineering and requirements development. If we can bring those organizations up to our level of maturity, then we're going to get tremendous bang for the buck. It will increase the benefits of software maturity as well as help out other disciplines, program management, and all other areas.u

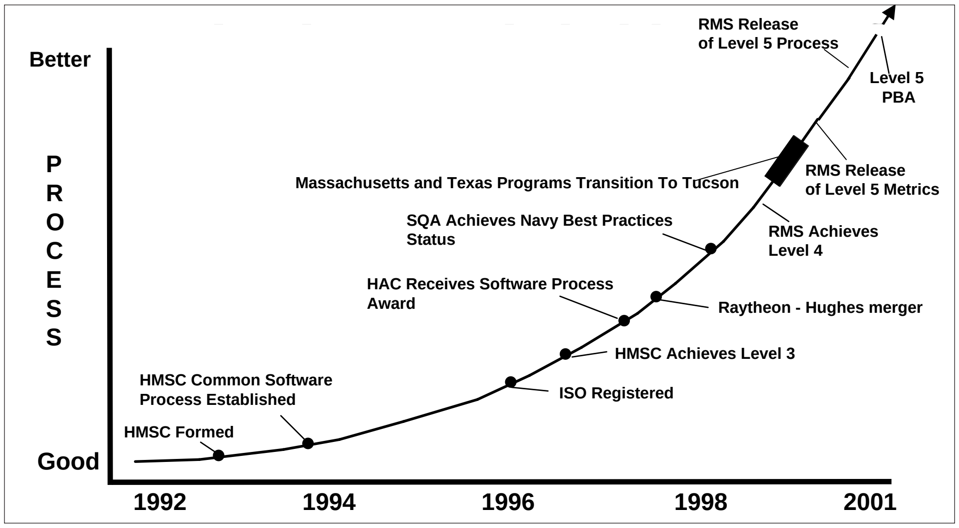


Figure 3. *Raytheon's Process Improvement Timeline*

Robert L. Keys is a senior principal software engineer at Raytheon Missile Systems (RMS) in Tucson, Ariz. He has 23 years of experience developing and managing real-time embedded software for various defense-oriented applications, including command and control systems and missile systems. For the past four years he has worked in the Software Engineering Process Group (SEPG) as RMS has progressed from Software Engineering Institute's CMM Level 3 to Level 4. He is currently the metrics-based management specialist within the organization and has been the leader of the software metrics program since 1998. He is also currently involved in developing the Level 5 processes and providing support for Level 5 project activities in preparation for certification in May 2001. Keys earned a bachelor's degree in computer science from San Diego State University.

Ginger Tonkin-Sugimoto is a principal software engineer at Raytheon Missile Systems in Tucson, Ariz. She has 28 years of experience developing and managing real-time embedded defense software. For the past four years she has worked in the Software Engineering Process Group (SEPG) and currently represents process for the air-to-air software product line. She is an active member of assessment teams working throughout Raytheon to mentor other organizations and to improve software development processes. She earned a bachelor' s degree in mathematics from the University of California, Santa Barbara.

Attention Air Force Software Program Managers!

Wrestling with legacy code? Slaved to proprietary hardware? Having difficulty hiring the talent you need?

The Air Force Computer Resources Support Improvement Program (CRSIP) is listening!

CRSIP invites your input to the Air Force's Software Technology Investment Strategy and your participation in a roundtable and panel discussion of software needs (technology, acquisition management, and process improvement) on Tuesday, May 1 from 12:00 to 1:00 p.m. during the 2001 Software Technology Conference in Salt Lake City, Utah. The complete conference schedule will be posted soon at www.stc-online.org

To gain wider input to Air Force requirements for software improvement projects, CRSIP is interviewing engineers and program managers of software intensive programs for inputs. The results of the needs analysis will be summarized at STC, and used to structure the Air Force's investment strategy for software improvement. If you would like your input to be a part of this effort, see http://crsip.hill.af.mil for a questionnaire and more information.

DoDTech Listserver Covers All IT Issues

George Hellstern
Office of the Secretary of Defense

During the year 2000 (Y2K) effort, the Office of the Secretary of Defense established a technology news service for the entire Department of Defense (DoD). The Y2K Technical Listserver was created to get important Y2K information to the program managers who needed it. Throughout 1999 the listserver distributed unclassified news highlights, technical developments, briefings, published policies, and upcoming Y2K events to more than 2,000 defense and government Y2K program managers. The listserver became a means for program offices to share newly discovered solutions across commands, services, and agencies.

The Y2K Technical Listserver provided information on the latest testing tools, the Microsoft Windows patch levels and their states of compliance to Y2K, as well as falsely reported problems such as the fabled Crouch-Echlen effect. Members of NATO, the Japanese Ministry of Defense, and the White House Y2K Directorate commented on its quick analysis of relevant facts, and its general usefulness. Users submitted articles as well, opening up an information flow across DoD organizational boundaries.

After the successful transition through Y2K, the listserver was renamed the DoDTech Listserver and continues its life. DoDTech maintains the intent and spirit of its predecessor, and has an expanded focus to include all information technology (IT) issues. DoDTech has addressed the topics of information security, migration to Windows 2000, mobile code, electronic commerce, product reliability, and Lucent's new Peta-bit all optical Internet-protocol based routers and switches.

To keep the audience informed of what's happening in the department, DoDTech includes highlights of hot IT initiatives such as the Global Information Grid, a set of programs to bring the DoD Network infrastructure as a whole into the 21st century. Other programs, policies- and IT-related events will all be listed and analyzed by the listserver. DoDTech continues to foster communication and information sharing across the government by featuring new policies signed at the executive level, and initiatives that are proving successful in other agencies.

DoDTech could not be successful without the continued interest of its audience, and the excellent coverage of IT news in the press, where it leverages the majority of its content. This electronic newsletter is currently published bimonthly, via email, to approximately 2,000 readers across the globe. DoD hopes to expand the listserver's reach by promoting the value of information sharing at DoD-sponsored conferences and events. To view an archive of DoDTech issues, visit www.c3i.osd.mil/org/cio/dodtech

To Join

Simply follow the steps outlined below to be added to the distribution:

1. Send message to: listserv@listservsas.c3i.osd.mil
2. Leave the subject line BLANK.
3. In the message body type: SUB DoDTech Your Name (for example: John Smith, not your e-mail address).

Once you have signed up you will start receiving biweekly news-mails compiled and published by the DoD DCIO Software and Advanced Solutions Directorate. If at any time you no longer wish to be a member of the list, follow these steps:

Send message to: listserv@listservsas.c3i.osd.mil
Leave the subject line BLANK (as always)
In the text section type: SIG DoDTech

We look forward providing you with useful and timely information, and to receiving your comments and suggestions. If you have any questions about the listserver please contact Kelli Gillis at 703-604-1484.

Contact Info: Voice: 703-602-2720 E-mail: George.Hellstern@osd.mil

Coming Events

March 5-8
Software Testing Analysis and Review
www.sqe.com/stareast

March 5-8
Mensch and Computer 2001
http://mc2001.informatik.uni-hamburg.de

March 12-15
SEPG 2001 FOCUSING ON THE DELTA
Software Engineering Process Group Conference
www.sei.cmu.edu/products/events/sepg

March 31-April 5
Conference on Human Factors in Computing Systems
www.acm.org/sigs/sigchi/chi2001

April 22-26
Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies
www.ieee-infocom.org/2001

April 29-May 4
STC
Software Technology Conference (STC 2001)
www.stc-online.org

May 1-3
2001 IEEE Radar Conference
www.atlaessgrss.org/radarcon2001

May 12-19
23rd International Conference on Software Engineering, and *International Workshop on Program Comprehension*
www.csr.uvic.ca/icse2001

June 11-13
E-Business Quality Applications Conference
http://qaiusa.com/conferences/june2001/index.html

June 18-22
38th DAC
ACM/IEEE Design Automation Conference
www.dac.com

June 25-27
2001 American Control Conference
www.ece.cmu.edu/~acc2001

July 1-5
INCOSE2001
11th Annual International Symposium of the International Council on Systems Engineering
http://incose.org/symp2001

July 7-13
2nd International Symposium on Signal Processing and Analysis
http://ispa.zesoi.fer.hr

July 29-August 2
TOOLS USA – Software Technologies for the Age of the Internet
www.tools-conferences.com/tools/usa/index.html

Join us in beautiful Salt Lake City, Utah, for the Thirteenth Annual

Software Technology Conference

STC 2001

2001 Software Odyssey: Controlling Cost, Schedule, and Quality

April 29 – May 4, 2001

The Software Technology Conference (STC) is co-sponsored by the Departments of the Army, Navy, and Air Force, the Defense Information Systems Agency, and Utah State University Extension. Over 3,000 participants are expected to attend more than 110 presentations (offered in 10 concurrent tracks) and the accompanying trade show. The trade show offers participants the opportunity to visit over 180 exhibiting organizations and learn about the latest products and services they offer.

Today's leading technology experts will look toward the future and guide participants in exploring opportunities for perfecting today's technology. Presentations are geared toward effectively managing the daily challenges that IT professionals face. This is one of the best opportunities for software professionals and managers in the DoD, contractors supporting the DoD mission, and individuals in industry and academia to learn of proven technologies, policies, and practices regarding issues common throughout the DoD. *It's not too late! Register today!*

Concurrent Speaker Luncheons

Monday, April 30 • 11:30 a.m. – 1:00 p.m.

Luncheon speakers will discuss their perspectives on the conference theme. Participants may register for the luncheon of their choice by marking the appropriate box on the registration form. Speaker choices include:

Luncheon A
Mitra Azizarad
General Manager,
Microsoft Federal Systems

Luncheon B
Dr. Barry Boehm
Director, USC Center
for Software Engineering

Luncheon C
RADM Patrick Moneymaker
USN (Ret.); Chief Operations Officer,
Ocean Systems Engineering Corp.

Opening General Session

Monday, April 30 • 1:30 p.m. – 3:00 p.m.

Dr. Vitalij Garber
Director of Interoperability,
Under Secretary of Defense
(Acquisition and Technology)

Plenary Speakers

Industry Speaker

Steven R. Perkins
Senior Vice President, General Manager of Oracle's Federal and Global Financial Services
Wednesday, May 2
8:00 a.m. – 8:45 a.m.

Academic Speaker

Dr. Eliyahu M. Goldratt
Educator,
Creator of the
Theory of Constraints
Thursday, May 3
8:00 a.m. – 9:40 a.m.

www.stc-online.org

STC 2001

General Information	**Technical Content Inquiries**
stcinfo@ext.usu.edu	stc@hill.af.mil
435-797-0423	801-777-7411
Trade Show Inquiries	**Media Relations**
stcexhibits@ext.usu.edu	stcmedia@ext.usu.edu
435-797-0047	435-797-1349

Watch for more information about STC 2001 in the April issue of CrossTalk.

Discounted Registration Fees

Completed registrations received by Monday, March 26 will be processed at the discounted registration fee of $560 military/government or $685 business/industry/other. Credit cards will be processed beginning Monday, April 2.

Registrations received after March 26 and those received without payment in full or a completed purchase order will be held for onsite registration at the regular fee of $620 military/government or $770 business/industry/other.

Registration instructions and forms are available at the STC Web site in PDF format. You may also register online using our secure server.

Updated Conference Information

Complete conference information is available on the STC Web site. Presentation summaries and speaker biographical information is posted there as it is received. Visit www.stc-online.org for up-to-date conference information as well as a dynamic conference schedule that allows you to "build" your conference week online.

Side Meetings

Side meetings are held in the evenings throughout the week and on Friday. There is a limited number of meeting rooms available. Requests to hold a meeting and reserve space must be made no later than Monday, March 26. Rooms are assigned on a space available basis. To schedule a room, contact STC management at 801-777-9828 or lynne.wade@hill.af.mil. All meetings scheduled prior to March 26 will be listed in the Conference Pocket Agenda provided at onsite registration. Reminder: Side meetings will not be scheduled to conflict with conference events.

The Best Measurement Tool Is Your Telephone

Don Scott Lucero
U.S. Army Software Metrics Office

Fred Hall
Independent Engineering Inc.

More and more people are becoming interested in software measurement for three reasons: Measurement and analysis are a Level 2 process area of the Capability Maturity Model Integration^{SM}, the International Standards Organization measurement standard ISO 15939 is being drafted, and Practical Software and System Measurement version 4.0 is being released. This paper provides some practical advice on implementing a software measurement program based on lessons learned at the U.S. Army Software Metrics Office during the last 10 years.

Everyone should know why the organization wants to measure. The most effective measurement programs are tailored to the important issues that must be managed in an organization. Although many software metrics programs are based on this issue-driven measurement approach, many are not effective because the selected issues are not commonly endorsed. One lesson learned by the Army Software Metrics Office (ASMO) is that the members of the organization that implement a measurement program must feel that they were involved in defining the issues to be measured.

The Army learned this lesson early. In 1990, the Software Test and Evaluation Panel (STEP) initiated an issue-driven process to define a standard set of software metrics for all Army programs. Members of the STEP metrics working group defined the Army-wide software management issues based on their experiences in Army software acquisition and test. Table 1 describes the Army issues and the metrics that were selected to address them. The selected set of 12 metrics was implemented in 1991 as a mandatory requirement for all Army programs. As expected, the STEP mandatory metrics program was received with extreme hostility from most acquisition program managers.

Hostility and marginal success were expected because the first objective of the mandatory metrics program was to shock managers to comply with software metrics policy requirements that had been ignored for years. However, the STEP members were surprised to learn that the most significant cause of this hostility was not the financial impact on the projects. Army managers were more upset by their perception that an *outside* organization had mandated how they would do their job. The underlying problem with this approach was that the Army program managers who had to implement the metrics were not involved in defining their issues.

The Army policy for the 12 mandatory metrics was superceded in 1996 with a policy [1] that requires each project to implement a set of software metrics to support its own project-specific issues in six common issue areas. The ASMO has found that the best approach to define project-specific issues is through a series of measurement-tailoring workshops. The workshops provide the forum to allow persons at all levels of the organization to define the project's issues in terms of their own problems, concerns, and lack of information in their day-to-day jobs. Employees tend to support a measurement program if they feel that they had a part in developing the process.

Never change an existing process to obtain a measure. The second most important lesson the ASMO learned is to only use data currently produced by an organization's process. This principle dictates that measurement programs should start small and not require any significant additional resources to implement. Radical change of any process in an organization (business, management, or technical) will usually render the process ineffective for a period of time until the new process is learned and becomes institutionalized. A radical process change may cause employees to no longer understand or support their daily activities. Employees will spend their time relearning their jobs or complaining about the new order. The overall result is that both the organization's new measurement process and other processes objectives are not achieved.

However, the lesson is not that an organization's process should not be improved to be able to provide more effective measures. The point is that radical change of any process is usually destructive. Managers should realize and accept the fact that organizational change to improve a process will be slow. For example, the empirical data that has been collected in the Software Engineering Information Repository (SEIR) shows that the median time to move from Capability Maturity Model® (CMM) Level 1 to Level 2 is 26 months for those organizations that began process improvement programs after 1992. The SEIR is sponsored by the Software Engineering Institute (SEI) and provides an extensive body of empirical evidence on software process improvement. The SEIR can be reached at www.seir.sei.cmu.edu

Measurement provides managers with an important, but limited, opportunity. The greatest benefit of a

Table 1. *Army issues, metrics selected to address them.*

Army Experience Issue	Metric
Software user requirements are not achieved in the code.	Requirements Traceability
Software fails under unexpected operational load levels.	Reliability Breadth of Testing
Unexpected changes are made to user requirements.	Requirements Stability
Software is released without adequate test.	Breadth of Testing Depth of Testing
Excessive design changes are made to the software.	Design Stability
Inadequate cost estimates are overrun.	Cost
Inadequate schedule estimates are slipped.	Schedule
Software problems are not resolved prior to release.	Fault Profiles Complexity
Physical computer resource capabilities are exceeded.	Computer Resource Utilization
The software developer does not have the capability to deliver an adequate product.	Software Engineering Environment

SM The Capability Maturity Model Integration and CMMI are service marks of Carnegie Mellon University.

measurement process is to improve objective communication within an organization. Fred Brooks' experience [2] in developing the IBM Operating System/360 in the 1960s convinced him that "all programmers are optimists." Brooks also found that optimism was pervasive throughout most software development organizations: "When a first-line manager sees his small team slipping behind, he is rarely inclined to run to the boss with this woe. The team might be able to make it up, or he should be able to invent or reorganize to solve the problem."

The ASMO has learned that these observations are still generally true after 35 years. They define the underlying need for measurement in a software development organization. As software can be rewritten and replaced, persons at all industry levels tend to feel that any technical problem can be resolved if they work hard enough. Communication in the software industry on the current status of software products is generally optimistic. The result is that bad news is not easily reported, but is retained at various levels of a software organization because employees feel they can resolve a problem before they will report it.

It is best that customers and managers at all levels realize that any subjective communication will tend to be optimistic. Problems will only be reported when they have grown too big for an employee to admit that he can no longer resolve them. For example, software development projects get behind one day at a time; however, schedule slips are usually reported to a customer only when they are several months behind, and some product must now be delivered. What a manager needs is objective data that is routinely collected as a by-product of an established process in the organization. This objective data will provide unbiased testimony on the status of a process.

Measurement is only one technique in an effective software management process. The process to define the project issues may show that other actions are more important for an organization than collecting data. For example, product quality metrics such as software trouble reports are ineffective if the software complexity prohibits adequate testing. If the system does not require complex software, managers should dedicate their resources to reducing software design complexity before measuring product quality.

Also, if software managers are not willing to make decisions based on software measurement data, a software measurement program is of little use.

Many software measurement reports are only as good as initial estimates of measures' target values. Many software metrics (cost, schedule, computer resource utilization, etc.) report a comparison of planned versus actual values. The most obvious information that is obtained from the metrics data is a comparison of the actual values to the planned values at some point in time. In most cases, this information only reflects how good the initial estimate was to the actual values that are achieved.

Because a manager will gain little by revising the initial estimate, these metrics often provide nothing but bad news that a manager cannot change. Spending too much time and effort to report and evaluate these metrics will provide little return on investment. Remember that the primary return on investment for a metrics program is the information that allows a manager to make an informed decision on an issue.

The underlying fact is, going back to Brooks' advice, that many estimates in software engineering are optimistic and are not based on reality. For example, in a section entitled Gutless Estimating, Brooks made this observation on schedule estimates:

> "Now I do not think software managers have less inherent courage and firmness than … other engineering managers. But false scheduling to match the patron's desired date is much more common in our discipline than elsewhere in engineering. It is very difficult to make a vigorous, plausible, and job-risking defense of an estimate that is derived by no quantitative method, supported by little data, and certified chiefly by the hunches of the managers."

The key to using metrics that are based on an initial estimate is to understand the basis of the estimate and the link to reality. If an initial estimate was unrealistic, users should adjust to the fact that the variances between planned and actual values will always look bad. However, these measures should not be discarded. These measures' users must adhere to one of the lessons that the ASMO has learned to achieve a good measurement process: "Do not focus only on the quantitative value of the data, but capture other information from the objective communication that usually results from the measurement process." For example, if you know that the schedule estimate is unrealistic, encourage persons in the organization to think of actions that may be taken to reduce the adverse impact of the upcoming schedule slip.

Late delivery makes any metrics data useless. The fundamental return on investment in a measurement program is the action that can be taken to effectively manage an issue. Measures provide relatively unbiased testimony on the issue's status, but do nothing to resolve the issue. Therefore, the most common measurement program benefit is early warning of a problem that allows time to avoid it. A slow data reporting process will delay any information that can be derived in time to take corrective action. Late reports may eliminate management options and reduce usefulness of a measurement program. Long delays only report a painful history, and the measurement process becomes a management burden.

The optimum solution is to use the measurement process to support *instant* information through improved communication. Army managers have learned that a defined set of software metrics will identify the important issues in a software process to all members of the organization. If the members of an organization understand the importance of data on status reporting for an issue, they tend to derive and report the information before the data reports are formally delivered.

Improved communication is a valuable objective in managing a software process. By the time measurement data is formally delivered to the appropriate management level, some significant time has already elapsed. A reasonable time between collection and reporting usually is 30 days. A slow measurement process may take two to three times as long. This lapse in time may limit the ability of the manager to take effective action. Data that is reported too late to support management information and action is useless. It is important that all participants in a measurement process understand this principle and report the results of data as soon as they are available.

Actually, the least important elements of an effective metrics program are the data reports and indicators that are eventually delivered. Since the objective is to allow managers to derive information as early as possible, the telephone is the most important measurement tool. If the tool is effectively used, managers at the data source will immediately broadcast any important information on the measured issues. In the best possible measurement process, the phone will ring long before measurement reports are delivered.

Data reports symptoms, not causes. Clever data presentation and intricate analysis methods may not be worth the effort. Data reports should confirm the expectations of the manager who understands the process and the issues that are measured. If the data is a surprise, the manager should pick up the phone and ask what happened. If the manager does not know whom to call to get the answer, that manager needs to learn more about the software process. More information on the underlying process is needed before the manager should use the data to draw conclusions and make decisions.

Data reports should not be used for a management decision without also reviewing other supporting information that provides insight into the software process. For example, a data report showing that some element of a project has exceeded planned cost does not independently allow a manager to derive information on the actual budget status. The reason for the high cost may have been that the organization made an early purchase of additional equipment to support a prototyping effort. Although expenditures are higher than scheduled, the early expense and prototype capability will allow a stable requirements baseline to be defined and will eventually lower the overall project costs. If a data report shows that a project has over planned cost, a manager is able to ask intelligent questions only if that manager understands current events in the project.

The information value of metrics data depends on who reads the data. Metrics data reports provide two limited opportunities to a manager:

1. The data confirms the manager's expectations and assumptions on the current status of an issue.
2. The data is a surprise, and it allows the manager to ask intelligent (issue-driven) questions.

A manager who obtains news on a project only by reading metrics data reports should not make significant management decisions. The manager who reacts to only data reports probably does not know enough about the project to make an informed decision. Management decisions and actions usually must balance several competing objectives and issues within an organization.

Most objective metrics data is influenced by someone's subjective judgment. Quantitative data usually is assumed to be accurate and unbiased because it is expressed as a precise number. However, managers should be aware that every data item that is defined and collected by another person will reflect some degree of subjective judgment. For example, metrics data on the number of design units that have passed integration test usually is assumed to be a precise number, determined only by technical characteristics of the software under test. However, the number of tests that have been passed depends primarily on the criteria used for the test. Examples of changes to integration test criteria follow:

- Level of assembly or the size of software components that are integrated for each test case.
- Number of inputs or conditions required for each test case.
- Level of conditional stress during test (concurrent tasking, shared memory availability, processor utilization, etc.).

The pass/fail criteria for a test is not reported in the metrics data, but any changes may radically drive data up or down. The principle that managers should understand is that the quantities that are reported by metrics data are a by-product of another process that may or may not be stable. Deriving information from metrics data requires a manager to understand that other process.

Not all organizations benefit from a metrics program. The process to identify the project issues will often determine that an organization is just not ready for measurement. The ASMO's experience in administering measurement-tailoring workshops has shown that many organizations must improve other software processes before an effective measurement process can be established.

For example, when selecting issue-driven measures, the data that is currently collected in the organization's configuration management (CM) process should be considered. If the data that is collected in the existing CM process cannot support a very basic measurement program, it is more important to first improve the CM process than to start collecting metrics data. Often, differences in the environments and methodologies within an organization or with software vendors make communication and software data collection difficult, at best.

Not all numbers are equal. Managers should have insight into the measured software project to know what numbers are most important. For example, it is understood that software problems or trouble reports must be assigned a priority according to the impact on the system or project. However, many other metrics should also be provided with a priority ranking. All software requirements, software components, test cases, etc. are not equal in their impact on the software process and products.

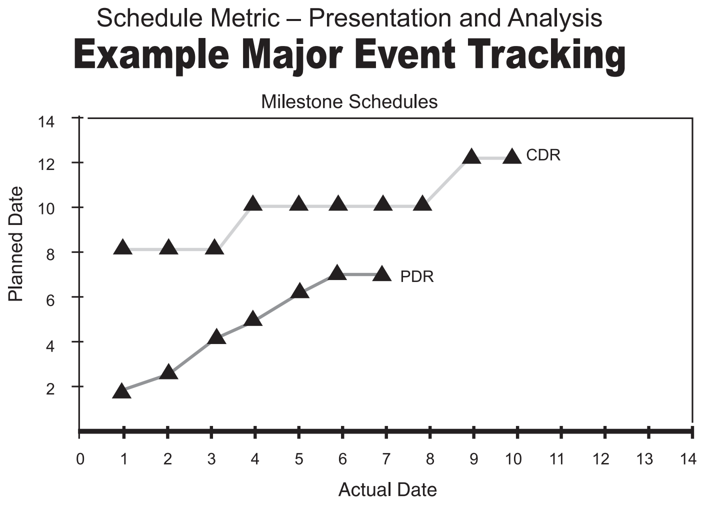
Managers must understand the priority or criticality of a measured element to the overall objectives of the system or project. The guideline is that managers should know which measured elements have the most impact on their project issues.

Data indicators should be tailored to a project issue. When building a data report or graphic indicator, managers should consider the specific kinds of information that is most useful. For example, when measuring schedule progress in a software development effort, the change of dates usually is not as important as the developer's reaction to the schedule changes. The ASMO's recommended display for the schedule metric is provided in Figure 1 as an example. In this indicator, we plot planned versus actual dates of events. Although this indicator is confusing, the data display focuses attention on the changes to future events that are mandated by schedule slips.

The ASMO's experience is that the final delivery date for a software product usually is established by system delivery or marketing promises that have little to do with the schedule needed to develop the software. Early software schedule slips will rarely delay the promised final delivery date. When schedule slips occur, future activities between events are compressed to ensure the planned delivery date is achieved.

The data display in Figure 1 shows that in month six the Preliminary Design Review (PDR) has slipped by five months, but Critical Design Review (CDR) remains within two months of the original schedule. The data display highlights the most important schedule issue. The developer now must complete the detailed design between PDR and CDR in three months, not the six months that were originally proposed.

Figure 1. *The ASMO's recommended display for the schedule metric focuses attention on the changes to future events that are mandated by schedule slips.*



Never implement an expensive measurement program. The only realistic reason that a measurement process will incur some significant cost is that the metrics data is not currently produced by the organization.

Two basic rules that must be observed to define an economical metrics program are:

- Only use the metrics data that is currently available from the organization.
- Do not require a metric to be reported if it would require the organization to immediately adopt or change an engineering or management process.

The ASMO has found that most violations of these rules occur when a customer requires a software vendor to provide metrics data. Often a customer will request data that is an effective practice for software engineering but is not a requirement element for all software processes. A common example of this type of metric is "cyclomatic complexity." This metric is commonly misused when a customer requires this data to be reported, and the vendor's process does not currently measure cyclomatic complexity. To report this metric the vendor must either (1) measure the complexity of design and code after it is completed, or (2) adopt a new quality process that is based on cyclomatic complexity. The new process will require new tools, training, and procedures for the software design team.

The first option to report cyclomatic complexity will produce a number but does nothing to change the vendor's process or the product that is delivered. The second option may eventually improve the vendor's process and the complexity of delivered software. However, while the vendor is learning the new complexity-driven quality process, his overall productivity and product quality may be degraded. The customer should adopt neither option. All managers should follow rule No. 1 and only use data that is currently available from an organization.

Any software engineering organization should be able to report basic configuration management data for little additional cost to a customer. If a software vendor is unable to provide this basic data at reasonable cost, it is clear that the customer has not learned the most important guideline for an effective metric program: "Do not hire a dummy to develop your software."u

References

1. Memorandum, Director of Information Systems for Command, Control, Communications and Computers (DISC4) policy SAIS-ADW, subject: Acquisition Reform and Software Metrics, Sept. 19, 1996.
2. Brooks, Fredrick P., *The Mythical Man-Month, Essays on Software Engineering,* Addison-Wesley Publishing Company, 1975.

About the Authors

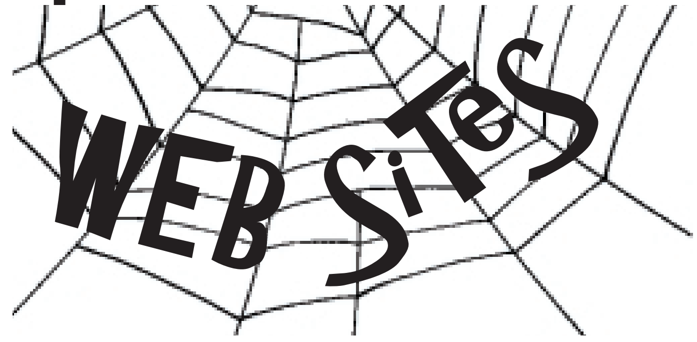
Don Scott Lucero is a software engineer on the headquarters staff of the Army's Evaluation Center. He is responsible for the Army's Software Metrics Office as well as Army Test and Evaluation Command's software test and evaluation policy and methods. Lucero has 17 years of experience working on Army software development projects and has both bachelor's and master's degrees in computer science.

U.S. Army Software Metrics Office, Attn: CSTE-AEC-MA
Park Center IV, 4501 Ford Avenue
Alexandria, Va. 22302-1458
Phone: 703/681-3823
DSN: 761-3823
Fax: 703/681-2840
E-mail: lucerodon@atec.army.mil
Internet: www.armysoftwaremetrics.org

Fred Hall provides training and product assurance engineering support, including software reliability, quality assurance, and systems reliability and maintainability. Hall has supported the Army Software Metrics Office from November 1989 to the present. He has also provided support to the Department of Defense Practical Software and Systems Measurement program. He received a master's degree from George Washington University in 1974, and a bachelor's degree in mechanical engineering from the U.S. Naval Academy in 1970.

Independent Engineering Inc.
4 Old Station Road
Severna Park, Md. 21146-4619
Phone: 703/979-9674
Fax: 703/979-8187
E-mail: fhalliei@aol.com

Process Improvement WEB SiTeS



Software Technology Support Center
www.stsc.hill.af.mil
The Software Technology Support Center (STSC) was established in 1987 as the command focus for proactive application of software technology in weapon, command and control, intelligence and mission-critical systems. The STSC provides hands-on assistance in adopting effective technologies for software-intensive systems. It helps organizations identify, evaluate, and adopt technologies that improve software product quality, production efficiency, and predictability. STSC uses the term technology in its broadest sense to include processes, methods, techniques, and tools that enhance human capability. Its focus is on field-proven technologies that will benefit the Department of Defense (DoD) mission.

Software Engineering Institute
www.sei.cmu.edu
The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the Department of Defense to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. SEI helps organizations and individuals to improve their software engineering management practices. The site features a software engineering management practices work area that focuses on the ability of organizations to predict and control quality, schedule, cost, cycle time, and productivity when acquiring, building, or enhancing software systems.

Software Process Improvement Network
www.sei.cmu.edu/collaborating/spins/spins.html
The Software Process Improvement Network (SPIN) is a leadership forum for the free and open exchange of software process improvement experiences and practical ideas. It promotes achieving higher levels of process maturity, software quality, and mutual respect. The goal is to help sustain commitment and enhance skills through an active program of networking, publications, recognition of excellence, and mutual support. The individuals are organized regional groups called SPINs who meet annually at the Software Engineering Process Group conference.

DoD Data and Analysis Center for Software
www.dacs.dtic.mil
The Data and Analysis Center for Software (DACS) is a Department of Defense Information Analysis Center. It is a software information clearinghouse that serves as an authoritative source for state-of-the-art software information and provides technical support to the software community. A software process improvement topic area provides a summary of findings on return on investment from software process improvement (SPI). Users will be able to see the cost benefits that can be achieved through performance of a SPI program, including success stories.

Software Engineering Process Office
http://sepo.nosc.mil
The Software Engineering Process Office (SEPO) is the software engineering focal point for the Space and Naval Warfare Systems Center, San Diego (SSC SD). SEPO provides software engineering processes and consulting services to projects, conducts and facilitates software engineering training, and acts as a software engineering clearinghouse for SSC SD. SEPO's goal is to raise software capability maturity in a concerted effort among the SSC SD software community, industry partners, and other government associates.

The Software Engineering Laboratory
http://sel.gsfc.nasa.gov
The Software Engineering Laboratory (SEL) is an organization sponsored by the NASA/Goddard Space Flight Center (GSFC). It was created to investigate the effectiveness of software engineering technologies when applied to the development of applications software. For more than two decades the SEL has been collecting and analyzing software development metrics from development projects within the GSFC Flight Dynamics Division. Its other sponsors are the Software Engineering Branch, University of Maryland, and Computer Sciences Corp., Space and Earth Technology Systems.

Software Productivity Consortium
www.software.org
The Software Productivity Consortium is a nonprofit partnership of industry, government, and academia. It develops processes, methods, tools, and supporting services to help build high-quality component-based systems, and continuously advance systems and software engineering maturity pursuant to the guidelines of all the major process and quality frameworks. The consortium's technical program builds on current best practices and IT to create project-ready processes, methods, training, tools, and supporting services that meet our customers' needs in systems and software development. The site has a quarterly news letter and on-line news section.

European Software Process Improvement Foundation
www.espi.co.uk
The European Software Process Improvement (ESPI) Foundation promotes good software practice through software process improvement. It provides information and training to assist those beginning the process improvement journey, and facilitate exchange of knowledge and experience between practicing organizations. The site has a latest news section and a monthly-featured periodical.

EGroups
www.egroups.com/group/spi
This particular eGroup site is an open forum for exchanging software process improvement information. If you like using the Capability Maturity Model® (CMM) for software, you have come to the right place. There is an electronic copy of the CMM in the files area, some good links, calendar items, and polls to take.

Software Process Improvement and Capability dEtermination
www.sqi.gu.edu.au/spice/contents.html
Known as SPICE, this is an international initiative to develop an international standard for software process assessment. A draft standard for this was achieved in June 1995 with the release of Version 1. The SPICE trials are now the principal focus of the project. Five international technical centers have been established to coordinate the effort on the project. More information about the trials is available on this site, including how to register as trial participants, more about the SPICE Assessor Training syllabus, or the annual Inter-national SPICE Symposium, and various articles and books.

Ada in the 21st Century

Benjamin M. Brosgol
Ada Core Technologies

Although Ada does not receive the publicity of some other languages, it is alive and well in a broad range of applications that include hard real-time embedded systems, safety critical programs, desk-top graphical user interface environments, and the Internet. The first internationally standardized object-oriented language, Ada is a language of choice in many environments where reliability is critical. It continues to play a significant role in teaching and research at computer science departments around the world. The Ada infrastructure, including the validation testing for implementation conformance with the standard, has been successfully transitioned from the government to the private sector. The language is evolving smoothly to meet the demands of the 21st century.

Editor s Note:
When Donald Reifer's *Is Ada Dead or Alive Within the Weapons Systems World* was published in December 2000, we said, "Ada has been surrounded by controversy almost since its inception. In this issue we offer one perspective on the current state of Ada and how this affects technology decisions for weapons systems. An upcoming issue will provide an opposing point of view." Here is another Ada report. Letters to the editor follow.

The U.S. Department of Defense (DoD) had two major goals when it sponsored development of the Ada programming language in the early 1980s. First on the technical side, Ada was designed to meet the requirements of large-scale, reliability-critical applications (specifically, embedded real-time systems). Second, Ada was intended as a common DoD language to reverse the language proliferation trend that was making source code portability all but impossible. Ada was the product of an international effort led by a team from France and involving representatives from government, private industry, and academia.

Ada became an international standard (ISO) in the mid-1980s and was revised (with full support for object-oriented programming) in the mid-1990s. Its advocates observe that Ada has met its initially intended goals, and that it is an effective and methodology-neutral language for teaching software engineering. Ada implementations, tools, and libraries are available on a wide variety of platforms through a number of vendors. The language is especially applicable to safety-critical and high-integrity applications, and it is heavily used in industries such as transportation. Ada offers more security than languages such as C and C++, and better efficiency and a simpler run-time model than Java. Indeed, an impartial observer, the respected real-time authority E. Douglas Jensen, offered this technical assessment:

> "Ada 95, including its Real-Time Systems Annex D, has probably been the most successful real-time language in terms of both adoption and real-time technology. One reason is that Ada is unusually effective (among real-time languages and also operating systems) across the real-time computing systems spectrum, from programming-in-the-small in traditional device-level control subsystem, to programming-in-the-large in enterprise command and control systems [1]."

Nevertheless, Ada has not enjoyed the commercial success or publicity of other languages. Occasionally the question is posed whether Ada is "dead or alive." This article offers a snapshot of the Ada industry as of late 2000, concludes that Ada is indeed alive (especially in the domain for which it was initially intended), and offers some predictions on its future.

The first several sections of this article provide some perspective: reviewing the history of the DoD's Ada policy, showing how the choice of a language affects productivity, and addressing the issue of commercial penetration as a factor in choosing a language. The paper then summarizes the supply side of Ada (compiler vendors, tools, bindings, etc.), looks at how Ada is being used today, identifies the main reasons it is chosen, and describes the lessons learned from user experience. Lastly, the article discusses Ada's role in teaching and academic research.

Ada Policy in the DoD

Ada is one of a small number of programming languages that is accompanied by a comprehensive test suite that reflects a compiler's conformance with the language standard. This suite, originally called the Ada Compiler Validation Capability and more recently designated the Ada Conformance Assessment Test Suite, comprises several thousand individual tests that exercise the core language, the standard library, and the specialized needs annexes. A compiler that successfully passes all applicable tests in the current suite is labeled *validated.* Although the evolutionary nature of the test suite precludes trying to standardize the tests themselves, the test procedures became an ISO in 1999.

From 1983 until 1997, the DoD's Ada Joint Program Office promoted a stringent Ada usage and validation policy requiring Ada on weapons systems applications and specifying that any project using Ada employ a validated compiler to produce delivered code. The policy had dual consequences. The Ada mandate seeded the market encouraging hardware vendors to supply Ada implementations and spurred a furious competition among Ada compiler vendors. However, it was imposed before Ada environments had become mature. Thus early experiences were often disappointing, especially given the pent-up demand and overly ambitious expectations that often accompany a new technology. A required waiver for not using Ada—and justifying the decision on projected cost saving—was widely ignored. Moreover, if an Ada project failed, it was convenient, even if not often justified, to blame problems on Ada language or implementation.

The validation policy also had mixed effects. On the positive side, it forced compiler vendors to focus on implementing the full language versus a subset. This was instrumental in realizing a high degree of portability for Ada source code across a wide variety of platforms—a benefit that should not be underestimated. Ada has had an excellent track record for source code portability. In some areas such as task scheduling, it is significantly more portable than Java despite the latter's claim of "write once, run anywhere."

At least in early Ada industry, achieving necessary validation sometimes led to distorted priorities as compiler vendors paid less attention to other important traits such as efficient run-time performance and support for external tools. By the mid-1990s the DoD took a new look at compiler validation. They recognized its importance in preventing the proliferation of subsets and dialects—real problems with earlier languages. But they perceived this was no longer a critical risk since the range of language choices was much narrower.

The DoD ultimately rescinded both the Ada mandate and the validation requirement in April 1997. Ironically, Ada was required before it was ready, and then the requirement was removed once implementations had matured. Concluding that the Ada language no longer required external support, the DoD subsequently dissolved the Ada Joint Program Office. Ada infrastructure elements—maintenance of the validation test suite and oversight on the language's evolution—were taken over by the Ada Resource Association, a vendor trade group described below.

Programming Language Effects

A programming language is not a magic bullet that will prevent or cure software ills. It should be regarded as one element of technology used in a software project that is complemented by additional factors: tool support, people issues like developers' talents and ability to work on a team, and the efficacy of the development process. Indeed, it is sometimes argued that the choice of a specific language is one of the less critical factors in terms of effect on project outcome.

A good language will not turn unskilled developers into software wizards, but it can leverage the talents of a skilled team making them more productive. Sometimes an inspection of technical features makes this obvious. For example, Ada allows the integer value 2^{63}-1 to be expressed in hexadecimal format as 16#7FFF_FFFF_FFFF_FFFF#, which is easy to read with the underscore character serving to separate the 4-digit groups. In contrast, C (and also C++ and Java) require the same value to be written much less clearly as 0x7FFFFFFFFFFFFFFFL. The absence of a separator character in the number is of no consequence to a compiler but is an invitation for human error.

It is difficult to conduct comparative quantitative studies of programming languages in an objective manner, especially for large projects. However, an impartial and thorough analysis [2] documented a significant difference in productivity between Ada and C on the components of the Verdix VADS product line. This development comprised roughly the same amount of code in the two languages. Zeigler's study considered all the relevant factors (such as the effect of programmer skills) and concluded that Ada performed approximately twice as well as C (i.e., costs for Ada were half that for C). Some of the reasons cited were Ada's additional compile-time checking, its higher-level features, and the Ada culture that encouraged up-front design. Interestingly, the study observed that using C++ rather than C would not change the underlying result, since bug rates in C++ were higher than in C. It concluded:

> "Our data indicates that Ada has saved us millions of development dollars. For every development dollar, we could make a case for another three dollars for customer support, sales, marketing, and administration costs ...."

This result should not be surprising since Ada was specifically designed to save life-cycle costs through software engineering support. Quantitative data provides evidence that this goal has been met, even if percentage of improvement varies per project.

Commercial Penetration

For a variety of reasons Ada has not had the commercial penetration of other languages. This raises an issue: How important is a language's popularity when an organization needs to choose a technology for future software development? A recent article [3] argued that commercial success should be a guiding factor in language choice for large-scale DoD weapons systems. We believe this is the wrong way to look at the issue for several reasons:

- Commercial success is almost always more strongly influenced by marketing factors than by technical characteristics. From a software engineering perspective, Algol 60 and Pascal were better languages than Fortran and BASIC but did not achieve the latter pair's high market share. An enterprise needs to look beyond the popularity of a language and make sure that it satisfies the technical requirements of the application.
- In the commercial software market the way to succeed is to release products that are good enough, at a price that customers will accept, in a timely fashion. In a market-oriented software industry, reliability is not the most important criterion. This is reasonable for many kinds of applications; if a Web browser locks up or a word processor crashes, the damage is relatively localized. But weapons systems do not have such luxuries.
- Times change and what is popular one day may be a nostalgic memory the next. If commercial penetration had been the deciding factor in earlier times, then weapons systems would have chosen COBOL during the 1960s and 1970s, and BASIC in the 1980s.
- Relative market share data can lead to the erroneous conclusion that non-market-leading technology has failed. This is partly due to the media's tendency to portray commercial competition as a sports event with only one winner. In fact, a number of technologies that do not receive much publicity today are indeed financially successful: Examples are OS/2 in the operating systems arena and PL/I and Pascal in the language area. Judging from the infrequent press coverage these technologies receive, one would deduce that they are all but extinct. Yet in fact they are still in heavy use.
- The market dynamics of a commercially popular product sometimes conflict with the requirements of a major long-term project such as a weapons system. To stay ahead of the competition, a tool vendor needs to upgrade and enhance its product at regular intervals. However, developers cannot afford the risks that disruptions of the underlying tool base would bring. They instead need to baseline a particular version for use

throughout the project (or at least until a major upgrade). Support for older versions is not generally a high-priority item for vendors whose products have a high market share.

Perhaps most importantly, commercial penetration itself is not what is important, but rather the effects it seems to bring—an adequate and varied supply of compilers and tools[1], and a pool of programmers already skilled in the language. Let's look at how Ada fares in regard to these factors.

The Commercial Players

Principal Ada compiler vendors are the members of the Ada Resource Association trade group: Ada Core Technologies, Aonix, Averstar, DDC-I, Green Hills Software, OC Systems, and Rational Software Corp. Several smaller companies such as Irvine Compiler Corp. also sell Ada compilers into specialized markets. All of these companies are established players in the field. The oldest were founded in the early 1980s, and the youngest was formed in the mid-1990s. The companies vary in their business models, with different mixes of off-the-shelf products, contract work, and support/consulting services.

Although the number of Ada compiler vendors is smaller today than it was 10 years ago, this is a common phenomenon throughout private industry as mergers and acquisitions have become standard business practice. Moreover, there is not a large number of compiler vendors for more widely used languages.

The Ada compiler industry today can be characterized as fairly stable with competition on the most popular platforms. The size of the Ada market is difficult to quantify with any precision. Based on an informal analysis by the Ada Resource Association and public data provided by some vendors, the worldwide market is about $80 million or so annually. This figure has been fairly steady during recent years and is not likely to change significantly in the near term.

Ada is available across a wide range of platforms. A partial list of Ada '95 compiler host environments includes Compaq/Digital Alpha (OpenVMS, UNIX), Concurrent/PowerMax, HP9000/HP-UX, IBM 390/MVS, Intel x86 (Windows-NT/9X/2000, Linux, OS/2, UNIX), PowerMac/Tenon, PowerPC/AIX, RS6000/AIX, SGI/IRIX, Siemens-Nixdorf/SINIX, SPARC/Solaris, and Vax/OpenVMS.

Target environments include all the above hosts plus ADI-SHARC/EONIC, ADI-21020/Bare, HP7xx/HP-RT, IBM390/CICS, i960/HAOS, Intel/ETS Intel/RTLinux, the Java Virtual Machine, M68k/VXWorks, MIPS/VXWorks, Nighthawk/6800, PowerPC/Bare, PowerPC/LynxOS, PowerPC/VXWorks, and also optimized ANSI C.

Several compiler companies use automated code generator technologies and layered run-time systems with a clear interface to target-dependent components. This makes it relatively easy to produce Ada compilers targeted to new chips and operating systems/real-time kernels. Implementing Ada on a new target is roughly comparable in effort to implementing C++ (except for needed concurrency support in the run-time system since C++ lacks this facility).

Ada compilers are supplemented by an assortment of productivity-enhancement tools, component libraries, and bindings to popular software systems. The compiler developers supply some of these: source-level debuggers, graphical user interface (GUI)-based tool environments, browsers, tools for memory monitoring, GUI builders, and target OS interfaces. Third-party developers provide others. These products include source analyzers (DCS, McCabe & Associates, Vector Software), formal verification tools (Praxis Critical Systems), automated design tools (TNI), interfaces to X-Windows (TopGraph'X), Ada support for CORBA (TopGraph'X, Objective Interface Systems), real-time embedded graphics (DCS), a *thick* binding to the Windows API (RR Software), and many others.

The availability of Ada Core Technologies' GNAT Ada compiler under the General Public License (GPL) of the Free Software Foundation has inspired the development of a variety of tools and bindings that are also available under the GPL. A number of these are available though the Ada Power Web site (see On-Line Resources). Examples include a COM/DCOM/COM+ framework and bindings to the Windows API.

A wealth of libraries is available in other languages: mathematical algorithms in Fortran, low-level communications functions in C, network-ready classes in Java, etc. This has been cited as an Ada weakness, but in fact one of Ada's unique strengths is the ability to interface with software in other languages in a standard and straightforward manner. Tools for several languages are available that automatically generate the appropriate glue specifications that provide the Ada interface to foreign components. If you need to mix C and Fortran, this is easier in an Ada environment than in either a C or a Fortran compilation system.

Ada is unique among programming languages in having a standardized interface for tool developers: the Ada Semantic Interface Specification (ASIS). This high-level interface contains relevant information about an Ada program in a format that is convenient for processing. Several of the tools previously mentioned work from an ASIS version of the source program. Moreover, most Ada 95-compiler vendors have made an effort to integrate their development environments with non-Ada tools such as configuration management systems. This results in Ada development environments that are as comprehensive as other languages with higher commercial penetration.

Another side effect of a language's market share is the supply of skilled programmers. It is a tautology to observe that the more popular the language, the higher the number of programmers who know it. How significant is this issue for a project that will continue over many years with a large team that produces perhaps a million lines of code?

On one hand it cannot be denied that starting with a team already familiar with the programming language will save some up-front costs. But this is largely a red herring issue. Most large projects need to reserve time for new team members to assimilate possibly unfamiliar technology. Any programmer who is skilled in C or C++ can come up to speed in Ada through a variety of approaches, including on-line tutorials, books, or professional courses. A professional programmer in any language can become Ada-proficient in a five-day course. In the process he or she will also learn software development techniques (package design, use of tasking features, and reuse through generics) that carry over to other languages.

While Ada may not have the commercial penetration of other languages, this should not be a major factor in choosing it for a large, long-lived project. Indeed, basing a decision on commercial popularity may even increase some risks due to the mismatch between market dynamics of the software industry and the requirements of enterprise-critical software. Ada tools and environments are also mature, competitive, widely available, and the apparent training gap in the supply of professional programmers is a problem that is easily addressed.

Ada Usage

Ada applications range from hard real-time processing to commercial desktop tools. Ada users include small start-ups, large established firms, all points in between, and encompass government agencies, the private sector, and academia around the globe.

The language has shown particular strength in the safety-sensitive domain, especially in the transportation industry. Boeing used Ada heavily for their 777 aircraft and continues to require it for all software of the highest safety criticality certification levels—DO178b levels A and B. Ada has been used on subway systems, including the London Jubilee, and the recent extension of the Paris Metro, and for work on the Carnarsie line of the New York City subway. It has been used on the French TGV trains and metrorail systems in Europe, Asia, and Latin America. It has also been used in commercial shipboard control.

Other domain uses include the TV and entertainment industry, medical computing, communications network switches, and financial and information systems. It has been used in industrial control, including safety-critical nuclear reactor shutdown, and in desktop software. It has of course been used traditionally for military programs in the United States and allied countries. Ada has also seen heavy use by nonmilitary governmental agencies such as NASA and the European Space Agency. There are hundreds of millions of lines of Ada code in operation today worldwide and in outer space.

It is beyond the scope of this article to document even a small fraction of these applications in much detail. Nevertheless, several consistent themes emerge among the reasons that Ada has been chosen, and from the experience and lessons learned that have ensued. Further information may be found through links at Professor Michael Feldman's Web site (of the George Washington University), and the sampling of success stories on the Ada Resource Association's Web site (see On-Line Resources).

Why Ada?

Since its inception, the decision to adopt Ada has been a conscious choice commercially, and also for military applications after the Ada mandate was eliminated. In all cases the main reasons cited are the same: Ada was deemed to be a more reliable language than the alternatives, with run-time performance meeting the efficiency requirements of the application, and sufficiently mature compilers and support tools.

In some cases, bad experience with buggy or nonportable software written in other languages made Ada an attractive choice. In other cases, particular Ada functionality guided the decision (e.g., concurrency support, low-level and real-time features, or interfacing facility). Its status as an internationally standard language, backed by a validation suite measuring a compiler's compliance, has also been a relevant factor.

These reasons, especially Ada's security and focus on checks at all levels, have been convincing factors in the safety-critical and high-integrity software domain. A number of applications such as the French TGV rail system have found that Ada and formal methods make a happy marriage. Several Ada compiler vendors directly support safety certification through certified run-time kernels and other means. Ada is at the forefront of safety-critical technology with its Ravenscar profile [4], a reduced set of tasking features whose implementation can be certified against the highest levels of safety criticality. This profile strikes a delicate balance. It is restricted enough to allow a simple, efficient, and certifiable implementation, yet powerful enough to express common real-time idioms such as periodic and event-driven activities.

The safety-critical market is a small but important sector, and one where Ada continues to see strong interest.

Experience and Lessons Learned

Users have reported that expected Ada benefits have largely been realized: fewer bugs, easier maintenance, and higher portability than other languages. If there has been any disappointment, it has generally not been with the language or the compilers and tools, but rather with Ada's slow rate of commercial penetration and the resulting smaller supply of Ada-knowledgeable developers.

Projects that have chosen Ada tend to continue using the language as their systems evolve. This is neither surprising nor unique to Ada; if a technology works, there is no compelling reason to spend time and money converting to something else. A corollary is that many Ada 83 projects are continuing with Ada 83—using a baselined compiler or an Ada 95 compiler with an Ada 83 option—rather than moving to Ada 95, although Ada 95 is generally used for major upgrades.

Ada and Other Technologies

User experience shows that Ada fits well with modern technologies. In distributed applications and components Ada is supported for CORBA, and also for frameworks such as Microsoft's COM, DCOM, and COM+. In the free software/open source community, the GNU Visual Debugger, a new tool that will be part of the standard GNOME desktop environment, has an Ada graphical toolkit as its basis. In the safety and security domain, a seminal report on the relationship between Ada language features and techniques for integrity assurance identifies how the level of certification can affect the choice of features, and vice versa.

Ada historically and presently continues to influence many other technologies. Its exception handling, generics, and packages affect the design of exception handling (in C++, Eiffel and Java), and templates and namespaces in C++, respectively. Ada's real-time features directly influenced the real-time extensions proposed for Java. The Ravenscar profile influenced the Java real-time core High-Integrity profile. Ada's picture string localization affected choices made in COBOL, and the structured Ada syntax influenced PL/SQL.

Ada and Academia

A combination of factors affects the selection of a programming language for teaching and research. Several are technical:

Pedagogy. Does the language reflect sound software engineering practice (encapsulation, abstraction, object orientation, genericity, etc.)?

Teachability. Can the language be partitioned such that simple concepts can be covered first and more advanced topics later, with a minimum of forward references? Are textbooks and supporting educational material available?

“Although the DoD removed the Ada mandate in 1997, the Ada market has been fairly stable in recent years. Projects using Ada have tended to stay with the language as their systems evolve. Ada works, and the job of converting to another language is not worth the cost.”

Applicability. Does the language reflect the state of the art in the software industry and facilitate interoperability with elements such as modern windowing systems, network software, etc.?

Generality. Does the language span a variety of domains (such as systems programming, real-time applications, enterprise software, etc.)? Does it fit in with current research areas (formal models, proofs of correctness, parallelism, and distributed computing)?

Portability. Are programs easily ported across different hardware platforms (for example UNIX, Linux, and Windows)?

Tool Quality. Are compilers and supporting tools available that are applicable in a teaching/research setting (for example, with good diagnostic messages, an easy-to-use interactive development environment, and access to the source code of the components)?

Other factors are nontechnical:

Marketability. Will learning the language increase students' employment prospects?

Price. Are low-cost or free compilers and environments available?

Against this backdrop it is useful to look at Ada's history in academia. During the requirements phase and the development and review of the preliminary designs from 1977 to 1983, the academic community was heavily represented. Consultants from computer science departments at major universities in the United States and abroad helped shape the final design. There was some hope in the DoD that Ada would replace Pascal as the dominant language for teaching introductory programming.

On the technical side, Ada offered significant benefits such as language support for encapsulation, concurrency, and genericity, and a standard input-output library. However, several factors prevented Ada from realizing widespread penetration. The compilers available for Ada 83 tended to be expensive. Although the major compiler vendors gave academic discounts, the price still tended to be too high for most university budgets. Also, Ada was not widely supported on one of the primary machines then used in academia, the Macintosh.

Ada 83 did not support subprograms as data objects or parameters, making it complicated to express applications such as mathematical integration and awkward (and nonportable) to realize callback. As the 1980s drew to a close a language named C++ began to attract attention as a way to bring object-oriented programming (OOP) into the mainstream. Ada 83 supplied the major elements of object orientation but intentionally, in the interest of avoiding the need for implicit storage reclamation, fell short of full support.

Realizing the importance of penetrating academia, the Ada 95 effort attempted to address these issues. Complete support for OOP was provided; whatever can be done in languages such as C++, Java, and Eiffel can be done in Ada 95. Subprograms in Ada 95 are first class data objects, and the common callback idiom is easily and portably expressed. Most importantly, the Ada 95 project sponsored the development of the original GNAT compiler at New York University, based on the Free Software Foundation's General Public License. Thus an open-source Ada 95 compiler, free in both pricing and usage senses, was available when the language was standardized in late 1994.

Ada 95 has addressed the necessary technical issues for success in academia. It reflects sound methodology and, unlike Java, supports several traditional approaches and not just OOP. Indeed, it was the first widely-used language to be designed based on software engineering principles. The language can be effectively taught in several tiers: introductory concepts (the “Pascal subset” of data types and data structures, subprograms/algorithms); encapsulation; abstraction (generics); concurrency; and OOP. A number of high-quality books are available, including several targeted at universities. A CD-ROM with Ada resources is distributed annually by the SIGAda technical society. Ada is an ISO and a highly portable language. Free or low-cost (but high-quality) compiler implementations are readily available on platforms common in the academic community.

These traits have made Ada 95 an attractive option at a large number of colleges and universities around the world, and usage in computer science programs appears to be fairly stable. According to data gathered by Professor Michael Feldman (See On-Line Resources), approximately 150 institutions worldwide cover Ada in their computer science curricula, around 75 percent introduce Ada in a foundation course, and the remaining 25 percent cover it in an upper-level course. These figures have been consistent since 1997.

Of course, criteria beyond technical features play a role in language selection. The most obvious factor in the past 10 years has been the rise in usage of C++, and more recently Java. Universities often try to ensure that their offerings are relevant in the job market and thus will tend to teach subjects and technologies that reflect trends in the broader computer industry. This is not a new phenomenon; in the 1970s Fortran was heavily used for teaching introductory programming in universities even though Algol 60 was arguably the better choice pedagogically. It is thus not surprising now to see heavy adoption of widely-used languages such as C, C++, and Java in computer science curricula. Nonetheless, considering the substantially greater publicity that these languages receive, Ada continues to be a language of choice for teaching, research, or both at many colleges and universities.

Experience teaching Ada has been positive. In an article [5] comparing the performance of students in different introductory computer science courses at the U.S. Military Academy, the authors observed that students did better with Ada than with

Pascal, and pointed out that this experience was duplicated at the U.S. Air Force Academy. Another article [6] relating the author's experience conducting a real-time embedded systems lab course reported that the students were far more successful in Ada than in C.

Beyond its advantages as a teaching language, Ada is also proving a useful vehicle for research. Here are some examples:

- *Ada and Real-Time Systems*: Florida State University, York University (U.K.), the Technical University of Madrid (Spain), and the Naval Postgraduate School.
- *Ada and Distributed Technology*: University of Brest (France).
- *Formal Methods and Ada for Safety Critical Software*: Uppsala University (Sweden).
- *Tools and Components*: U.S. Air Force Academy

In summary, although Ada is not about to overtake better-known languages in academia, it has a strong and energetic following and is not about to disappear. Ada educators have been regularly providing experience reports at the annual Association for Computer Machinery Computer Science Education conference. Computer science faculty members are aware of Ada and its role in the curriculum.

Conclusion

Although Ada does not receive the publicity of other languages, it continues to play an important role in the software industry, both directly (through actual usage) and indirectly (through effects on other technologies). Ada has proved to be particularly strong in long-lived, reliability-intensive applications. This is hardly a surprise since the language was initially designed for this area. Although the DoD removed the Ada mandate in 1997, the Ada market has been fairly stable in recent years. Projects using Ada have tended to stay with the language as their systems evolve. Ada works, and the job of converting to another language is not worth the cost.

As we look toward the future, there are reasons for optimism. Arguably the period when Ada was most at risk was just after the Ada Joint Program Office's closing. There was uncertainty concerning the support of the necessary infrastructure for Ada's continued evolution. But the Ada Resource Association has successfully taken over this support role, in a smooth transition of responsibilities from a government organization to the private sector.

The Ada standard continues to evolve in an orderly fashion. The Ada Rapporteur Group, a collection of language experts in ISO's Ada Working Group, is considering a number of proposed extensions, including a mechanism that will make it easier for Ada to interface with Java classes.

The key to significant new growth for Ada is expansion in academia. Ada is well poised to make new inroads with documented advantages as a language for teaching software engineering, a track record of success as a vehicle for research (especially in the real-time domain), and with an open source Ada compiler technology available. In short, Ada has fulfilled the goals that the DoD had established for it at the outset of the project almost 25 years ago. It promises to continue that fulfillment in both the near term and long range.u

References

1. Bollella, G., et al., *The Real-Time Specification for Java*; Addison-Wesley, 2000.
2. Zeigler, S.F., Comparing Development Costs of C and Ada, March 1995 [www.adaic.com/docs/reports/cada/cada_art.html].
3. Reifer, D., et al., Is Ada Dead or Alive within the Weapon System World?, CrossTalk, December 2000.
4. Burns, A. The Ravenscar Profile; *Ada Letters*, Vol. XIX, No. 4 (1999), pp. 49-52. Available as www.cs.york.ac.uk/rts/papers/p.ps
5. Hamilton, J.A., Jr., J.L. Murtagh, J.L., Zoller R.G. Programming Language Impacts on Learning, *Ada Letters,* Vol. XX, No. 3, Sept 2000, p. 18.
6. McCormick, J., Software Engineering Education: On the Right Track with Ada, *Ada Letters*, Vol. XX, No. 3, Sept 2000, pp. 47-48.

Note

1 A choice of suppliers is not always a consequence of commercial success; an obvious illustration is a proprietary but widespread product such as Microsoft's Windows.

Additional Readings

- N. Audsley, *Ada Yearbook Millennium Edition,* Ada Language U.K. Ltd; York U.K., 2000.
- Language Impacts on Learning, *Ada Letters*, Vol. XX, No. 3, Sept. 2000, p. 18

On-Line Resources

- ACM SIGAda technical society: www.acm.org/sigada
- Ada Resource Association: www.adaresource.org
- David Botton's Ada Power site: www.adapower.org
- Professor Michael Feldman's summary of Ada usage: www.seas.gwu.edu/~mfeldman/ada-project-summary.html
- Professor Michael Feldman's summary of Ada in academia: www.seas.gwu.edu/~mfeldman/ada-foundation.html
- Usenet newsgroup: comp.lang.ada

About the Author

Benjamin M. Brosgol is a senior member of the technical staff of Ada Core Technologies Inc. with more than 25 years' experience in the software industry. He has been involved with Ada since its inception, as a language designer, educator, implementer and user; he was the principal author of the Information Systems Annex in the Ada 95 standard. Brosgol is currently chair of the Association for Computing Machinery's Special Interest Group on Ada. He has presented Ada papers and related technologies at conferences both in the United States and abroad. He was awarded a Certificate of Distinguished Service and a Certificate of Appreciation by the Department of Defense for contributions to the Ada language effort. He has a doctorate in applied mathematics from Harvard University and a bachelor's degree in mathematics from Amherst College.

Benjamin M. Brosgol
Ada Core Technologies
79 Tobey Road
Belmont, Mass. 02478
brosgol@gnat.com

Letters to the Editor

Dear CrossTalk:

The Ada article in the December 2000 issue of CrossTalk is a good example of government folks letting contractors blow smoke where it doesn't belong. The deceit is subtle, but I'll try to expose it.

First, as always, they are comparing Ada to C++. But which C++ are they talking about? Borland C++ is different from Microsoft C++, which is different from Symantec C++. The language was standardized in 1999, but no one is following the standard. The standard is weak, and is already being studied for changes. Table 1 of the article should drop the degree of standardization for C++ down to 2 or 3, and object-oriented (OO) support down to 3 or 4; C++ is not a very good OO language. In fact, C++ isn't a very good language at all! And has anyone beside myself noticed how much Delphi code is in those C++ journals? I don't expect C++ to last; it would be gone already if it weren't for the well-funded Microsoft propaganda machine. Microsoft has recently come out with C#; even they see the writing on C++'s grave marker.

Second, they didn't look very hard for compiler/tool availability. From the article, I got the impression they went to one Web site. I recommend they try Adapower.com, or any of the other great Web sites that support Ada. Ada compilers are as cheap as any these days, including free. But $1000 will get one a good compiler with lots of documentation and graphical user interface (GUI) support.

In Table 3, I will agree that Ada training is scarce, but the last two rows in that table contradict each other. Good Ada programmers are hard to find, but I maintain that there is no such thing as a good C++ programmer. If they were good, they would know better.

The Tri-Ada conference is smaller these days, but it does exist. The ACM doesn't have a special interest group on C++.

My personal experience as a government software engineer is that software engineers want to use C++ because it will make them more marketable on the outside, period.

They don't care, and in most cases don't even know, that it is an inferior language. They pick up a trade journal, turn to the back and count job offerings that list C++ as a requirement, then reason that it must be a good language.

I have heard of a program in C that won't compile with the newer compilers because the language has changed so much. Much searching was done to find an old version of the compiler. I am aware of a similar problem with a C++ program. To be fair, I am also familiar with an Ada program that didn't upgrade well when the operating system changed, and most of the problem was in the GUI. They would have had the same problem if the software had been in C to C++.

Ada 95 greatly improved on Ada 83. The language is here to stay.

– Dennis Ludwig, Electronic Engineer, Warner Roberts-ALC

Editor,

The article *Is Ada Dead or Alive Within the Weapons System World?* contains three Figures showing graphs over time. It says "Figure 1 summarizes our findings relative to the availability of vendors, compilers and tools. This chart and Figures 2 and 3 were developed using public data available on www.adahome.com …."

Since the owner of www.adahome.com stopped maintaining it some time ago, the currency of statistics derived from it is in grave doubt. I ran a Web crawler on www.adahome.com that found 5,600 files, 60 percent modified in 1996, and 5% (280) in 1998 to 2000. The same program found 1,300 files on www.adapower.com; 97 percent (1,270) modified from 1998 to 2000. The activity shown in the figures clearly reflects a fading four-year-old snapshot data source, not current reality.

– Tom Moran, Decision Aids

Editor,

The article by Reifer, Craver, Ellis, and Strickland illustrates several deficiencies in the DoD software management process:

1. The article employs an obsolete source [www.adahome.com] instead of a current one [www.adapower.com].
2. It fails to mention or take into account both the fragmentation of the C languages and the Web revolution.
3. It primarily describes a Delphi exercise rather than being based on statistical process control data.
4. It does not describe the reasons for failure to transfer Ada to the civilian economy.

Java and C# were developed because of C++'s gross deficiencies. In Table 1 of Reifer et al. C/C++ and Ada are given the highest value of 5 for object-oriented support. In fact, the claimed great virtue of C++ was that it extended C to permit object-oriented programming with its potential for reuse. The true utility of C++ is summed up by this quotation from Microsoft's C# Web site [1]:

> "Yeah, yeah. C++ is object oriented. Right. I've personally known people who have worked on multiple inheritance for a week, then retired out of frustration to North Carolina to clean hog lagoons. That's why C# ditches multiple inheritance in favor of native support for the COM+ virtual object system. Encapsulation, polymorphism, and inheritance are preserved without all the pain."

In fact, an Ada generic that contains a tagged type (metaclass) can provide an excellent model of a physical entity. Perhaps the greatest fallacy of conventional object-oriented programming is the lack of appreciation of the power of generics, which both C# and Java lack. Table 2 includes "Bindings to graphical user interfaces and generators available (Fresco, etc)." No mention was made of the World Wide Web and XML. XML will be the major tool for human interfaces (screens and print). Fortunately, Ada is an excellent match for XML.

I do not fault the authors for the use of a Delphi exercise. I suspect that this was all they could do. However, as a taxpayer, I am appalled at the apparent lack of statistical process control data, which is fundamental to good manufacturing practices. A programming language is a tool used to manufacture software. The only reference was for a 1995 snapshot of Reifer Consultants Inc.'s databases published in 1996. Therefore, it does not contain any data on Ada 95. Since C++ has proliferated at the DoD, there should be a good opportunity to compare Ada, C++, and perhaps Java in terms of both development and maintenance costs. The DoD should have an ongoing system to monitor this data.

The "Colleges in the United States Teaching Ada" graph is a compelling example of the lack of DoD's ability to do technology transfer. Many of us would prefer to have reliable, efficient commercial off-the-shelf programs. It is also a good explanation of why the U.S. commercial software development process is so inefficient that we are compelled to import a huge number of programmers. It should be noted that, "By the National Science Foundation Act of 1950 the Congress established the National Science Foundation (NSF) to promote the progress of science; to advance the national health, prosperity, and welfare; to secure the national defense; and for other purposes [2]." Unfortunately, there appears to have been a lack of communication between the DoD and NSF on Ada and Software Engineering.

– Robert C. Leif, Ph.D., Newport Instruments (Ada_Med Division)

1. http://msdn.microsoft.com/msdnmag/issues/0900/csharp/csharp.asp
2. http://www.nsf.gov/od/lpa/nsf50/history.htm

SPMN Director Identifies 16 Critical Software Practices

Michael W. Evans
Integrated Computer Engineering Inc.

There are two distinct stages in improving a software process. The first stage is defining the optimum processes that can be applied successfully across multiple projects; to this end, tactical software processes have been developed. The second stage is embedding these processes in the project cultures that must apply them.

Author and Atlantic Systems Guild Principal Tom DeMarco makes an important observation in his book *Why Does Software Cost So Much* [1]. Instead of asking, "Why does software cost so much?" he says that we need to begin asking, "What have we done to make it possible for today's software to cost so little?" Like the question, the response seems to be "very little."

We as an industry seem to search in vain for the magic silver bullet—the right combination of methods and tools that would make predictable software cost, schedule, and quality performance a reality across the industry without a significant cost or schedule expenditure. Program managers, company presidents, controllers, and customers are continually surprised and disappointed when the magic does not appear.

Norm Brown, director of the U.S. Navy's Software Program Managers Network (SPMN), has recognized that, at least in the Department of Defense (DoD) and probably in other segments of the economy, the silver bullet is not now, or never will be a reality. Together with software industry leaders who meet regularly as part of the Airlie Software Council of 13, he has documented first a set of nine best practices, and now 16 Critical Software Practices for Performance-Based Management™, which appear to be common threads running through successful software projects. These 16 practices provide managers with specific methods that directly affect the bottom-line cost, schedule, quality, and user satisfaction metrics. Without exception, these practices may be implemented in less than a year.

Developers realize that implementation of the 16 Critical Software Practices requires the adaptation of each individual practice to each project, as well as cultural acceptance by the team who will do the work. As Tim Lister has pointed out, "Common processes represent 10 percent of the problem, adaptation is 90 percent [2]." The complexity of successful adaptation is as much due to cultural insertion as it is to tailoring to project needs and realities.

This paper discusses issues addressed by the 16 Critical Software Practices for Performance-Based Management, the rationale behind some key practices, and what can be expected when bringing the practices into a company culture or project environment.

The Basic Problem

As R.A. Radice and R. Philips point out in *Software Engineering: An Industrial Approach*, "We are still at the beginnings of becoming a science. We call ourselves computer scientists or software engineers, but it is more out of anticipation of what these roles offer than from a fully earned position. We, as an industry, still do not keep, analyze, or make public the necessary data to substantially prove our theories or to enable others to repeat our successes … We still cannot do as good a job on a new project as we did on the last [3]."

A reality since 1943, the software profession has had more than half a century to mature. However, since 1975, when *The Mudd Report* [4] was published, the DoD software community has been in a state of almost continuous crisis. Its attempts—without apparent result—to resolve crises through new technology, management practices, tools, or other actions have, at best, not improved the problems and have, at worst, compounded the effects. Reality is that in 1995, an estimated 53 percent of software projects cost nearly 190 percent of their original estimates, 31 percent of software projects were canceled before completion, and an estimated $81 billion was spent for canceled software projects by American companies and government agencies [5]. Apparently, we do not practice effective crisis management.

In an attempt to resolve the software crisis, DoD established the Software Engineering Institute (SEI) in 1984 as a federally funded research and development center at Carnegie Mellon University. SEI's mission is to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. The SEI vision is to bring the engineering discipline to software development and maintenance.

The Strategic Improvement Model

A key component of the SEI strategy is to establish a process to facilitate continuous process improvement within the software community. In SEI's view, continuous process improvement is based on many small, evolutionary steps rather than evolutionary innovations. The Capability Maturity Model® (CMM) provides a framework for organizing these evolutionary steps into five maturity levels that lay in order successive foundations for continuous process improvement [6].

In my view, the CMM has caused software providers to refocus on the benefits of, and the essential steps required for, improving software-engineering processes. Advancing up the CMM ladder is an essential strategy that many organizations follow to improve their software processes and enhance their image as serious software engineering contractors. CMM improvement requires significant time (12 to 18 months) and a significant commitment of resources to move up one level. Significant improvements in bottom-line metrics can be expected as new levels are achieved in projects where improvements have been applied (this does not include all organizations within the company).

However, this leads us to a dilemma. How do we guard against cost and schedule impacts, quality shortfalls, and user dissatisfaction with software systems produced or maintained by non-assessed projects or organizations? And how do we address the many projects and organizations that are just starting their climb up the CMM ladder? Current data show that for a total of 901 organizations assessed for SEI CMM implementation, 34.9 percent are at Level 1, 38.2 percent at Level 2, 18.5 percent at Level 3, 5.5 percent at Level 4 and 2.9 percent at Level 5.

A second problem in using CMM solely as a means to improve project performance is that the interpretation of the CMM focus is strategic in nature. It does not implement the tactical guidance required by individual projects to improve their performance to achieve immediate goals. Achievement of immediate goals may mean the difference between project continuity and project extinction.

Prior to further discussion on strategic versus tactical project implementation, it is important to define the two terms. Strategic processes are those of importance to the integrated whole or to the overall planned effect. While tactical processes are smaller in scale, serve the larger purpose when performed collectively, and are carried out with an immediate end in sight.

It is clear that the application and harmonious coexistence of both strategic and tactical processes will achieve overall project success.

CMM's five levels are a model for improving the software organization's capability. The CMM priorities, as expressed by these levels, do not address the root cause of problems faced by individual projects, nor do they point to remedies that are under the direct control of the project leader.

A troubled project's solutions might be of limited value to the rest of the organization. Other projects might have different problems or be unable to take advantage of its solutions because they lack the necessary foundation to implement the solutions [7].

There are two distinct stages in improving a software process. The first stage is defining the optimum processes that can be applied successfully across multiple projects. The second stage is embedding the process in the project cultures that must apply it. The process identification approach is fun and rewarding, while achieving cultural acceptance is tedious, frustrating, and difficult to accomplish. The truth is that if these improved common processes do not find their way into the project culture, project improvement of bottom-line metrics becomes a myth.

For example, a recently visited organization spent significant effort achieving CMM Level 3. They documented common processes, modifying them as necessary, and formed a very active software engineering process group (SEPG) composed of the "best software people in the company." The most frequent comment from projects was that the SEPG lived in an ivory tower. Management discounted the comment as *sour grapes* by a group of engineers who did not understand the real need to improve process.

As this organization moved up the CMM ladder, it became increasingly obvious that the expected improvements in development cost, schedule, product quality, and relationships with various user communities were not being realized and were, in some cases, getting worse. What was going on?

As the SEPG tried to force change on ongoing software projects, they were being *gamed*. The projects declared they had implemented the required key process areas (KPAs) but had segregated the project into two distinct teams—development and systems. The development team included all the software-related activities prior to delivery to the systems team, which integrated the products and delivered them to the customers. To minimize impact on the development culture, the KPAs were only applied to the products when they were released to the system team. Company management felt that directly impacting the development teams incurred too much risk.

I wish this example were the exception, but in my experience it is not. Too many organizations focus on the obvious recognition and marketing advantage that a higher CMM rating confers, without recognizing that the reason for pursuing continuous improvement is to improve bottom-line metrics, thereby improving the quality of products delivered within planned timeframes and within budget.

The strategic process role is to establish an environment within an organization that actively promotes, finances, and supports applying improved best practice initiatives. The strategic process should not attempt to precisely detail what steps need to be taken at any instant in time by each respective member of the project team. The strategic process should be tailor-made to specific project scenarios. It should give latitude to the project team to apply applicable solutions to specific project problems. In other words, the strategic process must allow the project team tactical freedom to achieve project objectives.

The Tactical Improvement Model

Brown and the SPMN recognized the need to develop a tactical model for software process improvement. SPMN is a grassroots organization of software managers formed by Brown in 1992 to share lessons learned and improve the bottom-line metrics of cost, schedule, quality, and user satisfaction on software projects.

Far too many large-scale software projects have become unaffordable and unable to deliver needed quality, reliability, and capability within the required time frame. Their outputs are not predictable. Their processes are little more than chaotic and do not effectively utilize the kinds of disciplines necessary to achieve success. They have not yet taken advantage of the types of practices used to effectively manage large-scale hardware projects [8].

From its inception, SPMN has focused on tactical issues and practical solutions that have been proven in industry and that focus on project rather than organizational issues. As part of its charter, SPMN has been a major identifier of software acquisition best practices both within and outside the DoD.

The [1995] Software Acquisition Best Practices Initiative [performed by the SPMN] was established to bring about substantial improvements in productivity, quality, timeliness, and user satisfaction by implementing best practices as a new foundation for DoD software management. Two purposes of the initiative include focusing the defense acquisition community on employing effective high-leverage software acquisition management practices, and enabling managers to exercise flexibility in implementing practices within disparate program cultures. The initiative is intended to influence both government software program managers and their industry counterparts.

Solutions are taken from successful programs' practices. When effectively implemented and given competent staff these practices help bring order, predictability, and higher levels of productivity and quality. Each one includes key applicability factors enabling adaptation to particular situations and environments.

These practices are focused upon effective management processes, techniques for finding defects as they occur, eliminating excessive and unnecessary costs, increasing productivity, and other beneficial effects. The Airlie Software Council and other industry experts and consultants are convinced that projects effectively utilizing the Software Acquisition Best Practices and other appropriate best practices will achieve significant cost reductions while simultaneously increasing quality and reliability [9].

Originally SPMN defined nine practices as being essential elements of a successful project. The original nine practices were applicable to all large-scale projects (i.e., projects relying on the full-time efforts of 12 or more people annually). Observably best practices should appropriately be used according to the particular circumstances and environment of a given project. From the outset, SPMN recognized that the practices, as with any best practice, are of little value unless they are adapted to specific project environments and cultures and are embraced by those who must implement them.

The software development process consists of many components that must fit together to create a total and integrated project environment. These individual project pieces must interface and interact efficiently within other project segments if the project is to function efficiently [10].

Concern about internal project consistency and practicality of recommended practices in light of project realities and constraints has always been a SPMN concern when recommending practices. SPMN practices are termed *best,* not because they have been intensively studied and analytically proven to be best, but simply because they are practices used by and considered critical to successful software projects. SPMN does not feel that the original list of nine and the current set of 16 practices are presumed to be best; nor that there may not be other, perhaps even better, practices. The practices that comprise the 16 critical software practices, when implemented in projects, will go far toward engendering successful software development and maintenance.

Before engineers can work effectively in an integrated team environment, they need to know precisely what to do. Teams can waste a great deal of time trying to establish goals, resolve their working relationships, and determine what to do [11]. SPMN's 16 critical software practices provide specific guidance as to which practices have been proven to work on similar projects, and how they can be implemented within a specific project.

As illustrated in Figure 1, the 16 critical software practices address three primary areas of software management: project integrity, construction integrity, and product stability and integrity. Project integrity includes practices that identify basic project constraints, requirements, and expectations. It also encompasses planning and implementing practices for a project environment to predictably satisfy them. Construction integrity comprises those activities that specify the basic product requirements, maintain traceability to these basic requirements, plan and control content and change, and ensure that all components of the project communicate. Product stability and integrity ensures that defects, which are inserted in products as part of the software process, are identified and removed in a timely fashion. It also ensures that testing is complete and effective and results in the right product consistent with the agreed-to requirements and actual expectations.

Although these 16 practices are useful individually, their complementary nature provides a strong synergistic effect when used as an integrated set. Using them will not guarantee success, but they can help facilitate it and avoid failure. Those familiar with process improvement models such as CMM will quickly realize that these practices supply tactical solutions that compliment the model's strategic orientation. The practices map to many of the model's KPAs, and should assist organizations striving to advance to the next CMM maturity level.

SPMN has developed a software evaluation model (SEM) that is based on the 16 critical software practices [12]. The SEM provides for each of the critical practices, practice essential elements, implementation guidelines, and a detailed question set to assist in implementation. In addition to the SEM, a mapping matrix is available that maps each of the critical practices with SEI's CMM Level 2 and 3 activities.

These practices are not rocket science. They can be readily implemented. Although some practices may require training in basic skills such as conducting effective meetings as a necessary foundation for formal inspections, for the most part they can be implemented without making investments in new equipment, technologies, or staff.

Figure 1. *16 Critical Software Practices for Performance-Based Management*

Project Integrity

- Adopt continuous risk management.
- Estimate cost and schedule empirically.
- Use metrics to manage.
- Track earned value.
- Track defects against quality targets.
- Treat people as the most important resource.

Construction Integrity

- Adopt life cycle configuration management.
- Manage and trace requirements.
- Use system-based software design.
- Ensure data and database interoperability.
- Define and control interfaces.
- Design twice, code once.
- Assess reuse risks and costs.

Product Stability, Integrity

- Inspect requirements and design.
- Manage testing as a continuous process.
- Compile and smoke test frequently.

Successful Implementation

More than 200 risk assessments conducted during the past five years along with effective risk mitigation advice to software developers and government software program management offices has proven that the 16 critical software practices are a successful tactical software project process

Typical of the success stories is one in-process program that suffered from being over budget, late in delivery, and producing product that failed to meet specified requirements. This program's management recognized that it was in crisis when a major system it was preparing for delivery failed to meet its operational evaluation. The program manger initiated a problem analysis that identified three contributing factors. First, the program team was continually reacting to crises and never had time to plan ahead and anticipate problems. Second, there was no focus on success or failure accountability in the program office, nor did this accountability flow down to the contract suppliers. Third, there was no process in place that would ensure that a quality software product was fielded.

To address crisis management and effectual program control, an effective risk management process was implemented. To anticipate actions to be taken in managing the program, the program office utilized risk management process and risk management tools. This was not easy culturally because risk prediction was associated with bad news and failure. In addition contractors had signed up for unrealistic tasks, milestones, technical commitments, and delivery agreements that rightfully pointed to gloomy risk assessment outcomes. Crisis management rather than effective risk management occurred because of the significant overcommitment of resources.

All project and supplier management staff were provided with risk management training, while the program office was given expert assistance to help develop and implement risk policies and plans. The risk training program focused on both procedural risk issues and problems with cultural acceptance of the risk process within the program. To track individual risks, risk management experts assisted program staff in identifying risks and seeding them into an appropriate risk management tool. Risk experts assisted the program staff in identifying what could go wrong and mapping the road ahead.

Risks were assessed on the basis of their impact and likelihood of occurrence. Risks with high impact and high likelihood of occurrence were given the highest level of scrutiny and attention. A monthly risk reporting process was implemented that initiated corrective action taken by the program manager. After implementing the risk management process, the program took an about turn. It shifted from unsuccessful crises management to a process that identified potential problems early and permitted effective corrective action well before disastrous or costly failures had occurred.

Setting and enforcing clear goals addressed the second problem of accountability. Goals included setting quantitative targets that could be measured. Progress reports that described the progress toward assigned goals were required on a regular basis. Five targets were established:

- Bring the reliability of all developed systems to a specified number of hours within a 12-month period.
- Deliver all future products within cost and schedule.
- Have all contractors' risk management process in place and compliant with the risk management plan within six months.
- Deliver software that is supported by adequate documentation.
- Deliver software that meets user requirements.

Finally, to address software process development standardization, training was provided to program staff on the 16 critical software practices. A practice assessment of all the program suppliers was performed, and each supplier was evaluated by practice area and rated on a five-point scale (one being the practice was nonexistent and five being the procedure was in place in the culture of the organization).

The first assessment scores averaged 1.8. By the third assessment the average score was greater than four. The established process improvement was evident in the measured metrics of cost, schedule, quality, and user satisfaction. The program made great strides in software process improvement during a one-year period. The program manager rated the cost of installing the improved process as extremely cost effective (approximately 1 percent to 2 percent of the program cost).

Program suppliers measured success not only by product quality, but also by adherence to set program goals and their degree of implementing the 16 critical software practices. The program progressed from having consistent software acquisition problems to a competent software acquisition organization with a reputation for delivering quality product, on time, and within budget.u

References

1. DeMarco, Tom, *Why Does Software Cost So Much?* New York: Dorsett House Publishing Co., 1995.
2. Lister, Timothy, Software Management for Adults, Salt Lake City: Software Technology Conference, 1996.
3. Radice, R.A. and Philips R., *Software Engineering, An Industrial Approach*; Prentice Hall, 1988, pp. 2-3.
4. Weiss, David, *The Mudd Report: A Case Study of Navy Software Development Practices,* Washington, D.C.: Naval Research Laboratory, May 21, 1975.
5. Standish Group International, Chaos, *Open Computing*, March 1995.
6. Paulk, Mark C. et al., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Version 1.1, Reading, Mass., Addison-Wesley, pp. 15-17.
7. Ibid.
8. Software Program Managers Network, *The Program Manager's Guide to Software Acquisition Best Practices,* V. 2.2, Arlington, Va., SPMN, 1998, IV.
9. Ibid.
10. Evans, Michael W. and Marciniak, J., *Software Quality: Management and Assurance*, New York: John Wiley & Sons, 1987, p. 20.
11. Webb, David and Humphrey, Watts, Using the TSP on the TASKVIEW Project, CROSSTALK, February 1999.
12. SPMN, Software Evaluation Model (SEM), version 5.3.1, May 2000 [www.spmn.com].

About the Author

Michael W. Evans is president of Integrated Computer Engineering Inc. He is experienced in providing direct technical services and support in software engineering methods and processes, software standards, quality assurance, configuration management, and testing. He is cofounder and prime contractor for the Software Program Managers Network, the driving force behind the Department of Defense's Software Acquisition Best Practices Initiative. Evans is the author of *Principles of Productive Software Management, Productive Test Management, Software Quality Assurance and Management*, and *The Software Factory*.

ICE Inc.
142 North Central Avenue
Campbell, Calif. 95008
E-mail: candca@aol.com
Internet: www.iceincUSA.com

How Process People View Life

"You want Daddy to read you a bedtime story? How about *Goldilocks and the Three Bears?"*

Once upon a time [What accuracy of time? How many digits of precision are needed? Is real-time really necessary here?] there was a little girl named Goldilocks [Why Gold? Did the customers request a specific color this early in the design phase?] who packed a picnic basket to go visit her grandmother. [What are the grandmothers' dietary requirements? What are her nutritional needs? Are there any food allergies?] Goldilocks set off through the forest to grandma's house [Why did she set off so quickly? What life-cycle model is she using that suggested implementation so early? Did she have a prior visit to base her projected arrival date upon?] to deliver the basket. [Was Goldilocks goal-oriented and quality-driven? There seems to be no evidence of an enabling infrastructure to help her self-actualize and fulfill her needs. Does Goldilocks need a nurturing work environment to be truly productive?]

At the same time, there lived three bears in the forest. [Same time? Is this a potential concurrent-processing problem? Are we going to have to worry about parallelism? Does our design methodology support real-time interfacing? We should probably use Ada for implementation.] The three bears were daddy bear, mommy bear, and little baby bear. [Is this a well designed, object-oriented system? Is inheritance correctly used?] Mommy bear had just prepared porridge [Was this on her list of deliverables for the current milestone?] and the family sat down to eat. Unfortunately the porridge was too hot, so they decided to take a walk to let it cool. [Is this interruption necessary? Once the team loses focus, it is difficult to recapture a synergistic mindset. Couldn't other team goals be worked on to ensure the team stays cohesive? Why didn't mommy bear perform a personal review prior to conducting a peer review?]

While the bears were out walking, Goldilocks wandered upon the house and, lost and hungry, smelled breakfast and decided to eat. [What kinds of protection scheme for critical resources are in place? Doesn't Goldilocks have both milestones and inch-pebble deliverables to keep her on track?] The first bowl of porridge was too hot, the second too cold, and the third was just right. [Good fence-post testing techniques. However, shouldn't independent verification and validation be contracted to ensure that her testing parameters were realistic and customer-oriented?] After eating, she went upstairs for a nap. [Good idea. Studies show that neural activity begins to decline after sustained exertion. Rest breaks promote quality and reduce rework.] Upstairs, Goldilocks found three beds. The first bed was too hard, the second too soft, and the third was just right. [Is she qualified to test both porridge and beds? Seems like a reusable test case, but is it correctly tailored and parameterized for beds? A dedicated test team might be needed, with both porridge and bed domain experts available] She laid down to rest.

While she was sleeping, the bears returned. Daddy bear found his porridge, mommy bear found her porridge, but baby bear discovered that his porridge was all gone! [Poor allocation of resources. Is the work breakdown structure set up to handle dynamic reallocation so that baby bear will not be the critical path?] Suspecting something was wrong, all three bears went upstairs. Daddy bear found his bed disturbed, as did mommy bear. Baby bear, however, found someone sleeping in his bed. [This is the second time baby bear has found serious defects in his personal deliverables that effect other team members' schedules. If this trend continues, we might have to assign baby bear some additional quality training. In the worst case, we might have to bring in a better team player.]

Hearing the commotion, Goldilocks awoke with a start, and ran down the stairs, out the door, and straight to her grandma's house. [When the bears saw the problem, did they take proactive steps to prevent a future occurrence? Are they using root-cause analysis to fix the problem, rather than just fixing the symptom? More importantly, if Goldilocks knew the way under pressure, was she just slacking off early in the project? Were sufficient key process areas in place to ensure her original schedule was realistic? Was she revising her schedule during each iteration of the life-cycle model? She wasn't self-assessed, was she?]

Along the way to her grandma's house, Goldilocks ran into the big bad wolf. The wolf convinced Goldilocks to accept work for a competing grandmother, at a higher salary with better benefits. Unfortunately, the basket to the original grandmother was delivered over-budget and behind schedule. Luckily, a source of continuing funding exists, so errors are still being fixed in Grandma's Basket release version 4.3a. [Gosh, I just LOVE a happy ending!]

"What, honey? What do you mean, you want Mother to read to you from now on?

– David Cook, Shim Enterprise Inc.

ATTENTION AIR FORCE SOFTWARE PROGRAM MANAGERS!

Wrestling with legacy code?
Slaved to proprietary hardware?
Having difficulty hiring the talent you need?

MISSION CRITICAL SOFTWARE
CRSIP

The Air Force Computer Resources Support Improvement Program (CRSIP) is listening!

CRSIP invites your input to the Air Force's Software Technology Investment Strategy and your participation in a roundtable and panel discussion of software needs (technology, acquisition management, and process improvement) on Tuesday, May 1 from 12:00-1:00 p.m. during the 2001 Software Technology Conference in Salt Lake City, Utah. The complete conference schedule will be posted soon at www.stc-online.org

STC 2001 SM

To gain wider input to Air Force requirements for software improvement projects, CRSIP is interviewing engineers and program managers of software intensive programs for inputs. The results of the needs analysis will be summarized at STC, and used to structure the Air Force's investment strategy for software improvement. If you would like your input to be a part of this effort, see http://crsip.hill.af.mil for a questionnaire and more information.

MISSION CRITICAL SOFTWARE
CRSIP

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

CrossTalk / TISE
5851 F Avenue
Building 849, Room B04
Hill AFB, UT 84056-5713

PRSRT STD
U.S. POSTAGE PAID
Kansas City, MO
Permit 34