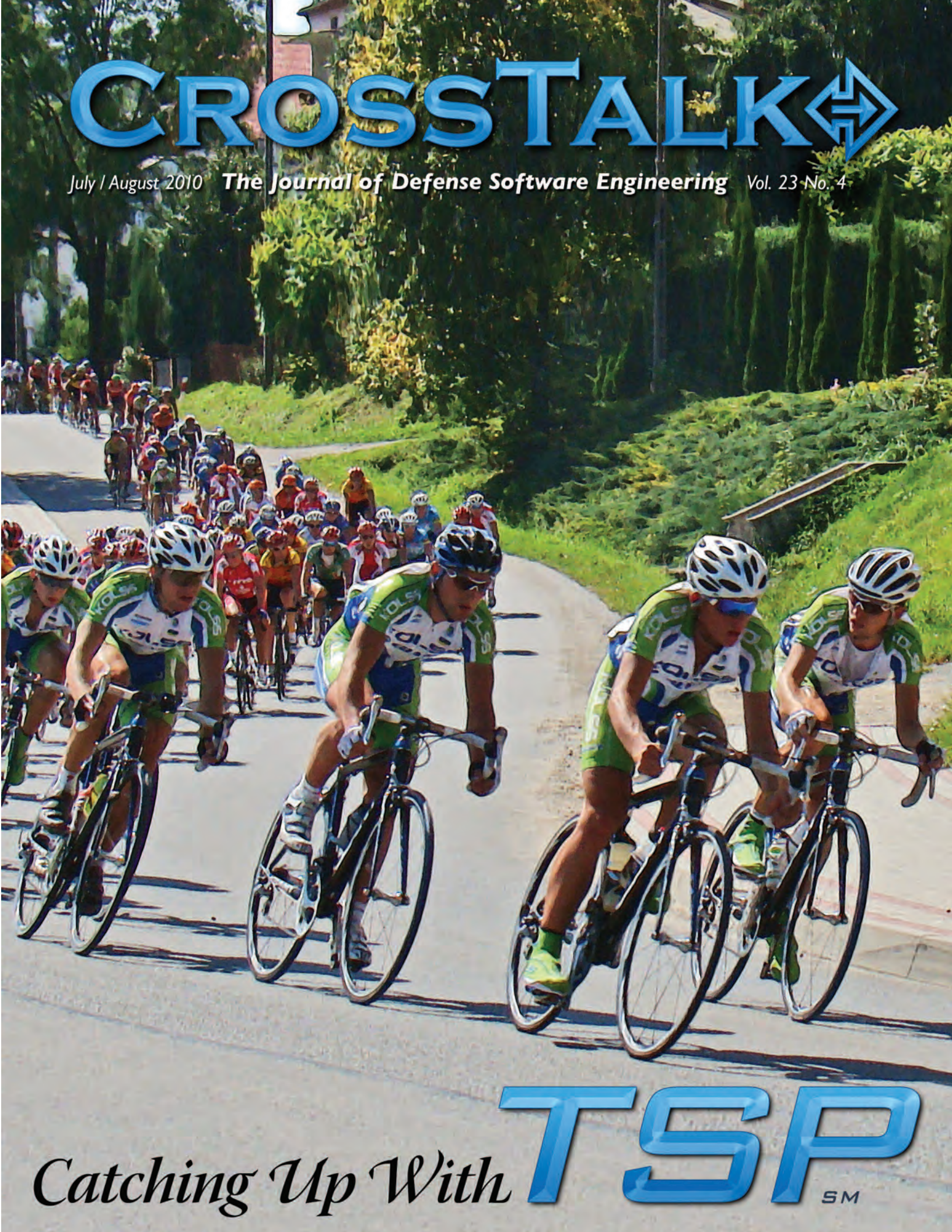


CrossTalk

July / August 2010 The Journal of Defense Software Engineering Vol. 23 No. 4

Catching Up With TSP℠

Catching Up With TSP

Open Forum

ON THE COVER
Cover Design by Kent Bingham
Photo by Silar

Departments

CROSSTALK

OSD (AT&L) *Stephen P. Welby*
NAVAIR *Jeff Schwalb*
309 SMXG *Karl Rogers*
DHS *Joe Jarzombek*

MANAGING DIRECTOR *Brent Baxter*
PUBLISHER *Kasey Thompson*
MANAGING EDITOR *Drew Brown*
ASSOCIATE EDITOR *Chelene Fortier-Lozancich*
ARTICLE COORDINATOR *Marek Steed*
PHONE (801) 775-5555
E-MAIL stsc.customerservice@hill.af.mil
CROSSTALK ONLINE www.stsc.hill.af.mil/crosstalk

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Office of the Secretary of Defense (OSD) Acquisition, Technology and Logistics (AT&L); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). OSD (AT&L) co-sponsor: Software Engineering and System Assurance. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cybersecurity Division in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal. CROSSTALK's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 21.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the author and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CROSSTALK *Online Services:* See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

TSP: Tailor ... Learn ... Grow ... Apply ... and, of Course, Repeat

NAVAIR

CrossTalk would like to thank NAVAIR for sponsoring this issue.

If you've been involved with software and system process improvement for even a short time, you've most likely experienced the challenges associated with applying process methodologies and tools to your real-life projects. For many, the realization that "one size does not fit all" can lead to frustration about how best to tailor the processes and tools to fit real-life project needs. TSP is one process framework and toolkit helping teams improve software quality and productivity. Thoughtfully marrying TSP application with your unique team, products, and goals can put you on the path to meeting software cost and schedule commitments.

In my experience, the most valuable leg up for adopting TSP are TSP coaches, who focus on supporting the team (as well as individuals) to transition from workshop learning to practical application. They play a huge role in motivating and guiding the team through their TSP journey. Given the coaches' first-hand TSP experience—and their in-depth knowledge and appreciation of the toolkit—they lend a supportive hand as the team tailors, monitors, learns, and grows.

An example of TSP tailoring that can have powerful results is modifying role definitions. While TSP does define specific and meaningful roles, the assumption is that these roles can (and may need to be) thoughtfully tailored. In considering how to apply the roles to your project, it is best to evaluate each role in the context of your team's culture, size, and dynamics. Also, make an effort to align teammates to the roles based on the expectations for a specific role and their unique capabilities. Just going through this effort to align roles with your team's context and characteristics can lead to unexpected insights and learning.

Standard TSP application assumes that you are tracking a single product from start to finish. Since this is not always the case, think carefully about how best to apply the processes when multiple efforts need to be completed in unison.

TSP offers a useful and free tool to gather and report metrics. This tool is most valuable when you take the time to understand how the metrics will be used in your larger project context. Based on your experiences, you may even be able to offer insights into how to make the tool more useful. For instance, based on user feedback that indicated a need for milestones to support parallel task execution, the tool now offers a single target data feature. With this feature, progress toward incremental milestones can be evaluated and understood.

While TSP is for software, it provides a construct for detailed planning and task allocation to any engineering effort or product. Basically, any task or group of tasks that can be broken down into increments, activities, goals, and timelines can benefit from applying TSP. Again, it's a matter of understanding your particular requirements and context, and determining how best to integrate TSP capabilities.

And finally, to support your endeavors are the annual TSP user conferences, with the next gathering in Pittsburgh September 20-23 (see <www.sei.cmu.edu/tspsymposium/2010>). These get-togethers provide a forum for open and honest dialogue about the "goods, bads, and others" related to teams' efforts to adopt TSP. These symposiums reinforce the culture and context you would expect to find in any authentic improvement and learning effort.

So the bottom line is that to be successful, no matter what approach you choose, means that you have to take the long-haul perspective and tailor, learn, grow, apply, and repeat as needed. Of course, it goes without saying that you will also need to factor in a healthy dose of relentless patience.

Susan G. Raglin
Head, Software/Systems Product Development & Integration Division
Naval Air Systems Command

Why Can't We Manage Large Projects?

Watts S. Humphrey
Software Engineering Institute

Changing managers, procurement regulations, acquisition procedures, or contracting provisions have not resolved the cost and schedule problems of large-scale system development. This article shows the problems that organizations face with large system projects—and how one government organization has succeeded, over a period of several years, using the Team Software Process (TSP[SM]).

The Naval Oceanographic Office (NAVO) Systems Integration Division began working with the SEI 15 years ago. Their group produces software for a range of systems that supply oceanographic and meteorological data to the U.S. Navy's worldwide fleet. These are enormous terabyte systems that operate 24-7, and their subsystems provide critical operational information to almost every branch of the Navy.

Ed Battle—branch head then, and now Systems Integration Division director—recalled that when they started working with the SEI, projects were always late, requirements were frequently misunderstood or wrong, and there was no cooperation among the many interdependent groups. When critical delivery dates approached, the director tracked the work with regular Monday, Wednesday, and Friday status meetings. While these meetings raised the pressure and took a lot of time, they didn't shed much light on project status.

Battle's question to us at the SEI was: "Isn't there a better way?"

The Large System Problem

The problems Battle's group faced are typical. Large system projects fail all the time and the larger they are, the more likely they are to fail. For example, the new IRS system was five years late when it was first used in 2005, but its costs had exploded to $2 billion. A recent Government Accountability Office defense acquisition assessment of 72 typical weapons programs found that the development costs had climbed 40 percent from the first estimates, there was an average delay of 21 months, and the total systems overrun was $2 billion [1].

The situation is even worse for truly massive systems programs, as the New York Times also recently reported: Two-thirds of the largest weapons systems ran over their budgets last year, for a combined extra cost of $296 billion [2]. These programs were, on average, almost two years behind schedule.

Problem Causes

Studies show that these development problems are typically not caused by technology issues but are largely due to program management [3]. Unfortunately, the common reaction to program management problems is to replace the program managers. This blame-based culture stifles communication and fosters an opaque and defensive management style. We have been changing managers for years, but it should now be obvious that the problem isn't bad managers: They are good people put in untenable positions.

> *"We have been changing managers for years, but it should now be obvious that the problem isn't bad managers: They are good people put in untenable positions."*

For example, the replacement FBI system was recently killed when it fell three years behind schedule and after the project had spent $150 million. The program had a total of five CIOs and nine program managers. Clearly, changing managers did not fix the FBI's problems. But neither did changing acquisition systems, reorganizing the Pentagon, or modifying procedures. Projects keep failing. In fact, more and more large projects fail these days than in the past—and the failures are even more expensive and painful.

The common view is that the program manager is responsible for doing whatever is required to get the job done. If new management or technical methods were needed, he or she should put them in place or take whatever steps were needed to do so. But the fact that these large projects keep failing suggests that program managers don't know what to do. However, we must do something and it should by now be clear that relying on program managers to fix these projects isn't working. This article suggests how to address these problems in a way that program managers can implement today.

Knowledge Work

We explained to the NAVO that the problems with software work were an early indicator of the problems that would soon plague all aspects of modern engineering work. Software has been hard to manage since the beginning, but the reason has nothing to do with the technology. The reason is that software is a different kind of work.

For the more traditional work of the past, the managers could walk around the lab or plant and see what was going on. This is called management by walking around (MBWA), a very effective way to keep management informed about the work and for keeping the workers on their toes. However, the principal problem with MBWA is that it is only effective for work that one can understand by watching the workers do it. Today, most sophisticated technical work is more like software: A great deal of the creative effort is done on a computer or in a worker's head, and results are largely invisible to the casual observer. Peter Drucker, the first to describe knowledge work, said that it is work with the mind rather than with the hands [4]. The products, instead of being things you can touch and feel, are ideas. While these ideas may ultimately be embodied in physical products, the bulk of the work, and the true product value, is in the creative effort required to develop

[SM] Team Software Process and TSP are service marks of Carnegie Mellon University.

these ideas and transform them into marketable products.

Traditional Management

Even though the workers and much of their work is vastly different from 100 years ago, today's traditional management methods are still based largely on the principles from Fredrick Winslow Taylor's 1911 book, "The Principles of Scientific Management" [5]. Taylor's methods were designed for uneducated workers and the relatively simple manual tasks of the past. The kind of work and the skills and methods involved in much of today's work are quite different, but today's management methods still follow Taylor's command and control principles. Unfortunately, with software and most other sophisticated technical work, these methods are not effective in controlling project costs, schedules, or quality. While the managers may try valiantly to manage the work, they cannot know what the knowledge workers are doing or how they are doing it.

The end result is that today's managers cannot truly manage their knowledge-working projects. That means that these projects are not being managed, and everybody knows that unmanaged projects usually fail. Unfortunately, the managers are generally blamed for the failures when the real problem is with the management system—and not the managers. The answer is not to replace potentially very capable managers, but to change the management methods. Program managers, however, typically do not know what changes to make and are understandably reluctant to change to a new management method that is not in general use by other similar programs.

Managing Knowledge Work

In considering how to manage knowledge work, Drucker concluded that since managers cannot truly manage such work, the knowledge workers must manage themselves. While many managers say that they already involve their people in their own management, involvement is quite different from responsibility. To truly manage themselves, the knowledge workers must be trained in personal and team management methods and they must be held responsible for producing their own plans, negotiating their own commitments, and meeting these commitments with quality products. The manager's job is no longer to manage the knowledge-working teams but to lead, motivate, support, and coach them.

Software teams like to work this way. Where once they struggled to meet management's schedule targets, they now negotiate their own commitments with management. The teams feel personally responsible for and in control of their work, they know project status, and they have the data to defend their estimates. When they see problems, they resolve them or get management's help. Furthermore, when the knowledge workers measure, track, and report on their work, the managers have the data to help them resolve problems. Then the entire management system can participate in making their programs successful.

When knowledge-working teams have appropriate management, training, and support, they can work in this way (see the sidebar for the principles of knowledge management). Then they consistently meet their cost and schedule commitments with high-quality products. What's more, these identical knowledge-working principles can be applied to all of the engineering projects in an organization, producing a measurable and trackable knowledge-work management process across a large program or even an entire organization.

Workplace Objectives

One of the more fundamental problems with current management practices is that the workers and managers have different views of project success. Studies show that product developers view a project as successful if the work was technically interesting and they worked on a cohesive and supportive team [6]. This was true whether or not the project met its cost or schedule objectives. Conversely, the managers viewed projects as successful if they met their cost and schedule targets with little regard for the nature of the technical work or the working team environment. This difference in workplace objectives has a profound effect on program management. For example, when the program manager wants to know when some large program will finish, he or she asks the project leaders. They then talk to their team members. The team members view the schedule as management's problem, however, and give vague answers such as "I'm almost through the design," or "Just a couple more bugs and I'll finish testing." While the knowledge workers are typically the first to sense that a project is in schedule trouble, they have no way to precisely describe job status. Rather than say something and risk getting involved in a lot of management debates, knowledge workers would rather concentrate on their technical work and leave the schedule problems for their managers.

The Surprise Problem

Fred Brooks once said, "Projects slip a day at a time" [7]. To keep their projects on schedule, all that managers have to do is make sure that their teams recover from these one-day slips every day. With large-scale knowledge work, however, the managers can't see these small daily problems and the developers don't have the data to describe them. As a result, the managers can't take action to recover from the one-day slips. By the time the schedule slips are large enough to be visible, it is too late to do anything about them. This is why projects that are run by very capable and experienced managers keep having cost, schedule, and quality problems. The managers don't have the feedback they need to see problems in time to prevent them. It is as if they were driving a car at a high speed in a dense fog. Once they see a problem, it is right in front of them, and they must make a panicked effort to avoid a crash. Today, in large systems projects, the managers are driving fast in a fog—and crashes happen all the time.

By the time more senior managers see these project crashes, the schedule delays

Management Principles for Knowledge Work

The management principles for knowledge work are fundamentally different from those for traditional engineering. The five management principles for knowledge work—which were adopted from my forthcoming book "Leadership, Teamwork, and Trust: Building a Competitive Software Capability"—are as follows:

1. **Trust the knowledge workers.** Management must trust the knowledge workers and teams to manage themselves.
2. **Build trustworthy teams.** The knowledge-working teams must be trustworthy. That is, they must be willing and able to manage themselves.
3. **Rely on facts and data.** The management system must rely on facts and dates—rather than status and seniority—when making decisions.
4. **Manage quality.** Quality must be the organization's highest priority.
5. **Provide leadership.** Management must provide their knowledge workers with the leadership and support they need to manage themselves.

are typically quite significant. Furthermore, on a large project with many interdependencies, delays in any one part will affect many others. This means that many parts of a large program will probably get into schedule problems at about the same time. The managers of the many parts of the program then face a difficult choice: be the first to admit to schedule problems or wait for someone else to get into trouble first.

Blame-Based Management

Unfortunately, with the current system, senior leadership tends to blame the managers for management problems. By being the first to admit problems, the managers could easily be blamed for the entire program's problems. Not surprisingly, most managers decide to concentrate on the problems they can solve and wait for someone else to blow the whistle. By the time the problems are visible to senior leadership, the program is in such serious trouble that there is no chance to recover. Then everyone upstairs is surprised.

The combination of a *blame-based management system* and the lack of precise project status measures motivates both opaque management and a general reluctance to admit to problems. With large and complex systems programs, every part is important: Problems anywhere can delay everyone. That is why every component element of the work must be managed and tracked and why every team must strive to meet all of its commitments. That is also why, without precise status information, all estimates and commitments at the team level (and, for that matter every higher level) are just guesses. Finally, that is why, with today's typical management systems, large projects are almost always late and over budget.

The NAVO and the TSP

After we had reviewed these points with Battle and his associates, he agreed that it all sounded very reasonable—but wondered how it would help him and the other managers keep their large programs on schedule. We explained that the SEI had developed a knowledge-working process called the TSP, and that one of its principal features was that its management system was based on precise, operational-level data [8]. With the TSP, the developers gather and use data to manage their own work, and they use their data to accurately measure project status to within fractions of a day. TSP teams report their status to management every week, and management can see exactly where every element of every project stands. With precise status information, management can see small cost and schedule problems before they become serious. They can then take timely action to identify and resolve the problems.

When knowledge workers have been trained and know how to manage themselves, they have detailed plans and know project status precisely. They also feel responsible for managing their own problems and, when they need help, can call on their teammates or, if needed, on management. No process can eliminate problems; they are a natural consequence of doing large-scale complex work. But with sufficient warning, recovery actions are almost always possible—and most of the problems can be avoided or resolved without a crash. The key is early warning: That is why detailed plans, precise status measures, and working-level issue ownership are critical. For knowledge work, you will only get an early warning when the knowledge workers manage themselves.

"No process can eliminate problems ... But with sufficient warning, recovery actions are almost always possible—and most of the problems can be avoided or resolved without a crash."

However, just training workers how to manage themselves is not enough. Many of the problems with current engineering work are caused not by the workers and managers themselves, but because they do not properly use the knowledge they already have. To use what is learned, they must know what to do and how and when to do it. For large-scale projects, an operational process is essential. Program management is a matter of detail, and every step must be done precisely and correctly. Just like airline pilots when they do their final preflight checks, they follow a detailed checklist. While they know every step and have done it thousands of times, studies have shown that most airplane accidents involve at least one case of a skipped step or an improperly followed checklist. This focus on precise work is the role of an operational process: to ensure that every step is done precisely and correctly.

For many of the simple tasks that we do all the time, we know unconsciously what to do and how to do it. But for complex or new and unfamiliar tasks—such as personal planning, precise schedule management, and data-intensive quality management—the steps are not obvious. That means that merely training the knowledge workers in theoretical methods will not get them to use the methods correctly or consistently. For that, they must have an operational process with quality measures and trackable plans. But once knowledge workers are properly trained, know why and how to manage themselves, and have an operational process that they actually use, they can make and follow detailed plans and precisely track and report their progress against these plans.

The NAVO Experience

When the NAVO started working with the SEI, they originally used the Capability Maturity Model® (CMM®). It was helpful, but gave them the *what* when they needed help with the *how*—and it was difficult to implement. On the other hand, the NAVO found that the TSP was a better fit, with the guidance they needed to properly manage their projects. It also provided for rapid training (initial team-member training takes a week), with teams soon after launching the TSP and managing themselves.

Once the teams were using the TSP, the benefits of better planning, tracking, and reduced test time were immediately apparent. Many organizations even found that the savings from just the first project pay for that team's entire training and introduction costs. The team can then continue using it without any further training investment.

After using the TSP for several years, Battle reported that their product quality levels have improved by about 10 times and that testing times have been reduced from months to weeks. Schedule and cost performance is much more predictable than before, and the Monday, Wednesday, and Friday weekly status meetings are no longer needed. Team cooperation and coordination was also greatly improved. Battle's final conclusion was that, "This is the only way to manage large knowledge-working projects."

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

Conclusions

The consistent failure of large-scale development programs not only costs a lot of time and money, it delays the introduction of promising new technology and deprives our fighting forces of the tools they need to protect our nation. By now it should be obvious that the U.S. defense industry lacks the motivation to address this problem. For example, a mid-level executive of a major defense contractor recently told me that he could not afford to use high quality development methods like the TSP because it would reduce his revenue. His organization gets paid when they overrun projects and they get new contracts to fix their defective products. If this executive eliminated this source of revenue, he would lose his job. One could argue that the answer to this situation would be fixed-price contracts, but this approach has been tried several times in the last 50 years and has not solved the problem. It merely converts technical issues into contract disputes and the contractors get paid anyway.

Similarly, the program managers can't solve this problem. Even if they were familiar with the TSP and convinced that it would work, they would be reluctant to try something before it had been widely used by other programs or recommended by acquisition management. The TSP has a proven record of success and it could help to address this problem right now. The DoD—or some other government agency—should evaluate or test the TSP[1] and other promising methods to determine their suitability. It should then determine the best methods to use in managing these large programs and recommend that program managers require their contractors to use these methods. This should not be an expensive or time-consuming effort. Large-scale systems development is too critical a national problem to ignore—and the savings could be enormous.◆

Software Defense Application

There aren't many organizations bigger than the defense industry—and none with a bigger need for success in their large-scale development programs—where failure can have billion-dollar financial impacts and, worse yet, present dangerous security vulnerabilities. TSP creator Watts S. Humphrey, whose groundbreaking 2000 report outlining the TSP (see <www.sei.cmu.edu/reports/00tr023.pdf>) was sponsored by the DoD, feels that our defense industry can benefit significantly more from the process. Through past experiences, and the success of an organization providing oceanographic products and services to all DoD elements, Humphrey shows how and why the DoD needs the TSP now more than ever.

References

1. GAO. *Defense Acquisitions: Assessments of Selected Weapons Programs*. Report to Congressional Committees. GAO-08-467SP. Mar. 2008 <www.gao.gov/new.items/d08467sp.pdf>.
2. "A Lot More to Cut." Editorial. *New York Times*. 11 May 2009 <www.nytimes.com/2009/05/11/opinion/11mon1.html>.
3. Office of the Under Secretary of Defense. *Report of the Defense Science Board Task Force on Defense Software*. Nov. 2000 <www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA385923&Location=U2&doc=GetTRDoc.pdf>.
4. Drucker, Peter F. *Landmarks of Tomorrow*. New York: Harper & Row, 1957.
5. Taylor, Frederick Winslow. *The Principles of Scientific Management*. New York: Harper & Brothers, 1911.
6. Linberg, Kurt R. "Software Developer Perceptions about Software Project Failure: a Case Study." *The Journal of Systems and Software* 49 (1999): 177-192.
7. Brooks, Frederick P. *The Mythical Man Month: Essays on Software Engineering*. 20th Anniversary Edition. Reading, MA: Addison-Wesley, 1995.
8. Humphrey, Watts S. *Winning with Software*. Reading, MA: Addison-Wesley, 2002.

Additional Resources

1. Callison, Rachel, and Marlene MacDonald. *A Bibliography of the Personal Software Process (PSP) and Team Software Process (TSP)*. SEI, Carnegie Mellon University. Special Report CMU/SEI-2009-SR-025. Oct. 2009 <www.sei.cmu.edu/reports/09sr025.pdf>.
2. Hefley, Bill, Jeff Schwalb, and Lisa Pracchia. "AV-8B's Experiences Using the TSP to Accelerate SW-CMM Adoption." CrossTalk Sept. 2002 <www.stsc.hill.af.mil/crosstalk/2002/09/hefley.html>.
3. Grojean, Carol A. "Microsoft's IT Organization Uses PSP/TSP to Achieve Engineering Excellence." CrossTalk Mar. 2005 <www.stsc.hill.af.mil/crosstalk/2005/03/0503 Grojean.html>.
4. Lopez, Gerardo, et al. *TOWA's TSP Initiative: The Ambition to Succeed*. Proc. of the 3rd Annual Software Engineering Institute Team Software Process Symposium. Phoenix. 22-25 Sept. 2008.
5. Nichols, William R., et al. "A Distributed Multi-Company Software Project." CrossTalk May/June 2009 <www.stsc.hill.af.mil/crosstalk/2009/05/0905NicholsCarletonHumphreyOver.html>.

Note

1. For the basics of the TSP, see <www.sei.cmu.edu/tsp> and past CrossTalk issues (<www.stsc.hill.af.mil/crosstalk/2005/03>, <www.stsc.hill.af.mil/crosstalk/2006/03>, and <www.stsc.hill.af.mil/crosstalk/2002/09>). To examine more detailed information about the TSP, see the Additional Resources section of this article. For a summary of TSP project results, see <www.sei.cmu.edu/reports/03tr014.pdf> and slide 17 of <www.cmminews.com/2009/pdfs-sessions/73.pdf>. For more on organizations using TSP, see <www.sei.cmu.edu/tsp/case studies>.

About the Author

Watts S. Humphrey joined the SEI after his retirement from IBM. He established the SEI's Process Program and led development of the CMM for Software, the PSP, and the TSP. At IBM, he managed their commercial software development and was vice president of technical development. He is a fellow for the SEI, the Association of Computing Machinery, and the IEEE. He is also a past member of the Malcolm Baldrige National Quality Award Board of Examiners. In 2005, President George W. Bush awarded Humphrey the prestigious National Medal of Technology for his contributions to the software engineering community. He holds master's degrees in physics and business administration and an honorary doctorate in software engineering.

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

An Interview With Watts S. Humphrey

With more than 50 years in software and countless CrossTalk *articles, Watts S. Humphrey needs no introduction—but we will anyway.*

After World War II and academic work at the University of Chicago, Humphrey led an engineering group at Sylvania Electronic Products. Humphrey then joined IBM in 1959, where he worked on everything from fixing the OS/360 to leading projects as Director of Programming and Vice President of Technical Development. After retiring from IBM in 1986, he joined the SEI, where he established the Software Process Program, led development of the Software Capability Maturity Model, and introduced the Software Process Assessment and Software Capability Evaluation methods. Humphrey also led the development of the Personal Software Process℠ (PSP℠) and the TSP. At a White House ceremony in 2005, President George W. Bush awarded Humphrey the National Medal of Technology. Known as the "father of software quality," he is also the author of 12 books—with another one, "Leadership, Teamwork, and Trust: Building a Competitive Software Capability," on the way.

CrossTalk *talked with Humphrey, who delved into his past and present work, as well as discussed the future of CMMI®, PSP, TSP—and the software industry.*

Q: What were the personal experiences and values that influenced you while creating CMM, PSP, and TSP technologies?

Watts: Three experiences had a major impact on me.

Let me start with one of my first management jobs. I got hired by Sylvania in Boston to manage a fairly large circuit-design group that was building a great big cryptographic system. I had this group of young engineers all designing circuits, but I had been trained as a physicist and didn't know the first thing about circuit design.

Rather than fake it, I just spent my time asking them what they were doing and had them educate me. It was a different kind of management style than what people are used to. Usually, managers have done development work themselves, know how it ought to be done, and try to tell everybody how to do things. I couldn't do that. It was an education for me and highly motivating for the engineers. They loved it, and began to manage themselves. The exciting thing for me to discover was the fact that it's the only way you can manage really large groups. You discover when you get groups of hundreds or thousands of people—which I later did—that you can't manage what they are doing, so you need to count on them. That is the style I've used throughout my career. It's influenced everything I've done.

Basically you treat management as a continuous learning process, as a leading process, as a motivating process, and not as a directional process. So you're not telling people what to do—you're having them work it out and explain it to you and justify it. It really makes an extraordinary difference.

The second experience was at IBM where I was a crisis fixer. I found that the problems were never technical; they were always management problems. That's what I have struggled with trying to fix. I didn't know it then, but it's something I would be working on for rest of my life.

Fundamentally, you need to challenge people to prove to you that they are managing themselves: Putting motivation and accountability together turned out to be very effective. By and large I'd say that, with essentially no exceptions, all the crisis projects I led were successfully fixed. One example was an enormous project of 4,000 developers building an operating system for IBM. It was terribly late. We basically stopped everything for about 60 days and had them make plans, and it worked.

The reason I feel that the planning issue was critical comes from my third influence—my MBA education at the University of Chicago. For some strange reason, I decided to major in manufacturing. The manufacturing professor emphasized three things in management: planning, planning, and planning. Basically it's what he focused on throughout the whole course.

What fascinated me was that while hardware engineers have to work with manufacturing, the software engineers don't. The manufacturing people require plans, so the hardware engineers have to understand planning. The software people could manage their own work if they learned how to make plans and manage themselves. The CMM, the TSP, and the PSP all start with planning—it's the first step for everything you do. But software people are never taught how to plan. You can't just tell them to plan—you have to show them. That's a big part of what we do.

Q: As you expanded the PSP process to the TSP, did the industry develop at a slower or faster pace than you envisioned?

℠ The Personal Software Process and PSP are service marks of Carnegie Mellon University.

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Watts: Slower. I am reminded of a story about Fredrick Winslow Taylor[1]. More than a century ago he was working with a machine tool shop in England and invented the idea of lubricating the cutting tool: It was effective, it was very easy to introduce, and it cost practically nothing. He went back 20 years later and checked on machine shops in England and discovered that only one other shop was using it.

Now my point is that extraordinary methods that save an enormous amount of money are often relatively easy to put in place and are unbelievably effective—but they don't get adopted. You have to wonder why that is. I've concluded that there are three reasons.

First, every five years or so for the past 60 years that I've been working, some "magical new software method" comes along, and most of them don't work—except for the person who invented them. This has caused a lot of skepticism. Workable new ideas won't be believed until software engineers see them work for themselves. And they won't try it for themselves until they believe it will work—so you're stuck with a chicken and egg problem.

The second is that introducing new ideas is always difficult. When things are going well, organizations don't think they need to change—and when things are going badly they can't afford to change. So you're stuck and it takes people with great vision to see the strategic need to change even when they don't have to. Most only change when there's some pressure that makes them change.

And the third is that very few managers below senior levels are willing or able to take the initiative to introduce new methods—even when the benefits are obvious and proven.

Q: So what you're saying is they feel powerless.

Watts: Well ... they feel that way, but people really aren't powerless. I've talked to managers who have hundreds of people working for them and say they can't do anything. But I've been a manager with hundreds of people working for me and I've done it. I basically would just do what I felt I had to do and I'd tell my manager, "Here's what we're doing." I've always believed that if it makes sense and you can justify it, just do it and it almost always will work. Tell people in advance and don't ever surprise your boss, but I've always followed that old saying: "It is better to ask for forgiveness than permission."

Q: What are the other engineering areas to which you see the PSP/TSP expanding?

Watts: The PSP and the TSP are fundamental; they're not just about software. If you've read Peter Drucker, he started talking about knowledge work in 1959. Knowledge work is dramatically different from the typical work we do with our hands. It's work where you can watch knowledge workers but you can't tell what they're doing.

Now with typical engineering work—hardware engineering, manufacturing, construction, you name it—you can watch people doing their work and you can tell what they are doing and how well they are doing it. I mention in the article [in this issue], the method called "management by walking around" (MBWA): Managers go out and walk around the shop and see what is going on. As I say in the article, an increasing amount of the true work is done on computers and in people's heads. That's where the value is. And that is knowledge work.

So with a few exceptions, knowledge work is becoming pervasive—and why it's so extremely hard to manage some of these jobs. Managers don't know how to operate if they can't use MBWA—you can't use it with knowledge work. That's why software has been so extraordinarily hard to manage from the very beginning. Drucker's point was that the knowledge workers have to manage themselves, and that's what we are showing them how to do with the TSP.

Today's knowledge workers don't know how to manage themselves because they don't believe that they need to. They think that anything called management is the manager's job. So the knowledge workers don't manage themselves: they don't want to, they don't know how to, and they literally can't do it. As a result, knowledge work is not being managed—which means that projects often fail.

The TSP is designed for knowledge work. It's not just designed for software, it's designed for any kind of knowledge work—and we've used it with systems design, video game development with artists and game designers, as well as with software people and hardware groups. The TSP is universally helpful as a management system for just about any kind of complex work. One of the most impressive things to me was that we trained some Mexican software engineers to be TSP coaches—and they returned to Mexico and started a business. They've now grown that business to nearly 400 people, and they expect to be at 1,000 in a few years. They're even running their corporate office with the TSP.

The places where it can be used are almost limitless—and the benefits in software are so extraordinary that that's where knowledge workers are most likely to see the benefits of self-management. In software, you can cut test time incrementally. We've seen organizations that have been spending a year in test; with the TSP, they now spend a month and a half in test. You don't quite see that in the other areas, but it is equally applicable. The TSP is universally helpful as a management system for just about any kind of complex work.

Q: On a side note, our organization just finished a six-year training program where we trained more than a thousand leaders here and—I'll just be honest—we stole heavily from those concepts to get folks outside of software to use those types of disciplines. The thing that I noticed is people seem to inherently be attracted to the PSP/TSP—I mean, it just makes sense to them.

Watts: Well, people love it: Their morale goes up, they are excited about it, and their jobs are much better. PSP-trained people have to know how to manage, estimate, and track their own work—personally. If they don't know how to plan, they don't have the foundation required for self-management. That's why the training is so extraordinarily important. But managers just want to read a book on "how to be a team." That doesn't work because they don't really understand planning, manage-

ment, and tracking, or how to control a project.

Q: What is your vision for using the TSP concepts in whole organizations and how would they manage such an effort?

Watts: There are two ways to look at this. Let me first talk about a big company. I recommend that organizations start with a modest-sized area and run a few teams. Typically, they'll start with multiple projects of six to 12 people—which are great for TSP teams—and we have them run those projects and start building skills. Then, broadening TSP use across the business is purely a question of how rapidly the organization wants to go and how rapidly they can build the management skills. The engineering skills can be built very quickly: In one week, we can train an engineering team to use the TSP and be a TSP team. The problem is that it takes quite a while for management to understand what the TSP is all about. It's a change in management style, and changing management style in an organization is much more difficult. The engineers take to it like ducks to water—they just dive in. We typically have engineers refusing to work any other way after they've used it.

Second, there is the case where you've got an entire organization-wide program and the organization may have multiple companies, multiple locations, and it's all one great big job. When the people running these great big projects want to really know project status, the managers must go to the individual software and system design teams—and talk to the developers and ask them exactly where they stand on their schedules and how they're doing. Unfortunately, the teams can't tell them. They're guessing, they don't know. The project or team leaders may poke around and talk to everybody and they get vague stories. Then, when these managers talk to more senior management, they're guessing and defensive. They really can't level with management because they really don't know.

With the TSP, you don't sit down and argue and debate. You say: "Here is exactly where we stand, we've got these problems, and here we are." You discover that when you have the facts, your customers and your managers will work with you to solve problems. We have seen that with the Navy and with several DoD projects we've worked on. All of a sudden, instead of faking it, you know exactly what you're talking about. Nobody is guessing. You've got the data and you sit down and say, "Okay, we've got a problem to solve. How do we do that?" It is a totally different attitude.

In big programs you need to start at the base, get everybody using it, and have that whole attitude—of honesty, of leveling, of data, of facts—where you can really negotiate and deal openly with the management team. Building an environment of trust is absolutely crucial. We do not have that today in large programs because the facts aren't there to engender trust.

Q: PSP theory talks of experiencing a 100 defect per thousand lines of code (KLOC) defect density as typical for PSP software engineers (as well as the TSP teams they belong to) when developing software. With software teams using more powerful, context-sensitive editors for developing software, TSP teams with which we are familiar are seeing defect densities of anywhere from 50-70 defects per KLOC. Assuming this new "reality" exists beyond the teams we study, does this evolution in software development impact the way that PSP should be taught?

Watts: Well, the 100 defects per KLOC defect level is what we see when developers first start learning PSP. At the end of training, they're typically at 50-70 defects per KLOC or less. I've seen some down in the 20s. The numbers come down: People are using more disciplined methods and are aware of the defects they inject because they are measuring and tracking them. Just measuring and understanding the mistakes you make generates feedback real quickly. The numbers really do come down sharply when people understand their mistakes.

The second issue concerns the development environment. I've written some programs with .NET, with stuff like that. But those tools do not eliminate your errors and do not address the key problems with logic errors. All of these tools are designed to generate a working program from whatever the developer puts in, so it could very cleverly produce a working program from a highly erroneous design. Software people tend to think that when some tool fixes their trivial defects that all the defects are fixed. People have to be conscious of what they're doing and even more so when working with very sophisticated tools like .NET, where you can put in all these very complex functions that most people don't even understand. These languages are so complicated that very few people ever really understand them completely.

Q: Is it safe to say they are getting a false sense of security, as far as these defects go, by using some of these tools?

Watts: Yes, that is true. Powerful tools can lead to powerful mistakes. The whole process of designing tools and languages is aimed at richer and richer capabilities. Not a whole lot of attention is paid to understanding why developers make errors and/or how to design languages and tools that minimize human error. I've hoped that the academic community would look at this, but unfortunately no one has done it yet. It is time they started. We really do need that kind of help.

Q: You have stated that your personal goal since the mid-80s has been to transform the world of software engineering. In what ways have you succeeded in this, and what unreached goals do you hope to meet in the future?

Watts: When I retired from IBM, I made the outrageous statement that I was going to "change the world of software." You can never really do that sort of thing by yourself, but it was really motivating. What I found fascinating was that people got excited about it. They joined in. I got a

whole movement going and it was marvelous. When you get people working with you, you can get a lot done. That has been exciting—and we've had some remarkable successes as well as some real disappointments.

Let me talk about the successes first. With the OS/360[2], after I took over that project [at IBM], we put together a new plan and we put in place the management systems I've been talking about. This was at a much earlier level—we didn't understand it all then. But we [then] didn't miss a single delivery date for two and half years. There are not many people who have done that with big software systems. We put out the first 19 releases of OS/360 on schedule. Take a look at Microsoft or anybody else today: They never deliver on schedule, but we did, and it made an enormous difference.

Okay, so that is the success. We now have a basic understanding of this stuff. We know how and why software costs, schedules, and quality have been out of control—and we know what to do to fix them. I'm not saying that we have solved every problem, but when we work with organizations, we can help them build the capabilities they need to consistently deliver quality products on schedule. And it works: We have seen it with hundreds of teams across many businesses; we've seen it work with small two, three, four person projects, up to great big multi-company programs.

What is so disappointing is the acceptance of these ideas. The defense industry hasn't really looked at this at all. One of the main reasons and one of the big objectives I had when I started this whole thing was to address this national need: We have these enormous programs and our whole defense industry and military preparedness is dependent on these projects getting completed—and on time.

During my very first SEI project, I was working with the electronic systems command. That's where we started the CMM, the predecessor of CMMI. Every project was failing. They were all behind schedule—on average 60 to 70 percent late, and costs were at least twice what was planned. There was a recent report in the New York Times[3] saying that two-thirds of the largest DoD weapons systems ran over their budgets and the combined extra cost was $296 billion dollars—and they were on average two years behind schedule. This is a tragedy. We know how to do better—we are just throwing money down the drain.

I had a major executive on a defense contract ask me, "You mean to tell me you want me to spend profit dollars to cut revenue?" These companies today are being paid to do crappy work and then fix it later. And with the current system, you and I as taxpayers just keep paying for it. The more junior-level managers can't fix it, and it isn't that they don't want to; my guess is that they would love to, but can't. They are measured on revenue and profit and they literally cannot reduce it. They can't spend profit dollars to train their people to do quality work and have their revenue go down. The whole structure does not allow them to do it.

"[The TSP] lets teams know precisely where they stand. They can give data to their managers, they can tell them exactly what is going on, and they can identify any problems."

The DoD is constantly struggling. They are trying to change procurement regulations and trying to change managers and get smarter people, but there will be exactly the same problems until they start to deal with the fundamental management system that's currently being used. And that is what the TSP does: It manages knowledge work. It's not dealing with the work as something you can walk around and watch, because you can't. Knowledge work is invisible to the managers and if we continue to operate in the same way, none of the band-aids the DoD is trying will ever fix it. I've seen it for 50 years. So it is not going to change in the next five, 10, or 20 years until leadership begins to realize that we have to try something different.

And the TSP is different. It lets teams know precisely where they stand. They can give data to their managers, they can tell them exactly what is going on, and they can identify any problems. Most crises in big programs are obvious: They are identified years ahead by somebody way down in the trenches. And usually those people don't feel that they own the project. They assume somebody else will handle the problems, so they just go on with their jobs—and the crisis blows up.

What is exciting about TSP teams is they actually do risk analysis and track problems. They take ownership and assign individual team members to track and manage them. We have standard roles for TSP teams—a customer interface manager, a test manager, a design manager, and others—and each of these roles is assigned to a team member. So you now have ownership at a team level; you have team members who will bring issues upstairs when they need to. Instead of hiding their problems and going on with their work, they're actually addressing issues proactively and getting management's help when they need to—and it works.

We need that attitude throughout these enormous programs in the DoD. It must start down at the root level. If you have it there, it will build all the way up to the executive level. When the managers know what they are talking about, you begin to get cooperation between the defense contractors and the DoD. And the DoD can now deal honestly with Congress. Right now, Congress doesn't trust the DoD because they can't get the facts and everything is a surprise.

The other disappointment is the academic community. With few exceptions, computer science and software engineering programs have shown no interest in the TSP and PSP. Until they start teaching this stuff, their graduates won't understand it and industry will have to re-educate their people—and that's expensive. The way people are working today, they're basically beating their heads against the wall, testing until midnight, in at all hours. Nobody likes it, it's a painful job, there are failures all the time—and it has become a very unattractive career.

The U.S. Census Bureau did a study some time ago—and unfortunately I don't have the reference—forecasting that within 10 years, 50 percent of the people doing software work would leave the field. These are enormously talented and skilled people who have had 10 years of experience that are just going off and doing other things. They make some money and then they leave. They can't take it. It's because they don't know how to manage themselves; they don't know how to work in this envi-

ronment. The academic community really has to get on board: understand it and teach it. They ought to lead this charge.

Q: Do you see issues between the model community (CMMI) and the process community (TSP) and, if so, what are your thoughts on how to overcome such differences?

Watts: Frankly, in the past, we have had differences, but they were principally due to misunderstandings. The groups had not been working that closely together. We were basically on our own paths. Our challenge was to figure out how to make this stuff work. Now we've worked through these differences and see that the two approaches work together extremely well. The CMMI people now see that it works.

Fundamentally, one of the big problems CMMI has now is performance—performance for high-maturity organizations. The CMMI community is beginning to see that what we've got *does* work. And so we are beginning to work together; CMMI and the TSP are very complementary. We are now working as a coordinated group to figure out how we can better help people improve their organization's performance and accelerate process improvement.

Q: What is your opinion of the direction that CMMI appears to be headed for high maturity? Specifically, do you believe that we will see benefits from the kinds of Process Performance Baselines and Models for which lead assessors are now looking?

Watts: Let me talk about the two kinds of processes: procedural and operational.

CMMI is an excellent example of a procedural process. Fundamentally, it sets organizational standards and sets baseline procedures across a business.

The CMMI framework is exactly what we did at IBM: We defined standard milestones where the teams had to go through six project review steps before they could go out the door. They had to do this before they could get funded, before they could announce a product—that sort of stuff. We had steps that all the projects had to go through: guidelines for quality assurance, testing techniques, and inspection procedures. We established a review procedure and all the involved groups participated—the maintenance people, the marketing people, the support groups, and so forth. They all had to sign off. While this was a lot of bureaucracy, it forced the organization to do things that TSP teams do naturally. But it worked real fast. Since we were in a crisis, we needed that.

Before CMMI, we didn't have that sort of thing—everybody was off doing their own stuff, nobody had a standard framework, none of that. CMMI is extremely helpful in stabilizing an organization and getting a level of statistical control: It is repeatable; you can more or less get stuff to work in a predictable way. And so that is how CMMI—in Levels 2, 3, 4, and ultimately 5—stabilizes an organization and begins to build the kind of foundation you need for real improvement. So CMMI is what I call a procedural process.

Now you get to an operational process where you are talking about what the development teams do when they develop software. How do they do it? How do they manage quality and cost and schedule? What data do they gather? What measures do they use? We found that until you provide specific guidance to the developers, they won't do it.

Think of it this way: When you tell a developer, "I want you to make a plan," they don't have the vaguest idea of how to do it and they don't even know what one looks like. That's what I saw at IBM. Everybody was coding and testing. Everybody knew that they ought to have plans and requirements, and they knew that they ought to have all this other stuff in place—but they didn't know how to do it. I put a thousand managers through a course on how to plan. If we hadn't done that, the managers wouldn't have been able to make plans. We put that in place and it worked. It was extraordinary.

So the whole idea here of the procedural process is to build the base capability and then begin to move toward an operational process where people really do what they have to do to generate the data, manage the quality, and build the performance of the organization. So that's the distinction between the procedural process and operational process. With a procedural process, you usually need a bureaucracy to enforce it, but with an operational process—as long as the teams are properly coached—they can be trusted to do their work properly and the bureaucracy is unnecessary. So the tradeoff is coaching versus bureaucracy.

When we originally put together the whole maturity model framework, we were doing it for the acquisition community. We knew that we had to give guidance on the question, "What do you look for?" So we focused on artifacts. What is the evidence of an organization's performance? Say we want an organization that is producing plans and that uses configuration management and requirements management. What are the things that you'd have to have if you did that? You could say, "Well okay, if you do planning then you ought to have plans." You can now look around and say, "Do you have plans?", "Do you have review meetings?", and "Do you have review meeting minutes?" If you have configuration management then you ought to have configuration management audits, reviews, and updates so that there are actual artifacts produced as a natural consequence using the process. It shouldn't be expensive to produce. If you're actually using that process those are things you ought to naturally have; you look and make sure they're there.

When we put together the original maturity models, this is what we did. While the acquisition people didn't really understand the details, they could tell that somebody had a development plan, it was for this project, it was signed off, and it had what appeared to be the right stuff in it. It was fairly easy to do. The original intent was that these artifacts were the natural consequence of the process being used, so there shouldn't be a lot of cost involved in preparing for such a review.

Now notice what happened with CMMI: Appraisals became important so organizations were in a great hurry to reach a high maturity level. Increasingly, organizations discovered that it is extremely hard to change what the development teams actually do. It's a heck of a lot quicker to have task groups generate documents that meet the needs of the appraisal. So you've got groups that put together configuration plans, development plans, and all of this stuff. And it's not developed by the developers—but

there is nothing in CMMI says that it's wrong to do this—so you've got all of these artifacts. Now you have these independent groups bureaucratically producing stuff that has no relationship to the work that is being done. And so you don't improve organization performance at all. Unfortunately CMMI, as currently built, doesn't protect against that. And so that's what we need to focus on: How do we work together so the CMMI and TSP folks really focus on what it takes to have a high-performance organization?

This is why the performance idea is so critically important. If you really are talking about data and measurement, you have to think about performance in a different way. You need to show that you not only have the artifacts but you are getting the performance the artifacts should produce. So that's the thing that we are talking about. Up to this point, when we've put people together, it has cut the cost and time for process improvement. It accelerates the movement from one level to another and produces dramatic performance improvements.

Q: Where do you see things going in the future? Do you see the DoD taking more steps to utilize the TSP?

Watts: In terms of where we are going, the future is exciting. I've found that even enormous programs can be managed. Can you imagine how our economy would work and how the DoD would function if people could actually put together plans for these massive programs and then delivered them on schedule and for their planned costs?

People don't understand when I say, "Delivering on cost and within schedule," that this will be a fundamental problem for the DoD. It doesn't mean that the teams can deliver on whatever schedule the politicians or generals demand, it means that the development teams themselves—when they know how to manage their own operations and put together their own plans—can go to the generals. Then the generals can go to the politicians and say, "Here is what it is really going to take." Instead of saying, "We're going to do it in 18 months," you may do it in 30 months, but people will actually deliver on schedule and they will meet their cost goals with quality products.

We are seeing that time and time again. We are seeing it with big teams and we've even seen it with multi-company teams where you have people working together across several companies. There was one case with two competing companies[4]—under a DoD contract—where they actually did deliver on schedule and the product really did work. We heard the customers say, "This is extraordinary. We're not going to work any other way." So we know it can be done.

So the customers will like it, the politicians will like it, the generals will like it, the users will like it, and we'll get a hell of a lot faster stuff out there to the fighting forces. It's a very exciting future. I hope I'm there to see it.◆

Notes

1. See <http://en.wikipedia.org/wiki/Fredrick_Winslow_Taylor>.
2. IBM's OS/360, officially known as the IBM System/360 Operating System, was developed for IBM's then-new System/360 mainframe computers. The multiple virtual storage version of OS/360 was the first large-scale general purpose operating system and it was one of the first to make direct access storage devices a prerequisite for their operation.
3. See <www.nytimes.com/2009/03/31/business/31defense.html>.
4. This project is detailed in the May/June 2009 CROSSTALK article, "A Distributed Multi-Company Software Project," co-written by Humphrey with Dr. William R. Nichols, Anita D. Carleton, and James W. Over. See <www.stsc.hill.af.mil/crosstalk/2009/05/0905NicholsCarletonHumphreyOver.pdf>.

Updating the TSP Quality Plan Using Monte Carlo Simulation

David R. Webb
309th Software Maintenance Group

The 309th Software Maintenance Group at Hill AFB has started implementing an updated version of the TSP quality plan utilizing Monte Carlo simulation. This article presents an overview of why an updated quality plan with variability is needed, what data the model requires to be useful, and how the new model works. Actual data from Hill AFB projects that have implemented this method are presented for review.

The TSP quality plan is composed during meeting 5 of the launch[1] by determining the defect injection rates and yields for each phase of the product development process. Using the team's historical averages for these rates and estimated hours per phase, the team can predict how many defects will likely be injected and removed as products move through this process. Unfortunately, these averages do not take into account normal variability in the process. However, by applying a Monte Carlo simulation to the standard TSP quality planning process, a team can determine the historical distribution of process variability and produce a plan with ranges for expected defects injected and removed, as well as a *measure of goodness* for the product and process.

The TSP Quality Plan

One of the hallmarks of projects using the TSP is the attention to quality or, more accurately, the ability to manage product defects. In fact, TSP creator Watts S. Humphrey says:

> ... defect management must be a top priority, because the defect content of the product will largely determine your ability to develop that product on a predictable schedule and for its planned costs. [1]

A chief component of this focus is the quality plan developed during meeting 5 of the TSP launch (for a project). This plan is composed by estimating defects injected and removed during the various phases of the software process. The team uses historical averages of defects injected per hour to determine defects injected and similar averages for yield (the percent of existing defects found and fixed during a phase) to determine those removed (see Table 1 for a sample quality plan). According to Humphrey, the true purpose of the quality plan "is to establish team yield goals for each process step" [2]. If the team does not have sufficient historical data, average injection and removal data collected by SEI can be employed. Using this approach, the team estimates final product quality and then determines whether or not that quality will meet their customer, management, and team goals. If those goals are not met, the team decides what process changes should be made to meet them.

Once the plan has been developed and the launch completed, it is the role of the team's quality manager (assigned during the launch) to monitor progress against the quality plan. Results of the monitoring activities are discussed during the team's weekly meeting. In addition to monitoring actual values for defects injected and removed, the quality manager can help focus the team on quality issues by examining other metrics, such as the defect removal profile (the defects per thousand lines of code removed from software components as they move through the development life cycle) and the product quality index. Exercises such as the capture-recapture method[2] can even predict how many defects may have escaped a personal review or inspection. When done properly, these measures, metrics, and activities can improve the team's quality focus, reducing rework and improving on-time and within-budget performance.

Many TSP teams that have no issues with most TSP concepts struggle with this progress monitoring. While teams are excited about producing the quality plan during launch, the quality manager no longer reports quality progress after a few weeks—other than announcing when the next quality event (inspection, test, etc.) will take place. Let's say, at the project post-mortem, that a team dutifully collects the quality data needed for the next launch, but notes in the lessons learned that they "need to do a better job on the quality plan in the future." In my experience, there are few key reasons for this fall-off of the quality focus:

- The team has not collected sufficient historical data for defect injection and removal; they utilize the by-the-book numbers provided by the SEI, but do not really believe them because they are not their numbers.
- Historical averages blend the results of high performers with average or low performers. Depending upon who is working on a module or series of modules, the predictions may or may not truly represent the work being done, so the team doesn't trust them—and certainly does not use the predictions to guide their work.
- Defect injection rates (DIRs) are based upon the effort estimate for each module; while TSP teams are great at using Earned Value techniques to balance workloads to meet their estimates, not every module is accurately estimated, making the defect injection numbers suspect.
- Team members are not consistently collecting defect data; either individuals are counting defects differently or they are not measuring them at all, making any defect prediction model inaccurate, and thus, unusable.
- When actual data begins to come in, the quality manager, team leader, and sometimes even the coach don't really know what to make of it (e.g., does a lower number of defects than expected mean the team is just very good, or that the quality activity was badly executed?).

These issues can be addressed by two basic practices: 1) consistently collecting data; and 2) properly using the concepts of variability in developing and tracking the quality plan. What follows is an examination of some simple ways to ensure quality data are consistently and properly collected, and a discussion of how to use Monte Carlo simulation to account for inherent process variability—in turn making the quality plan more accurate and usable.

Consistent Data Collection

From an examination of the data of 10 randomly selected PSP students from various classes over a five-year period, it becomes obvious that the rate of defects

injected per hour varies widely by person (averaging 0-60 per hour); even the plots of the averages of defect injection rates in design (averaging from 0-30 per hour) and code (averaging from 2-10 per hour) show that every person is different—sometimes vastly different.

While some of this variability has to do with individual capabilities, the programming environment used, the difficulty of the assignment, and personal coding styles, much of it also has to do with common operational definitions and recording practices. Anyone who has taught a PSP class has noticed that not everyone fills out their defect logs the same way: Some students record several missing semi-colons as a single defect then fix them all at once, while others count each semi-colon as an individual defect with distinct fix times. Most instructors allow this individual style of defect logging, as long as the student is consistent in the method used; however, when determining team defect injection rates, this kind of instability in definitions and recording methods can cause a prediction model to behave erratically. This leads the observer to doubt the validity of using personal defect logs, unless all engineers are somehow coerced into using identical logging techniques.

Another reason to suspect that personal defect data may not be the best fit for a quality prediction model can be seen in the actual project data. The distributions in personal defect logs were collected over an 18-month period from a TSP team at Hill AFB. During this project's execution, the variability in personal defect logging noted in the classroom data did not stabilize or become more consistent. The most disturbing trend in these data is the severe lack of personally recorded data, as evidenced by the number of engineers with data from only one module or no defects logged at all. It is important to note that these data come from a team with strong coaching and a heavy quality focus (they have never released a major defect).

For these reasons, it appears to be undesirable to use personal defect log data for defect injection analyses. That being the case, the question becomes: What kinds of data would make sense? Interviews with the engineers on the noted project (as well as other TSP projects at Hill) suggest that more consistency may be found in defect data from inspection and test databases. These public databases require more strict control to ensure that defects are properly identified, analyzed, addressed, and tracked. This typically requires users to enter data according to a defined procedure and to use common definitions for defects and defect types. This kind of control seems to drive more stable operational definitions and data recording practices than evidenced in the personal defect logs.

When looking at the design and code inspection data from our TSP project, it shows that the distributions are much tighter than those in the personal logs without the problem of a lack of recorded data. That being said, there is still some variability in the data—in this case, higher in the code inspections than the design inspections. For example, the average DIR on both the design and code review data is toward the lower end of the distribution, suggesting a skewed normal or lognormal distribution in defect injection rates.

Therefore, a possible conclusion of this analysis is that personal defect log data is not as useful in creating a quality model for the quality plan as is data from public databases, such as the inspection and test databases. However, even in these data, the defect injection rates display a certain amount of variability that should be accounted for in our quality model.

One very important note here is that this analysis should not be used to suggest or validate the idea that personal defect logs are not useful. Several engineers interviewed found them very useful for personal improvement—they simply are not consistent from person to person, making the data unusable for team modeling purposes. Strict coaching and quality manager oversight, focusing on common operational definitions and recording procedures, may make these data more usable.

TSP (v1) Rollup Plan Summary Quality Summary

Defect Density

Defects/KLOC	Plan	Actual
Detailed Design Review	164	
Detailed Design Inspection	49.1	
Code Review	395	
Compile	87.9	
Code Inspection	61.6	
Unit Test	31.1	
Build and Integration Test	2.76	
System Test	0.55	
Total Development	1038	
Total	1.04	

Inspection/Review Rates

	Plan	Actual
Code Review	28.5	
Code Inspection	5.51	

Cost of Quality (COQ)

	Plan	Actual
Percent Appraisal COQ	32.70%	
Percent Failure COQ	4.69%	
Appraisal/Failure Ratio	6.98	

Phase Yields

	Plan	Actual
Requirements Review	70%	
Requirements Inspection	70%	
High-Level Design (HLD) Review	70%	
HLD Inspection	70%	
Detailed Design Review	70%	
Code Review	70%	
Compile	50%	
Code Inspection	70%	
Unit Test	90%	
Build and Integration Test	80%	
System Test	80%	

Defect Injection Rates

Defects Injected per Hour	Plan	Actual
Requirements	0.25	
HLD	0.25	
Detailed Design	0.75	
Code	2	
Compile	0.3	
Unit Test	0.07	

Table 1: *Sample TSP Quality Plan Created During Meeting 5*

Monte Carlo Simulation

One method of taking into account the variability of the defect injection rates and yields in a quality model would be using a technique called Monte Carlo simulation. The Monte Carlo method is any technique using random numbers and probability distributions to solve problems [3, 4], using the brute force of computational power to overcome situations where solving a problem analytically would be difficult. Monte Carlo simulation iteratively uses the Monte Carlo method many hundreds or even thousands of times to determine an expected solution.

The basic steps of Monte Carlo are as follows:

1. Create a parametric model.
2. Generate random inputs.
3. Evaluate the model and store the results.
4. Repeat steps 2 and 3 many, many times.
5. Analyze the results of the runs.

This is useful in creating a form of prediction interval around an estimate. For example, assume the number of defects in a software product (in the design phase of development) can be predicted by multiplying the historical defects injected per hour by the number of hours estimated for the phase. We can improve that estimate by using the ratio of historically estimated hours to actual hours, known as the Cost Productivity Index (CPI). The CPI

Continued on Page 18

Technology: Changing the Game

The 22nd Annual Systems and Software Technology Conference

Brig Gen John B. Cooper gives the opening general session remarks.

SSTC attendees in-between track sessions.

Marek Steed, CROSSTALK *article coordinator (far right), talks with visitors at the Software Technology Support Center trade show booth.*

The 2010 Systems and Software Technology Conference (SSTC), held April 26-29 in Salt Lake City, explored various technologies which are expected to make abrupt changes to common thought. Participants explored the tools, processes, and ideas which will change the game and make the way we have done things in the past obsolete.

The SSTC kicked off with Monday tutorials ranging from people technology to Agile software and systems engineering. After opening general session remarks by Brig Gen John B. Cooper, Commander of the 309th Maintenance Wing, afternoon sessions focused on assurance/security issues, modernizing systems and software, new processes, and lessons learned.

Along with a full slate of presentations, Tuesday marked the start of the always popular two-day trade show, including booths from IBM, INCOSE, the SEI, and the Software Technology Support Center—the organization behind CROSSTALK.

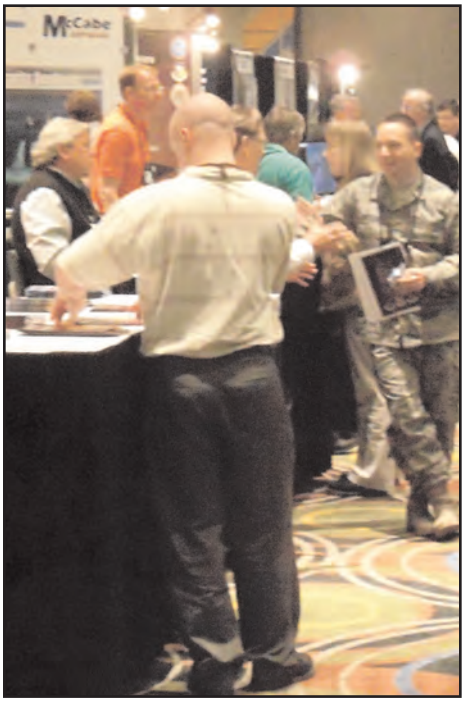
Wednesday proved to be the most action-packed day for conference-goers, from the plenary breakfast sessions ... to presentations ... to the trade show luncheon ... to dinner and the a cappella singing group Voice Male.

As with previous years, there were several CROSSTALK authors represented among the presenters. There was also some good follow-up work during the plenary sessions: Dr. Azad Madni's "Integrating Humans with Software and Systems" was a great companion piece to CROSSTALK's *Software Human Capital*-themed May/June 2010 issue; and Dr. Robert Cloutier's "Evolutionary Capabilities Developed and Fielded in Zero to Nine Months" presented an extended and updated version of his May/June 2009 CROSSTALK article (with Portia Crowe) of the same name.

Information for the 2011 conference will begin appearing in your e-mail and mailboxes beginning in late August with the "Call for Speakers" brochure. We can't wait to start reading those abstracts!

Photography by Drew Brown, Marek Steed, and Bill Orndorff

Above: David R. Webb (right) of the 309th Software Maintenance Group and CROSSTALK *publisher Kasey Thompson (left) present "Combining TSP, CMMI, Project Management, and People Skills to Create Better Software" with Larry W. Smith (unpictured).*
Right: Lt. Col. Scott Brown of the Directorate of Science, Technology & Engineering leads a panel discussion of software technology readiness levels and assessments.
Bottom Right: Hillel Glazer of Entinex expands on his January/February 2010 CROSSTALK *article "Love and Marriage: Why CMMI and Agile Need Each Other."*



Conference-goers enjoy the trade show.

Wednesday's dinner social.

Continued from Page 15

represents how well tasks have been estimated in the past; a number near 1 means that estimates have been fairly accurate in the past; a number greater than 1 tells us that we tend to overestimate; a number less than 1 says we typically underestimate our tasks. Dividing the estimated hours by the CPI will compensate for any tendencies to over- or underestimate. Thus, our final prediction equation for design defects injected is the DIR for design multiplied by the number of estimated hours in the design phase, divided by the CPI for design. This is the parametric model needed for step 1 of the simulation:

$$d = DIR_{design} \times Hours_{design} \div CPI_{design}$$

In step 2, we need to generate random inputs to the DIR and CPI variables of the equation, since these are parameters that are subject to variability in our historical data[3]. The question is: Where do we get these random values from? The answer can be found by examining each of the variables. For example, the typical TSP approach to estimating design defects would be to use the average historical values for the DIR and CPI, as defined in Table 2. The only problem with that approach is that, while the average DIR in design is 2.1, it can vary from 1 to 5, in a lognormal fashion. Additionally, the historical data in Table 2 shows that the average CPI for design is 1, but it varies from 0.5 to 1.5 according to a normal curve. With this in mind, we would use these distributions to generate our random input data for step 2 of the Monte Carlo process. Having estimated that 8.3 hours will be spent in design, we randomly select values from each of these distributions, choosing 0.88 defects per hour for the DIR and a value of 1.12 for the CPI. Therefore:

$$d = 0.88_{defects/hour} \times 8.3_{hours} \div 1.12 = 6.52_{defects}$$

This gives us the value of 6.52 defects, which is how we evaluate the model and store the results for step 3 of the process.

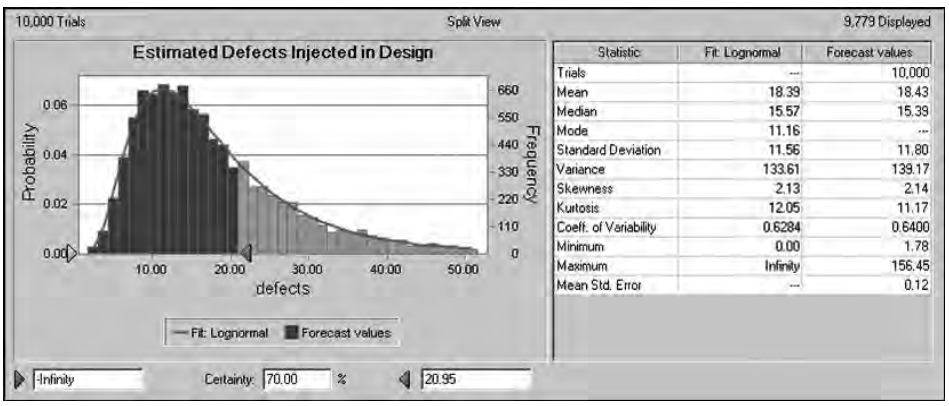
Step 4 of the Monte Carlo process simply requires repeating steps 2 and 3 many, many times—each time storing away the newly generated answers. Let's say we do 10,000 of these calculations and store them all away; when complete, we will have built up a new distribution for "d", the results of the equation.

Step 5 of this process is examining the distribution of the results to determine what we can learn. In Figure 1, we can see that the answers from our equation using the Monte Carlo process fall into a lognormal distribution, with a mean of 18.39 defects and a standard deviation of 11.56. Further analysis of the data suggest that 70 percent of the time, we should expect no more than about 21 defects will be injected in the design phase of our process. This provides us a bit more insight than we would see in a typical TSP quality plan. For instance, we now know that if there are fewer than 21 design defects found during our project, it's not necessarily a bad thing; however, if we find more than this, say 40 defects, something may be out of the ordinary (since that happens rarely). If we find many more than 21 defects—200, for example—then we can be pretty certain we have an issue that needs to be addressed. The wonderful thing about this is that we can determine these parameters at planning—a concept that fits well with TSP principles and philosophies.

Table 2: *DIR and CPI Notional Historical Data*

Project	DIR-Design	CPI-Design
P1	1.02	0.50
P2	1.33	1.15
P3	2.06	0.67
P4	1.13	0.88
P5	5.00	0.96
P6	2.50	1.35
P7	1.30	1.50
P8	4.10	0.62
P9	3.20	1.50
P10	1.08	1.38
P11	1.00	0.98
P12	1.62	0.89
P13	1.88	0.78
P14	3.10	0.88
P15	1.23	0.92
Average	2.10	1.00

Figure 1: *Sample Distribution of Results from Monte Carlo Simulation of Defects Injected in Design*



Using Monte Carlo Simulation for the TSP Quality Plan

There are essentially five steps in modifying a TSP quality plan to take advantage of the previously described Monte Carlo simulation techniques:

1. Gather historical data and determine distributions for the DIR, yield, and CPI.
2. Modify the equations that determine defect injection, defect removal, defects remaining, and any other metrics important to the team.
3. Run the Monte Carlo simulation using estimates for hours per process phase and the distributions for the DIR, yield, and CPI.
4. Examine the results, determine how well project goals are addressed, and come up with next steps for the project.
5. Use this plan to guide and track the project's quality progress.

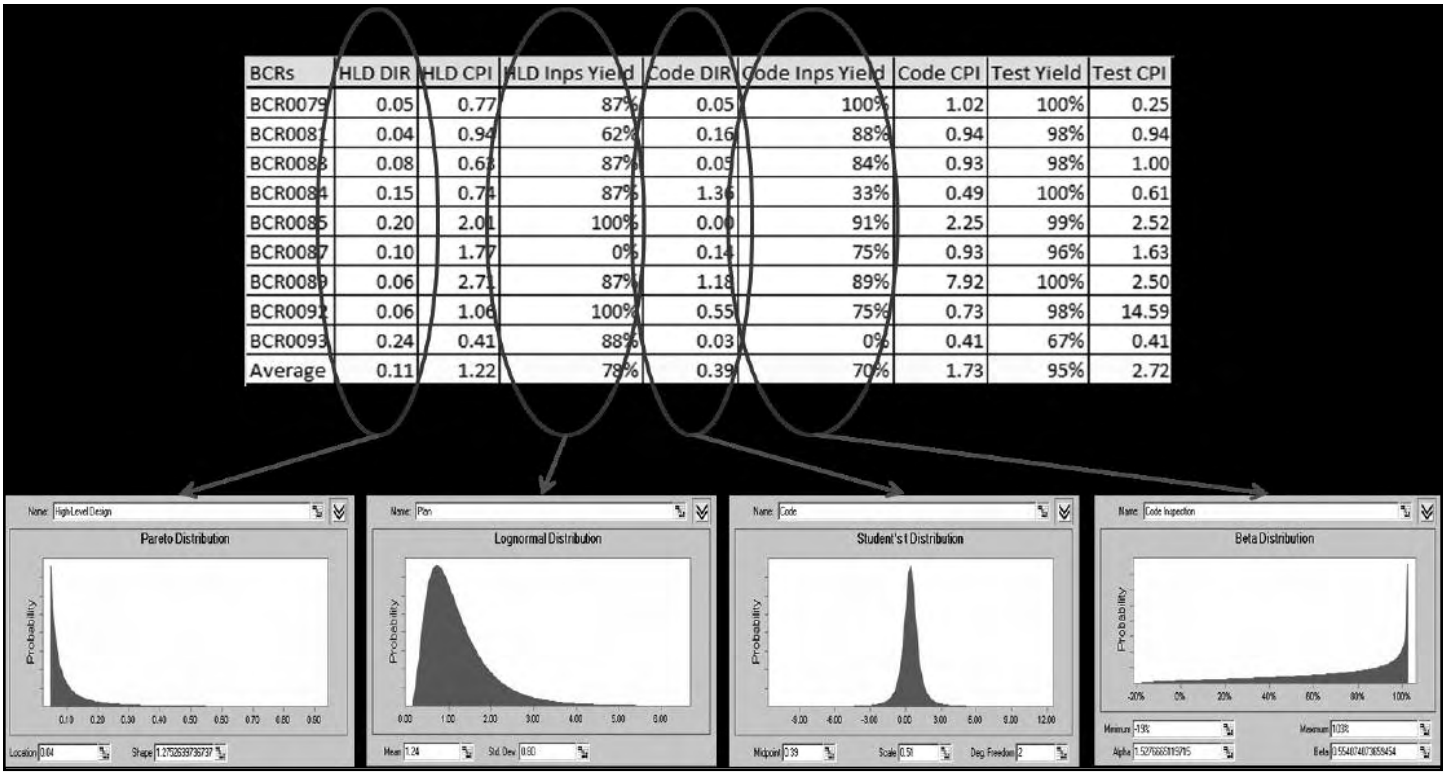
Gathering Historical Data and Determining Distributions

The first step is fairly straightforward for TSP projects that have been using the process for a while and have post-mortem data available. The team simply needs to gather data on the DIR, yield, and CPI for a number of past projects to determine the actual distributions of data. This can be done on a project-by-project basis, or by module, capability, or build (as desired). In Figure 2, the actual data from a Hill AFB project are listed as Baseline Change Requests (BCRs) and represent code changes made to an existing software baseline over 18 months. In this example, the team used Oracle Crystal Ball (a spreadsheet-based application suite for predictive modeling) to determine the distributions of each set of data.

Once the data gathering and analysis have been done, the team must determine the quality planning parameters[4], as shown in Table 3.

Modifying the Equations

Currently, the TSP quality plan predicts



BCRs	HLD DIR	HLD CPI	HLD Inps Yield	Code DIR	Code Inps Yield	Code CPI	Test Yield	Test CPI
BCR0079	0.05	0.77	87%	0.05	100%	1.02	100%	0.25
BCR0081	0.04	0.94	62%	0.16	88%	0.94	98%	0.94
BCR0083	0.08	0.63	87%	0.05	84%	0.93	98%	1.00
BCR0084	0.15	0.74	87%	1.36	33%	0.49	100%	0.61
BCR0085	0.20	2.01	100%	0.00	91%	2.25	99%	2.52
BCR0087	0.10	1.77	0%	0.14	75%	0.93	96%	1.63
BCR0089	0.06	2.71	87%	1.18	89%	7.92	100%	2.50
BCR0092	0.06	1.06	100%	0.55	75%	0.73	98%	14.59
BCR0093	0.24	0.41	88%	0.03	0%	0.41	67%	0.41
Average	0.11	1.22	78%	0.39	70%	1.73	95%	2.72

Figure 2: *Historical Data with Distributions*

measures that are useful in the planning stages of the project and can be used to guide the engineers during project execution. Some of these measures include defect densities per phase of review/inspection, review rates, and appraisal-to-failure ratios. In crafting the new quality plan, we can now be more specific and predict the expected number of found defects during each quality phase and how many defects are remaining in the product, with a prediction interval. The equations for doing this are a modification of the equation previously created, predicting how many defects will be injected in the design phase. Using this formula, we simply multiply by the planned yield of the inspection phase to estimate how many defects will be removed[5]:

$$d_{\text{design inspection}} = DIR_{\text{design}} \times HOURS_{\text{design}} \div CPI_{\text{design}} \times YIELD_{\text{design inspection}}$$

Similar equations can be generated for every phase, based upon the historical data from Figure 2. We can then use these equations, along with the distributions identified, to determine the results for our Monte Carlo simulations, as shown in the estimated defects portion of Table 3[6].

Running Monte Carlo Simulation and Examining the Results

At this point, during meeting 5 of the TSP launch, the Monte Carlo simulation is run with the variable inputs and the prediction equations. The simulation can create distributions of results for all 14 predictions highlighted in Table 3. The team can predict, for example, the minimum number of defects they would expect to find in each inspection phase, within a given prediction range (e.g., 70 percent of the time). In this case, the total number of defects found in detailed design inspection should be at least 456, and 633 in code inspection, 70 percent of the time, according to historical data.

To make this prediction even more useful, the team should run Monte Carlo simulation for each module following launch meeting 6. At this point in the TSP launch process, bottom-up plans have been made and hours have been estimated for each process phase of every module in the next-phase plan. Assuming every individual performs within the parameters established from the team data, the Monte Carlo simulation can now be run for each module. Table 4 (see next page), for example, shows a single BCR update to a software baseline, with its own design and code inspection predictions. Note how the numbers are much lower for this single update than for the combined numbers of the entire project update. When the Monte Carlo simulation is run for these planning numbers, the charts look similar. However, the key advantage is that we can now predict that 70 percent of the time the design inspection for this update

Table 3: *Sample Planning Parameters for a New Quality Plan*

TSP Quality Plan with Monte Carlo

Estimated Time	Plan	Actual
High-Level Design	434.93	
High-Level Design Inspection	147.28	
Code	901.87	
Code Inspection	175.82	
Unit Test	275.87	

Defects Injected per Hour	Plan	Actual
High-Level Design	0.11	
Code	0.39	

Project CPI	Plan	Actual
High-Level Design	1.22	
Code	1.73	
Unit Test	2.72	

Phase Yields	Plan	Actual
High-Level Design Inspection	78%	
Code Inspection	70%	
Unit Test	95%	

Estimated Defects Found	Plan	Actual
High-Level Design Inspection	30.61	
Code Inspection	143.83	
Unit Test	65.82	

Estimated Defects Remaining After	Plan	Actual
High-Level Design Inspection	8.85	
Code Inspection	154.21	
Unit Test	88.39	

Software Defense Application

The software defense community will benefit from utilizing the proposed TSP quality plan update, as this article shows how to determine and apply variability into the plan through Monte Carlo simulation. Users will be able to predict product and process quality at stages throughout the life cycle and at delivery. It will also help in meeting requirements for Quantitative Project Management and Organizational Process Performance at CMMI Level 4. These methods closely track product and process quality, providing tools for project managers in avoiding cost and schedule pitfalls—and in delivering near zero-defect products.

should find at least three defects (although it would not be unusual for the code review to find zero). This gives us some indication of the goodness of the inspections and a lower limit that we can look for during the execution of the project. Likewise, in the unit test for this change, we should find no more than 12 defects, 70 percent of the time (see Figure 3). In this case, we look for the upper limit, since our goals are to find more defects in inspections than in testing.

Guiding and Tracking Project Progress

Once the TSP launch is complete and the plans are approved by management, the team uses these plans to guide their work. The team also checks progress against the plans during their weekly meetings. The quality manager, for example, reports on the current defect injection rates and yields for modules complete to date. He or she also provides feedback on the current product quality index, defect removal profile, and so forth (as shown in Table 1).

Figure 3: *Estimated Maximum Defects Found in a Unit Test for a Single BCR*

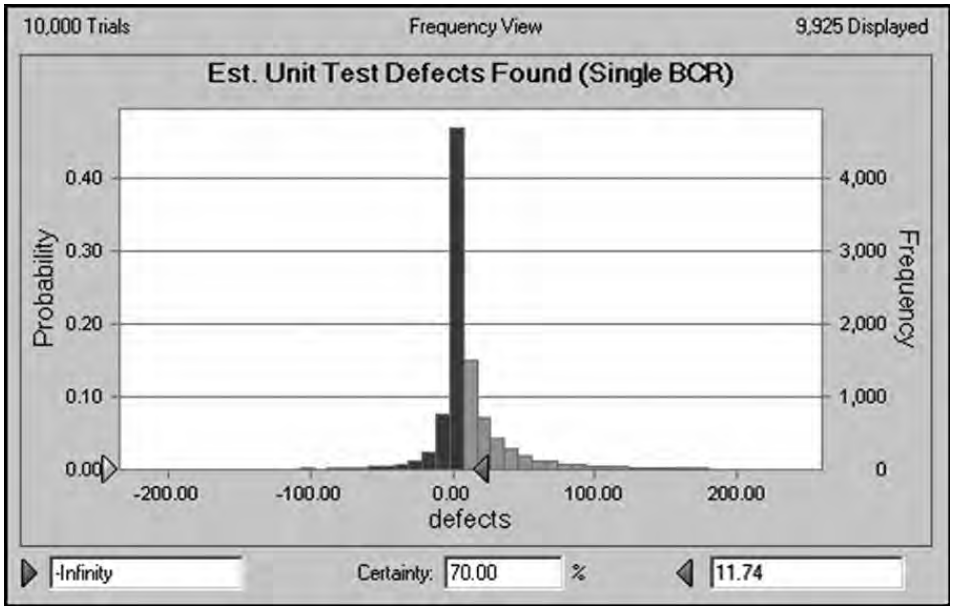


Table 4: *Sample TSP Quality Plan for a Single Update*

TSP Quality Plan with Monte Carlo (Single BCR)

Est. Time	Plan	Actual
High-Level Design	84	
High-Level Design Inspection	21	
Code	62	
Code Inspection	25	
Unit Test	14	

Phase Yields	Plan	Actual
High-Level Design Inspection	78%	
Code Inspection	70%	
Unit Test	95%	

Defects Injected per Hour	Plan	Actual
High-Level Design	0.11	
Code	0.39	

Est. Defects Found	Plan	Actual
High-Level Design Inspection	5.91	
Code Inspection	9.89	
Unit Test	5.57	

Project CPI	Plan	Actual
High-Level Design	1.22	
Code	1.73	
Unit Test	2.72	

Est. Defects Remaining After	Plan	Actual
High-Level Design Inspection	1.71	
Code Inspection	11.70	
Unit Test	6.13	

With the new Monte Carlo-generated quality plan, the quality manager has additional information to present at the weekly meetings. For example, he or she could present how many defects have actually been found in inspection or test activities—versus those predicted by the model. Another new metric is an updated estimate of the predicted defects remaining, easily calculated taking the estimates for defects injected and subtracting the estimates for defects removed. Once actual project quality data begins to come in, these models can be used again—this time replacing the estimated values with actual values and rerunning the simulation. This provides a new prediction for defects remaining that can be tracked throughout the project duration.

It is important to point out that this new way of examining and predicting the quality of the product in no way supplants those currently being used by TSP projects. This is simply one more weapon to add to the quality arsenal.

Summary

A TSP quality plan is a very effective way of focusing a team on the tracking and resolution of defects early in the project life cycle. However, the current version of the plan does not take into account variability. Applying Monte Carlo simulation to data already being collected by TSP teams provides a more robust insight into the quality processes TSP teams employ. It also gives further insight into what can be expected in terms of product and process quality. The TSP teams at Hill AFB recently started using this technique and are still gathering data on its usefulness.◆

References

1. Humphrey, Watts S. *TSP – Leading a Development Team*. Upper Saddle River, NJ: Addison-Wesley, 2006. Page 138.
2. Humphrey, Watts S. *TSP – Leading a Development Team*. Upper Saddle River, NJ: Addison-Wesley, 2006. Page 87.
3. Weisstein, Eric W. "Monte Carlo Method." *Wolfram MathWorld*. <http://mathworld.wolfram.com/MonteCarloMethod.html>.
4. Wittwer, J.W. "Monte Carlo Simulation Basics." *Vertex42*. 1 June 2004 <http://vertex42.com/ExcelArticles/mc/MonteCarloSimulation.html>.

Notes

1. The best resource to learn about TSP's numbered meetings and quality plans is Watts S. Humphrey's Nov. 2000 report "The Team Software Process." Section 7.1 discusses quality plans. See

<www.sei.cmu.edu/reports/00tr023.pdf>.

2. For more on this method, see <www.stsc.hill.af.mil/CrossTalk/2007/08/0708Schofield.html>.
3. Let us assume here that we determined hours earlier via Proxy-Based Estimation (PROBE) or other estimating model.
4. Don't be confused by the values you see in the shaded cells. Each of the highlighted cells for defects injected per hour, CPI, and yield in Table 2 initially contain an average value, similar to the current TSP quality plan; however, this value is replaced by the tool with random values from the distributions in Figure 2 when the Monte Carlo simulation is run.
5. In this situation, yield must be a decimal number between 0 and 1 instead of 0 and 100 percent.
6. The highlighted cells for "estimated defects found" and "estimated defects remaining after" in this table show the results of the parametric equations using the average values; these are replaced with the results of the calculations using random values from the distributions, during the Monte Carlo simulation.

About the Author

David R. Webb is a Technical Director for the 520th Software Maintenance Squadron of the 309th Software Maintenance Group at Hill AFB, Utah. Webb is a project management and process improvement specialist with 22 years of technical, program management, and process improvement experience in Air Force software. Webb is an SEI-authorized PSP instructor, a TSP launch coach, and has worked as an Air Force section chief, software engineering process group member, systems software engineer, and test engineer. He is a frequent contributor to technical journals and symposiums, and holds a bachelor's degree in electrical and computer engineering from Brigham Young University.

7278 4th ST
BLDG 100
Hill AFB, UT 84056
Phone: (801) 586-9330
E-mail: david.webb@hill.af.mil

Web Sites

The ITMPI's Fall Webinars and Conferences

www.itmpi.org/webinars

Now is a good time to sign up for the IT Metrics & Productivity Institute's (ITMPI's) free fall Webinars. Forthcoming Webinars include: ways to revolutionize the testing process with decision model; a "re-education" in basic and advanced software engineering principles; a "how-to" for organizations that want to become Agile; guidance on preparing an organizational training plan; increasing productivity through social networking; case studies from test assessments; guidelines on maintenance, support, and enhancement; and techniques, processes, and strategies to improve bad project planning. The ITMPI will also have all-day Webinars live from their Software Best Practices conferences in Baltimore (Sept. 14), Detroit (Sept. 28), Tallahassee (Oct. 7), Orlando (Oct. 13), Philadelphia (Oct. 21), and Rochester (Oct. 27).

Ahead in the Clouds

www.mitre.org/work/info_tech/cloud_computing

What are the essential components or capabilities necessary to create a private cloud computing environment? What can organizations do to facilitate the adoption of cloud computing to more effectively provide IT services? What is the most significant concern for federal organizations who want to use cloud computing? "Ahead in the Clouds" is the MITRE Corporation's public forum to provide federal government agencies with meaningful answers to common cloud computing questions like these, drawing from leading thinkers in the field. New questions are posed—and then industry experts chime-in with detailed responses.

Grady Booch Interviews Watts S. Humphrey

http://archive.computerhistory.org/resources/access/text/Oral_History/102702107.05.01.acc.pdf

With this issue's article and interview with Watts S. Humphrey—and CrossTalk's interview with Grady Booch appearing in our November/December 2010 edition—why not learn about what happened when the two legends met? The Computer History Museum sponsored this three-day (and eventually 184-page) oral history interview, by a developer of UML, of the man who developed the CMM, PSP, and TSP. Topics include his upbringing, formative years, time at Sylvania and Northeastern University, and his challenges in building a computer group. Also included is a thorough examination of the IBM years and, of course, his move to the SEI, discussing the CMM and CMMI and how his famed software processes took shape. Humphrey also talks about his family, and looks into the future of software.

Extending the TSP to Systems Engineering: Early Results from Team Process Integration

Anita Carleton
SEI

Del Kellogg and Jeff Schwalb
NAVAIR

A collaboration between the SEI and NAVAIR—Team Process Integration (TPI℠)—is currently underway. The TPI effort leverages the PSP and TSP research and body of practice. This article discusses the progress and performance through a pilot project with the AV-8B Systems Engineering team as well as others within NAVAIR that have utilized TPI in non-software domains. This article will share lessons and experiences with other industry/government organizations interested in applying the TSP in a non-software setting. The early results suggest some encouraging trends.

Since the emergence of software engineering in the 1960s, the size, pervasiveness, and complexity of software-intensive systems have increased by several orders of magnitude. The size of aircraft software systems in the 1960s approximated 1,000 lines of code while aircraft systems built in 2000 contained more than six million lines of code. The pervasiveness of software within aircraft systems has increased from controlling less than 10 percent of the functions the pilot performed in the 1960s to 80 percent in 2000 (as shown in Figure 1 on the following page).

We know that increases in software and system size contribute to increased complexity which, in turn, has contributed to pushing delivery and costs well beyond targeted schedules and budgets [1].

In a recent workshop conducted by the National Defense Industrial Association, the top issues relative to the acquisition and deployment of software-intensive systems were identified. Among them are:

- The impact of system requirements upon software is not consistently quantified and managed in development or sustainment.
- Fundamental systems engineering decisions are made without full participation of software engineering.
- Software life-cycle planning and management by acquirers and suppliers is ineffective.

So the biggest challenge is creating the right foundation: estimation, planning, development, and management practices as well as team processes, training, coaching, and operational support that will assist in a migration from buggy products and unnecessary rework (resulting in inflating development costs) to a proactive approach that builds integrated, quality software-intensive systems from requirements to field deployment.

℠ TPI is a service mark of Carnegie Mellon University.

Background

The SEI's TSP provides engineers with a structured framework for doing software engineering work. It includes scripts, forms, measures, standards, and tools that show software engineers how to use disciplined processes to plan, measure, and manage their work [2]. The principal motivator for the TSP is the conviction that engineering teams can do extraordinary work if they are properly formed, suitably trained, staffed with skilled members, and effectively coached and led.

> *"The principal motivator for the TSP is the conviction that engineering teams can do extraordinary work if they are properly formed, suitably trained, staffed with skilled members, and effectively coached and led."*

The TSP is already being used with great results on software teams [3]. A Microsoft study reported that by using the TSP, software teams cut schedule error from 10 to one percent. With its TSP teams, Intuit has increased by 50 percent the time that teams can spend in developing products during a typical year-long release cycle: Increased quality has dramatically cut the testing time required. An analysis of 20 projects in 13 organizations showed TSP teams averaged 0.06 defects per thousand lines of new or modified code. Approximately one-third of these projects were defect-free. Other studies show that TSP teams delivered their products an average of just six percent later than planned. This compares favorably with industry data showing that more than half of all software projects were more than 100 percent late—or were cancelled. These TSP teams also improved their productivity (size of developed code per hour of development time) by an average of 78 percent.

NAVAIR develops, acquires, and supports the aircraft and related weapons systems used by the U.S. Navy and Marine Corps. In recent years, interest in applying TSP to non-software domains has increased. The SEI TSP team has collaborated with NAVAIR to expand the TSP to teams that do other engineering along with software. These include areas such as systems engineering and integration, product integrity, CM/DM/QA (Configuration Management/Data Management/Quality Assurance), and process improvement itself.

NAVAIR already has a proven track record with the TSP and has demonstrated return on investment on their software projects [4, 5]. Table 1 (on the following page) shows TSP results from two NAVAIR programs: the AV-8B's Joint Mission Planning System (JMPS) program and the P-3C program. This result, due to the reduction in defect density, is a gross savings of $3,225,606 ($3,782,153 less the investment of $556,547). In turn, the ROI is derived from the cost savings compared to the cost of initially putting the TSP in place; in this case, it was a ratio of better than 7 to 1. Further, these organizations each reached CMM Level 4 in less

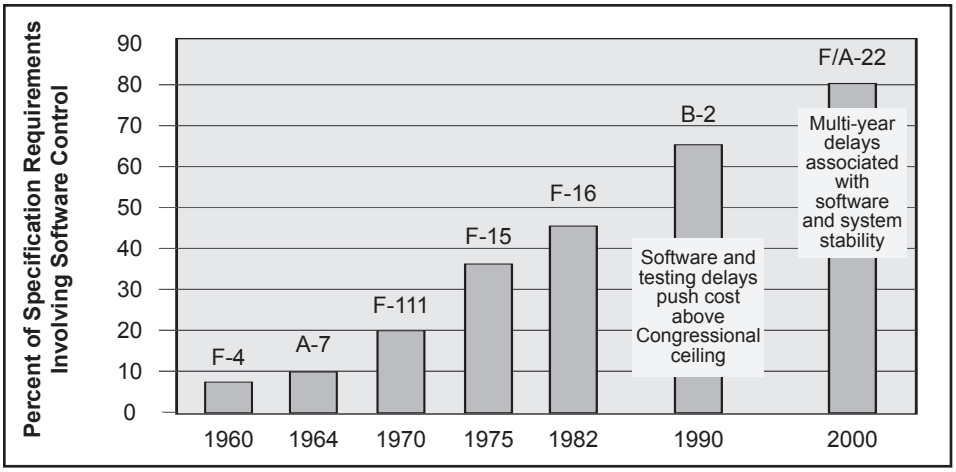


Figure 1: *Increasing Capabilities and Challenges of Software in DoD Systems* [1]

than 30 months—instead of the typical six years.

Very similar results occurred with other programs at that time, like with the E-2C aircraft program, also achieving CMM Level 4 in less than 30 months with their development teams using the TSP at the same time. Most recently (Jan. 2010), the H1 aircraft program worked less than 20 months to obtain a CMMI Level 3 rating while their development team used TSP to maintain aircraft software for the fleet.

The organizations referenced have standardized the TSP for all of their software development and maintenance work. These early adopters of the TSP are meeting their mission of producing higher quality products while maintaining significant cost savings. Their development teams now like using the TSP, saying of their staffs, "Once they have adopted it, they can't imagine working any other way." In all presented cases, the initial investment was returned in their first project and has then gone forward time and again to benefit the organizations for many years.

Results from these examples continue to inspire other NAVAIR System Support Activities (SSAs) to use the TSP. There are more than 20 additional NAVAIR SSAs now pursuing software process improvement activities. NAVAIR is seeing recurring savings and can now direct cost savings to the procurement of additional aircraft and weapons. In addition, NAVAIR used the TSP to accelerate CMMI improvement.

Starting TPI Efforts

Based on the demonstrated, measured success of software projects using the TSP in NAVAIR, other teams asked if they could apply the same processes to systems engineering and software/systems acquisition projects. As a result, NAVAIR has teamed with the SEI to expand the TSP framework to a technology called TPI. The SEI is also receiving additional requests to apply the TSP to non-software settings since it is becoming increasingly difficult to solve software problems without addressing systems engineering issues.

The NAVAIR/SEI collaboration entails testing the hypothesis that we can achieve the same kind of performance improvements applying TPI to systems engineering as we did applying the TSP to software projects, thereby improving management and communications in software-intensive systems and acquisitions. Our approach will entail conducting a series of pilot projects to determine if extending TSP practices to systems engineering results in measurable improvement. We will then use the results of this work to establish common processes for both systems and software engineering across the NAVAIR teams. Initially, the AV-8B Joint SSAs (developing the Harrier Aircraft) was selected as the systems engineering pilot program.

In kicking off these efforts, we realized that there were a number of research challenges that specifically had to be addressed. We extended the TSP practices to systems engineering by:

- Determining the baseline performance for systems engineering work at NAVAIR.
- Developing prototype processes/process definitions/scripts for systems engineering.
- Formulating relevant measures, especially size and quality measures pertinent to systems engineering.
- Building conviction and discipline in our leadership and team member training materials for teams that don't necessarily write software programs.
- Developing an extensible tool that allows for outlining any process, for collecting data unobtrusively, and for defining a measurement framework pertinent to any engineering domain.

Early results of applying TPI show some encouraging trends. The AV-8B Systems Engineering pilot project team is changing the way they do their work and is beginning to see some results similar to those realized by TSP teams. The AV-8B team is practicing more disciplined methods for planning and executing their work. They are meeting their missions and beginning to see some cost savings. In addition, the pilot team is inspiring other NAVAIR 4.0 SSAs to pursue process improvement [6].

Benefits

Through the pilot effort, we are seeing some of the following benefits:

Establishment of a Systems Engineering Baseline

We are beginning to establish a baseline for systems engineering performance at NAVAIR that can be used for estimating, planning, and tracking projects and programs:

- The requirements productivity rate varies between three and nine require-

Table 1: *TSP Results at NAVAIR*

Project	Defect Density Before TSP	Defect Density After TSP	Total Defects Before TSP	Total Defects After TSP	Average Cost to Fix	Product Size (KSLOC)	Cost Savings From Reduced Defects
AV-JMPS	1.13	0.59	501	261	$8,330	443.0	$1,992,663
P-3C	4.60	0.60	176	23	$8,432	38.3	$1,789,490
						Total Savings: $3,782,153	

ment statements per hour, depending on the complexity of the project[2].

- By just tracking requirements size growth, the team was able to decrease the rate of project size growth from 23.6 percent in the initial development cycle to 11.5 percent in the subsequent development cycle.
- By collecting the planned and actual requirements size and growth for the various components and the team productivity rate, the team builds up historical data that can be used on future projects.
- To quote one team leader: "Prior to TPI, we made estimates in a bubble. Now we are establishing and maintaining baselines for all of our releases, which allow us to make better estimates and more realistic plans and schedules."

Establishment of Planning Practices

Planning at the program and team level is now accomplished by holding multi-team launches that involve all of the teams implementing either the TSP or TPI. At first, they plan for no more than four months of work at a time so that their tasks can be detailed enough with fairly stable component sets. The component sets start to vary for a longer development duration so their plan would be less stable. This process is used by the AV-8B program to understand requirements from management, assemble plans, allocate work, and achieve commitment to plans from management and team members. The overall plan for the year and the next-phase plan are developed by the teams, work is allocated by the team, and the schedule is determined and committed to by team members.

Establishing Tracking Practices

For tracking purposes, work is broken down into small chunks that can easily be tracked (tasks are tracked at a granularity of less than 10 hours). Tracking only the task hours per week (planning for around 20) allows two or three tasks to be completed each week. Work is tracked daily by team members and discussed weekly in team meetings: Every team member knows how they are performing to their individual plan and the team plan. Monthly status reports are derived from the consolidated weekly reports by the team leader and presented to the integrated product team leads.

Twelve team members were able to achieve (on average) between 18 and 22 on-project task hours per week. The team performed well above the planned task hours: 15 per week in the first cycle.

The engineers embraced project planning and tracking. Each individual is able to track personal commitments to the team, enabling the team to better monitor commitments to the program. Tracking the work helped the team members with staying on-task, commenting that: "I need to stop doing *X* to get back on track. It is very easy to see the impact daily and weekly of not working to the plan."

Developing Standard Processes, Measures, and Tools

Standard processes, measures, terminology, and tools were developed and used by the AV-8B Program:

- The PSP-derived Excel spreadsheet and a process support technology Access-based tool were used for estimating, planning, and tracking work for team members and team leads.
- Team members identified, defined, and documented all systems engineering standard life-cycle processes in the tool. The team defined and developed an 18-step overall systems engineering process and a 482-step detailed systems engineering process.
- Through the defined processes, NAVAIR was able to maintain the consistency of processes across projects/programs. The defined processes also offered the ability to cross-train individuals. One integrated product team lead said: "We have a team concept across our program with all of the sub-teams (systems engineering, product integrity, software, test, lab, etc.). We also have a common set of processes and metrics to help all of the teams better communicate and address dependencies across the teams."

Performance Trends

With no historical data to go by, the team's initial plan identified a guess at a goal of less than 5 percent schedule slip and measured performance against the goal. The actual performance had an overrun of less than 10 percent. Now with some historical data, the team can set more realistic goals and try to continually improve on them. As far as cost and quality performance, size and effort estimates were within ±10 percent of what was planned, and there were no high-priority problem reports coming out of test.

Employee Work/Life Balance

TPI helped improve employee work/life balance. In order to get their job done before implementing TPI, employees routinely worked overtime. With TPI (and in

Coming Events

August 23-25

The 13th IASTED International Conference on Computers and Advanced Technology in Education

Maui, HI

www.iasted.org/conferences/home-709.html

September 13-16

Military Logistics Summit 2010

Washington, D.C.

www.militarylogisticssummit.com

September 13-17

PSQT 2010

Practical Software Quality and Testing

Minneapolis, MN

www.psqtconference.com/2010north

September 19-23

Oracle OpenWorld 2010

San Francisco, CA

www.oracle.com/us/openworld

September 26-October 1

STARWEST 2010

San Diego, CA

www.sqe.com/starwest

October 25-28

TechNet Asia-Pacific International Conference and Exposition 2010

Honolulu, HI

www.afcea.org/events/asiapacific

October 31-November 3

MILCOM 2010

San Jose, CA

www.milcom.org

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: <marek.steed.ctr@hill.af.mil>.**

order to get their 18-22 task hours per week), they did not have to work as much overtime. Overtime was decreased from being standard practice—sometimes 25 percent or more—to occasional overtime hours (less than 10 percent).

Customer Responsiveness

Customer responsiveness has improved to the fleet, the naval aviators, and to the internal program managers. The systems engineering team is able to more easily adapt to program and personnel changes. The pilots are beginning to provide input early on in the project—during the launch process—before the work has commenced (instead of providing feedback during the test phases). Program management feels that the TSP/TPI efforts are a success because the teams understand their work and the dependencies among all of the teams. The systems engineering team can also plan for a percentage of unplanned tasks to use their data to negotiate impact and trade-offs of unplanned work to planned work.

More Teams Doing TPI

We have since launched more non-software teams using the TPI approach. One of these is a mixed engineering team at Joint Munitions Effectiveness Matrix Weaponeering Systems that is applying the TPI to their non-software work as well as to their software team. This team has been using TPI for more than a year, has gone through four launch/relaunches, and has seen the types of benefits that the AV-8B team has seen. They also are seeing steady progress in making more accurate and precise estimates of their work, and have refined the triggers that would initiate an adjustment of their behavior so they stay on schedule.

Another example is the Precision Attack Weapon System Tactical Program Office, demonstrating the effectiveness of the TPI approach for one of their systems engineering teams. Their team has been using the TPI approach for more than a year and saw immediate benefits. During the initial launch, they developed a never-before-seen detailed plan that gave senior management the needed data to get additional project funding without having to *arm wrestle* the Program Manager, Air (PMA).

Then there is the P-3 lab team at the Patuxent River, Maryland Naval Air Station, who has been applying this approach to the many configurations of the lab setup they must provide. The P-3 team started applying TPI as an approach to the implementation phase of their Black Belt DMAIC (or Define, Measure, Analyze, Improve, and Control) project. Since starting about three years ago, the team has since provided two annual cycles of lab services and is halfway through their third. The P-3 lab team supports their customers by providing more than a dozen lab configurations across the PMA. This breaks into two basic types of support: usage in terms of running tests, and support in terms of configuring labs for those tests. Aggregate lab usage data shows a deviation of 12 percent less than planned while aggregate lab support data shows a deviation of .5 percent more than planned. While performance is impressive, deviation was at times greater when examined at the individual lab-customer level. As expected, this aggregate deviation demonstrates the advantage of estimating in smaller increments.

At the time of writing this article, several other process improvement efforts at NAVAIR are getting started with plans of applying TSP to their software teams and TPI to their non-software teams.

Summary

All engineering efforts must start with integrated teams. These teams must plan their work—and work to those plans—while collecting basic measures. They must then apply analyses to this data and derive metrics to determine their status on current work and, eventually, as a source for improving their planning capability on future work. From this approach, we have seen quality products and services delivered over and over with the potential for further improvement.

To make this happen, we have seen the need to put in place the TPI foundation of estimation and planning processes, team processes, development and management practices, effective and timely training, as well as launch, coaching, and operational support.

Projects that have adopted these methods have shown a dramatic increase in product quality and fidelity of schedule and effort estimates. The methods are supported by a doctrine that trains and sustains performance and quality improvement in an organization.

This article has shown what is possible when teams use TPI to establish this foundation to meet critical business needs. The end result is the delivery of

high quality systems, on cost, and with improved productivity.◆

Software Defense Application

Software defense organizations will benefit by learning about Team Process Integration, the continuing collaboration between the SEI and NAVAIR. As detailed in the article, results from current projects utilizing TPI show a gross savings of more than $3.7 million and a net savings of more than $3.2 million, with a return seven times the original investment. Quality improvement on two examined projects was a reduction in defect density from 1.1 to 0.59 defects per thousand LOC on one and 4.6 to 0.6 defects per thousand LOC on the other. TPI lowers costs, helps projects meet schedules, and improves productivity.

References

1. Walker, Ellen. "Tech Views – Challenges Dominate Our Future." *DACS Software Tech News*. Oct. 2007 <www.softwaretechnews.com/stn_view.php?stn_id=43&article_id=86>.
2. Humphrey, Watts S. *TSP: Leading a Development Team*. Upper Saddle River, NJ: Addison-Wesley Publishers, 2006.
3. Davis, Noopur, and Julia Mullaney. *The Team Software Process (TSP) in Practice: A Summary of Recent Results*. SEI, Carnegie Mellon University. Technical Report CMU/SEI-2003-TR-014. Sept. 2003 <www.sei.cmu.edu/reports/03tr014. pdf>.
4. Wall, Daniel S., James McHale, and Marsha Pomeroy-Huff. *Case Study: Accelerating Process Improvement by Integrating the TSP and CMMI*. SEI, Carnegie Mellon University. Special Report CMU/SEI-2005-SR-012. June 2007 <www.sei.cmu.edu/reports/07tr013.pdf>.
5. Saint-Amand, David. *Process Improvement at NAVAIR Using TSP and CMM*. Proc. of the 1st Annual TSP Symposium. San Diego: Sept. 2006.
6. Carleton, Anita, et al. *Extending Team Software Process (TSP) to Systems Engineering: A NAVAIR Experience Report*. SEI, Carnegie Mellon University. Technical Report CMU/SEI-2010-TR-008. Mar. 2010 <www.sei.cmu.edu/reports/10tr008.pdf>.

Notes

1. This graphic was created based on a table called "System Functionality Requiring Software," but the original creator of the table is debated: either PM Magazine or a U.S. Air Force "Bold Strike" Executive Software Course from 1992. To view the table, see: Ferguson, Jack. "Crouching Dragon, Hidden Software: Software in DoD Weapon Systems." IEEE Software July/Aug. (2001): 105-107.
2. For example, AV-8B uses Telelogic DOORS Objects to identify the number of requirement statements and, hence, the size of the requirement set. Any organization/program product can be viewed as a comparable proxy.

About the Authors

Anita Carleton is a senior member of the technical staff at the SEI, Carnegie Mellon University, where she has worked for more than 20 years on software process improvement, process measurement, and the TSP. She is the author of "Measuring the Software Process: Statistical Process Control for Software Process Improvement." Carleton has a degree in applied mathematics from Carnegie Mellon University and is a member of the IEEE Computer Society and the National Defense Industrial Association.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213-2612
Phone (412) 268-7718
Fax: (412) 268-5758
E-mail: adc@sei.cmu.edu

Del Kellogg is a PSP Certified Developer, PSP Certified Instructor, and TSP Authorized Coach for NAVAIR-China Lake. He has spent most of his 30 years at NAVAIR working on development of embedded software for the A-7E, AV-8B, and the AH-1W aircraft. He has applied the PSP and TSP for the last nine years within multiple NAVAIR teams. He is currently working in the Process Resource Team at NAVAIR. Kellogg's background is in computer science, physics, and math, and received his bachelor's degree in computer science from the University of Idaho.

NAVAIR Systems/Software
Support Center
1900 N Knox RD
BLDG 1494 (MS 6308)
China Lake, CA 93555
Phone: (760) 939-5494
Fax: (760) 939-0150
E-mail: delwyn.kellogg@navy.mil

Jeff Schwalb is employed by NAVAIR at China Lake, California, where he has been since 1984. He currently leads a NAVAIR enterprise team that helps provide continuous process improvement support across NAVAIR. Schwalb first became involved with process improvement in the 1990s using the SW-CMM, then becoming a certified PSP instructor and TSP coach. He has taught each of the TSP/PSP courses and has been involved in the TSP launch of several projects across NAVAIR. He is now working with the SEI to extend TSP practices into other domains. He received his bachelor's degree in computer science from California State University, Chico.

NAVAIR Systems/Software
Support Center
1900 N Knox RD
Building 1494 (MS 6308)
China Lake, CA 93555-6106
Phone: (760) 939-6226
Fax: (760) 939-0150
E-mail: jeff.schwalb@navy.mil

Building Critical Systems as a Cyborg

Greg Ball
cyborgg.com

In science fiction, a cyborg is a marriage of machine and human flesh. I'm not suggesting that you turn your favorite officer into an espresso machine, but this article explains the seemingly outrageous possibility that cybernetics may be the next step in the evolution of critical systems, demonstrates actual code and technology that is available, and describes real-world experiences in using it. The strength of this technology is in its resilience and adaptability in building complex critical systems that must face the real world. However, its use requires a shift in thinking about software—much like the introduction of "object-oriented" concepts once did.

Cybernetics is the study of communication and control processes, especially the comparison of these processes in biological and artificial systems. It attempts to learn principles that can be applied to any type of system regardless of its material realization. This kind of study began long before the existence of the modern digital computer. The term itself goes back to Plato.

Don't assume those early cyberneticists would be impressed by our modern high-availability computer systems. They might even view our conventional approach to software as fatally arrogant, requiring a programmer to anticipate everything.

Conventional software is based on the algorithmic approach pioneered by John von Neumann in the 1940s. An algorithm is just "a series of steps to achieve a desired aim" [1] that we then give to our machines to execute. It is a well-behaved approach with predictable results—so long as all of your assumptions are valid, your code is perfect, the world doesn't change, and your enemies are powerless to interfere.

I assume you've experienced what happens otherwise. The more critical a conventional system is, the more rigidly and exhaustively we must define those steps. We must also carefully control its runtime environment. According to the highest standards of compliance (e.g., DO-178B/ED-12B or MIL-STD-498) we must test every possible decision, every pathway, and every conceivable combination of data.

If certification is required, then the cost to produce the associated verification evidence grows exponentially with the size of the application. At some point, this is impossible—even in a modestly complex closed system. And in an open system, we can't even control the scope of the problem.

I sometimes wonder if, like an overprotective parent, our emphasis on rigor hasn't actually made our systems more vulnerable. Whenever our conventional systems encounter something other than the sterile environment that we intended, what sort of coping skills have we given them?

Von Neumann himself wrote about an alternative neural approach, one in which new behaviors can emerge in response to changes in the environment. This would fit the theoretical principles of our cyberneticists exactly, as they emphasize the use of feedback to accomplish goals rather than following a predetermined set of steps. While a neural or cybernetic approach is less well-behaved and less predictable than the software we are used to, it is also extremely adaptable and powerful.

Rather than spending too much time on a soapbox, I would rather present you with a question: Given the right tools, could you design a system that is safer and more economical to build because it has the ability to overcome its own imperfections and environmental obstacles and still complete the mission? Assuming that you are at least thinking about it, let's talk about how you might go about designing such a system.

What Is a Cyborg?

We want both kinds of the behavior that I've talked about, with predictable systems that follow established rules and procedures. But we also want them to adapt in the face of the unexpected. So it would seem that what we need is a hybrid approach: a combination of cybernetics technology with some other type of system. And that's a fairly good working definition of a cyborg. Fair, but not great; it is a bit like describing a car as "something with tires."

The original authoritative definition was published by Dr. Nathan S. Kline and Manfred Clynes in the September 1960 issue of the scientific journal Astronautics. And yes, they did suggest that the bodies of pilots could be modified for space travel using drugs and assorted parts (yikes, can't imagine why that wasn't popular). But those sensational examples were not part of the definition. Instead, they proposed a cybernetic principle that can be applied to any type of system. In their own words:

> What are some of the devices necessary for creating self-regulating man-machine systems? This self-regulation must function without the benefit of consciousness in order to cooperate with the body's own autonomous homeostatic controls. For the exogenously extended organizational complex functioning as an integrated homeostatic system unconsciously, we propose the term "Cyborg." [2]

Homeostatic is the idea of an open system that can regulate itself to function effectively in a broad range of conditions.

Open, as used here, refers to a system in which energy or material (resources) can be added or lost. It also means that the type and number of parts that make up the system are not static.

Exogenous in this context means any material that is present and active in an individual organism but that originated outside of that organism. It is meant to describe a cyborg's blended nature, where control is extended over other non-cybernetic parts.

A cyborg has the authority to unconsciously alter its operation. This language coincides with their example of the human autonomic nervous system. For example, you don't generally think about breathing. You *can* control it, but normally you concentrate on the mission while the body adjusts to your activities, environmental conditions, threats, etc.

A cyborg may alter its operation, but only to maintain a stable state or accomplish goals that we've set for it. Therefore, this definition both empowers and sets specific limits on the authority that is given to a cyborg.

One thing that the original definition does not explicitly mention is the concept of self—though you might infer that from the root words *cybernetic organism*: An organism is a separate distinct individual.

In my opinion, a cyborg must be able to distinguish *self* from any other organism,

or the environment, because it must attempt to regulate only itself.

It must not get confused and try to impose its goals on others. It must not attempt to change or take over the universe. It has to have a clear idea of which parts belong to it and which do not. It must have healthy boundaries to protect itself and play well with others.

Since we're now somewhat stretching the original definition of a cyborg, the term we propose to use is *cyborg gratia* or "for the sake of the cyborg." It means that the cybernetic organism is operating for itself as an independent organism inside a larger social structure.

Social governance is the final piece necessary to complete the concept. In an open system, you have to expect communication and cooperation, but also sometimes conflict between organisms. It is highly desirable to design a resilient system as an ecology of independent, cooperative, and adaptive organisms—one that can embody complex relationships with security and selective trust.

Such systems can align themselves with the changing and varied relationships between partners, alliances, and customers. They also enable a different paradigm for development and maintenance that embraces change and diversity.

Technology in Action

The question now is how to make that a reality. Cyborgg (pronounced "cyborg gee") is commercial open-source cybernetic technology in its second generation. It is impossible to describe everything in a short article—and difficult to know where to start. But I can show that working with a cyborg is not onerous.

Cyborgg employs a heterogeneous network of several neuron types to facilitate the integration of these cybernetic extensions into the rest of your system[1]. The data that they work with is not limited to numerical values[2]. They fall into two general classifications:

- Afferent (or sensory) neurons are used to receive input.
- Efferent neurons are used to manipulate or interact with the outside world.

One reason cyborgg was made open was to drive consensus on some basic terms and standards. For example: Just as the format of an e-mail address is important to everyone, so is standardizing the format of a Cyborg URI[3] (or CURI).

CURIs are a key mechanism for surgically implanting complex cybernetic components in conventional code, as shown in the following three examples:

EXAMPLE No. 1

Hooking into the neural net: We provide it with some feedback on system performance using a cyborg helper class.

```
// Define the neuron in question
CURI curi = new
CURI("curi://afferent/response
time$my service");

// You could obtain and use the neuron
directly
// but this helper class is convenient
TimeMarker marker =
TimeMarker.start(curi);

// Do something that you want to mea-
sure, then
marker.stopAndRecordTime();
```

EXAMPLE No. 2

Another indirect way of hooking into the neural net includes defining an attribute for a class that cyborgg will dynamically control. The following organelle shown is a convenient wrapper around a neuron that implies that this class is an organ. But where is the CURI? Cyborgg creates it behind the scenes by inspecting the rest of the class:

```
protected Organelle queueSize =
Organelle.newInteger("queue size",
10);
```

EXAMPLE No. 3

One can also select a cybernetic component or service. Note that many factors are in play here including failover, system load, authentication, biases, automated service discovery, etc. But these are handled invisibly by the cybernetic core:

```
// Asking for a certain type of service
with no filters or restrictions
CURI curi = new CURI("curi://$my ser-
vice");

// Cyborgg class used for dynamic depen-
dency injection
Injector injector  = new Injector();
// obtain the service
MyService service = injector.use(curi);
```

What happens when you do something like this? Under the hood, cyborgg complies with Aleksander's definition of a neural net as having a "network of adaptable nodes which, through a process of learning, store experiential knowledge and make it available for use" [3][4]. New system behaviors can and do arise from changes to the structure of this net.

Each cyborgg neuron has a complex internal structure that is more cell-like than the classical neural approaches[5]. This was done to overcome two barriers to general use.

The first barrier was the difficulty of understanding and having confidence in the decisions made by the net (analysis of a classical neural net is something only a researcher could love).

To address this, each cyborgg neuron contains a nucleus—a statistical model that captures information about the neuron's behavior (most of the information that a Six Sigma practitioner would ask for). Therefore, when a neuron fires, it can tell you plainly things like "with 80 percent certainty, these adjustments to the system are predicted to change the behavior of a certain aspect by 68 percent (plus or minus 10 percent)."

You can track the difference between the neuron's prediction and the actual result—and track the performance of the system in general. It keeps the cyborg from acting on weak or invalid assumptions. It is also used to discover new or unexpected correlations. This not only gives the cyborg power, but allows it to serve as a research tool.

The second barrier was the difficulty of training the net, of knowing what synapses to forge, and how the health of one neuron is related to the health of another. So each neuron contains a genetic algorithm or genotype that is used to grow and test new relationships. A pluggable axon allows the system to grapple with how it should respond[6]. Both of these are guided during the cyborg configuration.

This configuration includes the ability to define rules to shape, as well as extend or modify, the cyborg's behavior even after the system is deployed and running. It will obey broadcasted commands (using an extensible lexical command processor), including built-in diagnostic and test commands.

I have barely scratched the surface of just this one aspect, and there is no room to explain how service selection and failover takes place. There so much to describe about the social structure that is a central part of that decision—or the communication and other technology that supports it. Still, I've made clear what the purposes of a cyborg are and presented enough about the technology to encourage you to explore and test it for yourselves.

Lessons of Use

Cyborgg is currently in use supporting a number of medical facilities from cancer research to small practices. It is the technical foundation or glue inside a leading

health care software vendor's product, which brings together a distributed group of components from multiple vendors into a single enterprise services bus for health care. Its application has included electronic medical record services, disease management, clinical trials management, transcription, and document scanning.

Among the lessons learned in its use (so far):

- Configuration of a large distributed adaptable system can be problematic, which led to a redesign of the configuration subsystem in the latest version of cyborgg[7].
- The amount of benefit that you receive is closely related to the way that you modularize and package your system. Greater benefits come when applications are not monolithic.
- We found that good modular designs were sometimes negated by the deployment model. This led to the introduction of organs as an additional cyborgg concept, and Java Network Launch Protocol Replaceable Units as a supporting service or technology.
- The concepts are currently different and new enough to require good training of your development team. It particularly rewards a savvy architect that takes the time to learn its capabilities.
- Visibility is a key organizational success factor, underlining the importance of features to allow technical users to interact with a cyborg.

The strength of this technology is in building complex critical systems that defense organizations face in the real world. That the old combat phrase "no plan survives contact with the enemy" still holds true, as is the belief that any system that cannot adapt is likely to fail. Cyborg technology represents a controlled step defense organizations can take to be more adaptable—away from their stiff, pre-programmed conventional software and towards systems that have greater problem-solving skills.◆

References

1. von Neumann, John, Arthur Burks, and Herman H. Goldstine. *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*. U.S. Army Ordinance Department Report. June 1946.
2. Clynes, Manfred E., and Nathan S. Kline. "Cyborgs and Space." *Astronautics* (Sept. 1960): 26-27 and 74-75.
3. Aleksander, Igor, and Helen Morton. *An Introduction to Neural Computing*. London: International Thomson Computer, 1995.

Notes

1. For example, motor neurons work with threaded processes, failure analysis neurons are for failure analysis, and *germ layers* work with exogenous services in a service-oriented architecture.
2. Strings and other non-numeric data are converted to ordinals.
3. A Uniform Resource Identifier (URI) is like the familiar Uniform Resource Locator (URL) except that the resource it identifies does not necessarily specify location. A URL is a type of URI.
4. Igor Aleksander's work led to the development of the first computer based on neural principles to reach the marketplace.
5. For example, Boltzmann machines, Kohonen maps, and perceptrons.
6. Slipping into the math for a minute, the relationship between real-world neurons is likely not a simple linear relationship. Ask questions such as "Should it be quadratic?" and "Should we use a radial basis function?" The default axon uses a type of ragged cube that allows each neuron to fire according to a data-driven complex curve. However, the cyborgg application programming interface allows and encourages the researcher to substitute their own firing function and measure its effectiveness versus other approaches.
7. Changes included: automatic discovery of cyborgg-enabled services; the option to use configuration references where a commonly used set of goals or other parameters is defined only once, using a unique identification, and then referenced by other components; and with the addition of new installation and configuration services that allow the pushing of upgrades to remote customers.

About the Author

Greg Ball is a software and system architect with more than 15 years of government and private industry experience. He prefers a complex technical problem to most other forms of entertainment. Ball is the original creator of cyborgg.

7911 Woodstone LN
Dallas, TX 75248
E-mail: gball@cyborgg.com

Video Thrills the Radar Tsars

It happens to the best of us: the dreaded day our trusty computer leaves the surly bonds of electromagnetism to touch the face of neutrality—leaving the unprepared in several days of pure hell. Sure, it teases with a spark or sputter only to return to the blue screen of death, the black screen of doom, or the nauseous bios merry-go-round.

Such was my fate at the onset of 2010. It began with an unsolicited Windows upgrade that automatically downloaded and deemed itself so urgent the operating system incessantly begged for a restart and eventually took matters into its own hands. Luckily, I had the common sense to back-up my data to my new Christmas present—a 500GB Hitachi mobile hard drive that resembles an armadillo tank. Thanks, Santa!

My first step in Hades started with a "Gold Support" call. It sounded good when purchased, but what I didn't realize was that Gold Support starts with a call center operator on a mission to get you to fix your computer via telephone. Sure, I don't mind a few qualifying questions to eliminate boneheads ("yes, it's plugged in..."), but after that I want help; not computer repair on-the-job training. Note to computer manufacturers: How about "Platinum Support" that gets me back up and running without delay, no questions asked?

After Harold and Kumar failed to find my laptop's heartbeat, I took it to a local repair shop only to find out the remedy would outpace the cost of a replacement. The computer was limping through its fourth year of service and I was eager to jettison Vista (and its barnacles picked up over those years). It was time for a new computer. I had threatened to purchase a new computer over the past nine months; those threats rang hollow as I realized that necessity is not only the mother of invention but also the ugly stepsister of action.

I needed a computer quick—and a custom computer was going to take time. I don't know about you, but if I'm taking a computer into the trenches, I prefer it be tailored to my needs. So I was in need of quick makeshift computing while I found a long term solution. Enter my son's college computer (he's on hiatus in Oregon). With minimum processing power and disk space, I coupled it with my new tank drive to survive. It was enlightening to see what you can do with mobile storage and provisional computing.

Now I could take some time to find a suitable replacement. Along with Windows 7, I was interested in the new Intel Core i3, i5, and i7 processors—as well as their 32-nanometer submicron processing technology. While perusing the specs, something caught my eye. The i7 has floating-point processing capability.

Did you hear me? A readily available, commercially supported floating-point processor. To the ears of an embedded system designer that is like Charlie finding the golden ticket. Finally!

After years of neglect, has Intel decided to focus on military embedded systems? Hardly. It turns out Intel customers are attracted to floating-point to power a new generation of personal computers that handle high resolution graphics and high volume video.

Nowadays, it's not good enough to share pictures. Those pictures have to move in multiple windows, at the same time, and in high definition. We want "Avatar" in 3D THX surround sound streaming on our laptop with pop-up director notes devoid of jitter, loading stutter, or delay. That's where floating-point comes in.

While The Buggles decried, "Video Killed the Radio Star," it turns out in this millennium, video has an unlikely partner: the military. Not a partner in murder, rather a partner in readily available commercial off-the-shelf floating-point processors.

Video is an unintentional accessory to the art of war. Yes, your desire to stream your baby's first steps to grandma and grandpa has military embedded computer designers dreaming of a commercial processor crunching complex fast moving radar, sonar, and electronic warfare data by day and handling routine data parsing, links, stores management, and fault tolerant checks by night.

Sure, standalone digital signal processing (DSP) chips have been in operation for years; however, as standalone chips, they command their own real-estate on densely populated circuit boards. That gives rise to high rent, heavy footprints, and congestion. Now designers can use a processor that performs DSP as well as general-purpose processing on a single chip, shrinking substantial processing capability into a smaller space.

Couple size benefits with the cost savings from commercial processor mass manufacturing, and you have embedded system designers squirming with joy like Iggy Pop sucking an extra sour Warhead.

So I had a choice: i3, i5, or i7. Did I really need the i7? No. Did I want the i7? Yes. Did I get the i7? Of course—as well as a couple of 500GB hard drives in RAID configuration with a boatload of RAM.

Now I need a good lead on a miniature radar antenna—about five to six inches in diameter with a SCSI output. I'll plug that into my new laptop and start painting targets in the office by day and research why Cher is morphing into Joey Ramone by night. All because Dave Cook wants to watch movies on his laptop[1].

Thanks Dave.

—Gary A. Petersen
Arrowpoint Solutions, Inc.
gpetersen@arrowpoint.us

Note

1. Although Dave's always talking about his laptop in his BACKTALK, see <www.stsc.hill.af.mil/crosstalk/2008/12/0812BackTalk.html>.

Can You BACKTALK?

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

For more information on how to submit your BACKTALK article, go to <www.stsc.hill.af.mil>.

CrossTalk / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

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

CrossTalk thanks the above organizations for providing their support.