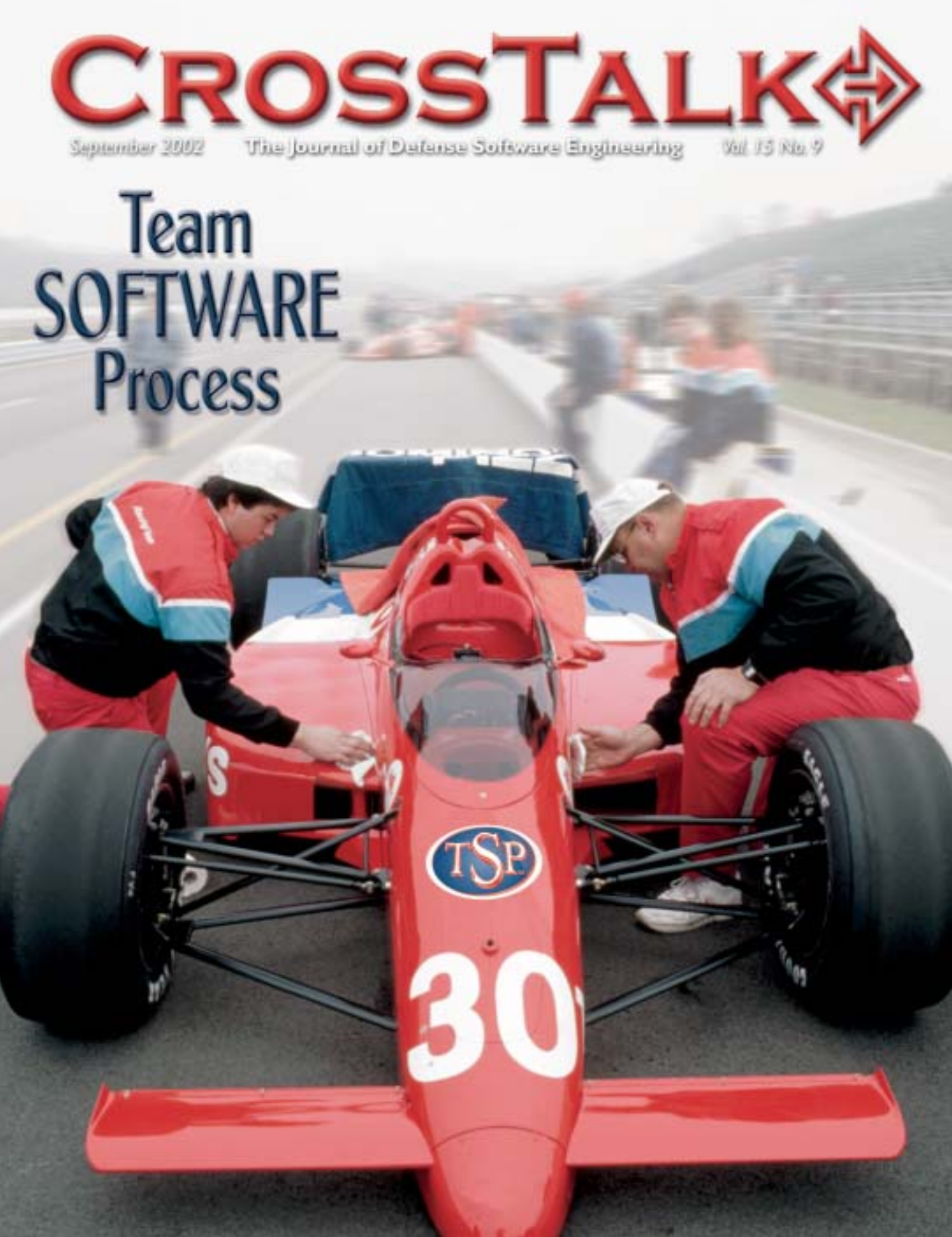
CrossTalk
September 2002 The Journal of Defense Software Engineering Vol 15 No 9

Team SOFTWARE Process



Policies, News, and Updates

Team Software Process

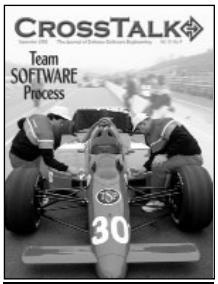
Best Practices

Software Engineering Technology

Open Forum

Online Articles

Departments



ON THE COVER

Cover Design by Janna Kay Jensen. Photo © Neil Rabinowitz/ CORBIS.

CROSSTALK

SPONSOR	*Lt. Col. Glenn A. Palmer*
PUBLISHER	*Tracy Stauder*
ASSOCIATE PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Pamela Bowers*
ASSOCIATE EDITOR	*Chelene Fortier*
ARTICLE COORDINATOR	*Nicole Kentta*
CREATIVE SERVICES COORDINATOR	*Janna Kay Jensen*
PHONE	(801) 586-0095
FAX	(801) 777-8069
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/ crosstalk
CRSIP ONLINE	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 28.

Ogden ALC/MASE
7278 Fourth St.
Hill AFB, UT 84056-5205

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

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

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

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

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

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

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

The TSP Builds Teams and Successful Software

The Team Software Process℠ (TSP℠) was developed by the Software Engineering Institute to help integrated teams plan and track their work while developing and enhancing software intensive systems. While the general concepts of effective teamwork are well known, engineering teams do not generally get guidance on how to build cohesive teams or on how to effectively lead and coach such teams. The TSP builds and guides "self-directed" teams. These are teams that manage their own work, build quality products, and consistently meet commitments. As the articles in this issue show, such teams provide a rewarding working environment and are a sound business investment.

By establishing effective team-working practices, organizations also build mature management practices. The TSP provides specific guidance on how to most effectively implement many of the high-maturity practices specified by the Capability Maturity Model® (CMM®). As this month's authors point out, implementing the TSP accelerates an organization's CMM improvement program.

In the first theme article, *AV-8B's Experience Using the TSP to Accelerate SW-CMM Adoption*, Dr. Bill Hefley, Jeff Schwalb, and Lisa Pracchia describe how using the TSP accelerated their CMM improvement efforts. The first step to CMM Level 2 is the most difficult and is where teams often need the most guidance. As this article shows, the guidance provided by the TSP cut the typical time to reach Level 2 by about 40 percent.

Next in *How the TSP Impacts the Top Line,* Robert Musson examines the management consequences of using the TSP. He addresses the cost benefit trade-offs management must make in running a software-intensive business and explains how the TSP impacts these trade-offs. He shows that organizations can cut development costs by 25 percent and reduce total life-cycle costs by 60 percent.

Two theme articles discuss the benefits of using the TSP. In *All the Right Behavior*, David R. Webb describes how the TSP helps teams determine their status and manage their work. When projects cannot precisely measure their status, they cannot recognize their schedule problems in time to economically resolve them. The earned value tracking provided by the TSP enables development groups to recognize their problems in time to effectively address them.

The final theme article, *Managing a Company Using TSP Techniques* by Dr. Carlos Montes de Oca and Dr. Miguel A. Serrano, describes how they used the TSP to manage a software company. As they point out, "The TSP is a powerful team process that can be customized to manage the performance of teams beyond the software domain." While the TSP was originally developed to support integrated engineering teams, this article indicates the broad applicability of the TSP's team-building and team-working methods.

Two supporting articles in this issue deal with the costs and benefits of process improvement. In *SEI CMM Level 5: Lightning Strikes Twice*, Gregory P. Fulton describes three management elements required for rapid and effective process improvement. Even after organizations reach high maturity levels, process improvement requires management support that is largely independent of the technology used. Such support should help any group, whether using the TSP or not. Jim McHale addresses the principal costs of TSP introduction and use in his article *TSP: Process Costs and Benefits*.

Dr. Ioana Rus, Dr. Carolyn Seaman, Dr. Mikael Lindvall, Dr. Victor Basili, and Dr. Barry Boehm introduce a new Web-based repository of lessons learned in *A Web Repository of Lessons Learned from COTS-Based Software Development*. The new system will allow dialogues between users and experts that provide concrete support for problems.

This issue wraps up with two online articles. Noopur Davis' *Using the TSP to Implement the CMM* shows how TSP methods map to the CMM framework. Paul J. Solomon's article, *From Performance-Based Earned Value to the CMMI*, explains that earned value management can be a process thread to enable effective process integration and improvement during transition to the Capability Maturity Model® Integration℠.

Finally, a very important announcement is in this issue: Dr. Nancy Spruill of the Office of the Under Secretary of Defense announces CrossTalk's 2002 U.S. Government's Top 5 Quality Software Projects competition. If your project qualifies, I urge you to submit a nomination. I served on last year's Top 5 review committee and will also be a judge this year. Whether or not your project wins a coveted Top 5 slot, you will learn a great deal just from preparing a nomination.

Watts S. Humphrey
Fellow, Software Engineering Institute at Carnegie Mellon University

OFFICE OF THE UNDER SECRETARY OF DEFENSE

3000 DEFENSE PENTAGON
WASHINGTON, DC 20301-3000

ACQUISITION AND TECHNOLOGY

1 JUL 2002

MEMORANDUM FOR ALL GOVERNMENT SOFTWARE PROJECTS

SUBJECT 2002 U.S. Government's Top 5 Quality Software Projects

As the Department of Defense Executive Agent for Software Intensive Systems, I am pleased to announce that *CrossTalk, The Journal of Defense Software Engineering*, will be dedicating its July 2003 issue to the 2002 U.S. Government's Top 5 Quality Software Projects. The aim is to recognize outstanding performance of software teams and to promote best practices. The *CrossTalk* staff is accepting nominations for this honor through 13 December 2002.

To be eligible for this award, projects must have been performed under a government contract (internal government contracts also eligible) and provided a software deliverable to the customer during the period of June 2001 through December 2002 (previous Top 5 winning projects are not eligible). The deliverable may be a completed contract or an incremental deliverable.

The selection of the Top 5 software projects will be based upon the project's adherence to quality and to the contracted cost, schedule, and requirements. The specific nomination criteria, process, and form can be acquired from the *CrossTalk* Web site located at <www.stsc.hill.af.mil/CrossTalk>.

The Top 5 winners will be announced in *CrossTalk's* May 2003 issue and will receive awards at the Software Technology Conference (STC) 2003. Recognition of the Top 5 software projects helps promote best practices that have demonstrated results.

Nancy Spruill

Dr. Nancy Spruill
Director, Acquisition Resources and Analysis
Office of the Under Secretary of Defense
(Acquisition, Technology and Logistics)

AV-8B's Experience Using the TSP to Accelerate SW-CMM Adoption

Dr. Bill Hefley
Carnegie Mellon University

Jeff Schwalb and Lisa Pracchia
Naval Air Systems Command, China Lake, Calif.

The Personal Software Process℠/Team Software Process℠ (PSP℠/TSP℠) provides a framework for disciplined software engineers to successfully execute software development projects. This article describes synergies identified in a TSP implementation that have accelerated organizational software process improvement – resulting in attaining a Capability Maturity Model® for Software Level 2 over more than 40 percent faster than the average duration reported by the Software Engineering Institute (SEI).

This article describes an organization's experience successfully applying both the Software Capability Maturity Model® (SW-CMM®) and Personal Software Process℠/Team Software Process℠ (PSP℠/TSP℠). While work at the Software Engineering Institute (SEI) has shown how the TSP relates to the SW-CMM [1, 2], this article describes results from a CMM-based Appraisal for Internal Process Improvement (CBA-IPI) in a software development and maintenance organization.

These results show how the use of the TSP has accelerated the SW-CMM adoption. Feedback from engineers and the Systems/Software Engineering Process Group (SSEPG) on these results provide additional insight, and the current status of the TSP project is described. Continuous learning is also addressed as an organization applies its TSP experience to the rest of the organization as well as to improving the TSP, itself.

The organization in this success story is the AV-8B Joint System Support Activity (JSSA), located at China Lake, Calif. This Naval Air Systems Command (NAVAIR) team provides software support for the AV-8B Harrier aircraft for the United States Marine Corps and its allies, Spain and Italy. The AV-8B JSSA is a progressive organization in terms of ongoing process improvement initiatives. These efforts paid off when the combined benefits of the PSP, the TSP, and a robust Earned Value Management System (EVMS) were aimed at demonstrating a SW-CMM Level 2 process maturity.

While all of these factors were equally important, this article focuses on how the PSP/TSP successfully prepared the AV-8B JSSA for that Level 2 appraisal in an accelerated 14 months. This accelerated pace supports an earlier report by Boeing that showed that the PSP/TSP helped them reduce their time advancing from SW-CMM Level 3 to Level 4 by 33 percent [3], and is significantly shorter than the average time to move to Level 2 reported by the SEI [4].

Maturity Level 2 Impacts of TSP Realized

Requirements Management Key Process Area (KPA)

- The Team Software Process℠ (TSP℠) scripts (processes) direct software engineers to review allocated requirements provided by project system engineers before incorporating them into software activities.
- The TSP launch scripts supported software engineers in using allocated requirements as a basis for software planning, work products, and activities.

Software Project Planning KPA

- The Personal Software Process℠ (PSP℠)/TSP PROxy-Based Estimation (PROBE) method was used to estimate the size of software work products by combining guidance from the TSP scripts, historical data, and professional judgment.
- Software planning data were recorded in the TSP notebooks for project tracking and future planning. This included size, time, resource, and quality estimates, both individually and for the project team.
- The TSP launch processes and checklists made project planning more robust by incorporating the project's software development plan, quality assurance plan, configuration management plan, and the organization's earned value management system requirements.
- The TSP launch processes required the team to identify and assess programmatic and technical risks, and to create a risk mitigation plan.
- The TSP processes helped to facilitate more effective communication between the software, systems, and test integration teams in performing project planning throughout the build process.

Software Project Tracking and Oversight KPA

- The actual size of work products was recorded and tracked against estimates in the PSP/TSP workbooks.
- Project's effort, cost and technical activities were tracked in the TSP weekly team meetings through rollup of individual workbooks.
- The TSP re-launches provided additional periodic reviews of the project.
- The TSP data enabled corrective actions to be taken as necessary, based upon cost, effort, size, risks, and task assignments.
- The TSP team workbook clearly showed task assignments for all work products and activities.

Software Quality Assurance KPA

- Representatives from the Quality Assurance (QA) group had started to attend some of the weekly reviews to increase their understanding of the TSP processes.
- The QA group's organizational role in weekly management meetings expanded to report on the TSP project's use of their newly acquired processes.

Software Configuration Management KPA

- The project's Configuration Management (CM) plan was prepared in accordance with the TSP and the organization's CM and data management processes.
- The TSP scripts provided guidance on CM activities for the software engineers.
- Software work products placed under CM were identified in the CM plan and the TSP scripts, and the TSP process was used to track changes and problem reports.

Getting the PSP and the TSP in Place

The AV-8B JSSA's TSP project team consisted of seven software engineers, three systems engineers, and one test engineer. Before the appraisal, the project received PSP training during the fall of 2000. This

® Capability Maturity Model, CMM, Capability Maturity Model for Software, and SW-CMM are registered in the U.S. Patent and Trademark Office.
℠ Team Software Process, TSP, Personal Software Process, and PSP are service marks of Carnegie Mellon University.

Maturity Level 3 Impacts of TSP Realized

Organization Process Definition and Organization Process Focus Key Process Areas (KPA)

- Software engineering and other related groups received orientation on process improvement activities and their roles and responsibilities in the Personal Software Process℠ and Team Software Process℠ (PSP℠/TSP℠).
- Needed project processes were identified with the TSP process inventory launch step and process performance data was beginning to be collected.
- The organization's approved life cycles had started to incorporate the PSP/TSP.
- A process improvement proposal repository was being created and the project was conducting post-mortem sessions at the end of each build cycle.

Training Program KPA

- The project's training plan included the PSP/TSP.
- The TSP launch training provided an orientation to the applicable roles required to both launch the project and operate on a weekly basis.

Integrated Software Management KPA

- The TSP launch scripts required the team create a process inventory that identify any weak or missing processes needed to deliver their product.
- The project was managed according to the TSP scripts and used the TSP data (metrics) for software planning and estimating.
- Project cost and effort was managed in the organization's Earned Value Management System (EVMS) according to the TSP launch/re-launch handbooks. Detailed TSP data from the individual engineer level clearly fed into the EVMS system and provided insight to better understand EVMS data.
- Software risks (programmatic and technical) were identified, assessed, and tracked according to TSP risk scripts.

Software Product Engineering KPA

- The TSP software engineering tasks were clearly understood, integrated, consistently performed, and measured.
- Project TSP plan workbooks were used by all project personnel to plan and track their work efforts.
- Tools such as object oriented analysis and object oriented design, the PSP/TSP scripts and checklists, and EVMS were integrated into the project's process.
- Defects were collected by each member of the project team during every phase of development and then analyzed by the team at post-mortem.
- The TSP performance measures collected were then used to determine status on project phases and to improve the process for the future project plans.
- The TSP processes helped the software, systems, test integration and project management teams work more closely together on project planning throughout the software life cycle.

Peer Review KPA

- Peer reviews were planned and documented in each engineer's TSP project plan workbook.
- Peer reviews were performed in accordance with the TSP scripts and the organization's peer review process.
- Engineers, in both individual and peer reviews, used the TSP scripts and the launch workshop notebook to collect defect data throughout the project.

training consisted of an executive workshop (one day), management and support staff training (two days each), and software engineer training (14 days).

A TSP launch was then conducted in January 2001 where the project was defined to have five build cycles. As a result, the launch has been followed by supporting re-launches in April 2001, December 2001, February 2002, and most recently June 2002.

The Appraisal

The CBA-IPI appraisal of the AV-8B JSSA organization took place in May 2001. Two AV-8B JSSA software projects were examined during this appraisal – the TSP project and a non-TSP project of similar size. The appraisal focused primarily on SW-CMM Level 2 and Level 3 goals. To better understand the impact of the TSP, a decision was made by the appraisal team to also *listen* for Level 4 and Level 5 evidence as it related to TSP. These observations would help determine which SW-CMM Key Process Areas (KPAs) were influenced by the use of the TSP on the project and to what extent.

The Maturity Level 2 Impacts of TSP Realized sidebar (see page 5) and Maturity Level 3 Impacts of TSP Realized sidebar (above) outline the TSP-related observations for SW-CMM Level 2 and Level 3 KPAs, as noted during a CBA-IPI appraisal of the AV-8B JSSA.

These TSP observations represent a solid process foundation upon which other capabilities can be effectively built. In addition to the Level 2 and Level 3 evidence, the appraisal team also noted a number of higher-maturity observations at Maturity Levels 4 and 5. These are shown in the sidebar, High Maturity Impacts of TSP Noted During the Appraisal.

Feedback from the Engineers

The TSP-trained software engineers at the AV-8B JSSA enthusiastically admit that TSP improves the way the organization plans, schedules, and tracks work, and also provides a strong emphasis on higher quality software. This statement is even true of those who initially doubted the TSP as just another management fad.

These statements summarize their TSP sentiments:

- The TSP tracks the quality of what we are producing, not just the time we produce it in.
- You can look at what you did previously.
- You really see how good a product is.
- It is easy to track all the information.
- Once the PSP/TSP becomes your process, it is relatively effortless.

Comments from select TSP project team members are worth repeating. "It is the future, I am sold on it," adds the software team lead. The software lead goes on to explain how TSP benefits both the organization and the individual. "People with TSP/PSP training can go from project to project. They are much more versatile. When the organization as a whole does better work, you don't need a superstar to pull it along."

The TSP project's lead software design engineer has become one of the organization's strongest advocates of PSP/TSP. "PSP really sells you on finding defects early in the process. It really does make a difference at the end. We thought it wasn't going to work. But we all became converts by producing valuable data all along the way. We also significantly improved productivity. I worried because I have seen too many people more interested in the process than in the product. But TSP keeps you focused entirely on the project as you finish smaller products at regular intervals."

He also talked about the importance of software design in the PSP/TSP. "You have got to have good design to get good code. One advantage of doing design in TSP is the design review process. These reviews help you find and fix potentially costly defects much sooner."

Another project software lead views PSP/TSP like this: "The whole idea of keeping historical data is to make the

product cost less ... to make better estimates for future work. A big benefit of PSP/TSP is that you can document what changes will cost."

The leads of both the TSP and non-TSP projects shared the same viewpoint. "In the end, what this is really about is people. No matter what you invest in terms of training and overhead, what you are really investing in is people. And the important thing is that we improve what we are doing."

Feedback from the SSEPG

"Including a TSP and non-TSP project in the same appraisal was very insightful," said the lead of the AV-8B JSSA's SSEPG. "While both projects had the necessary process evidence, finding and understanding the TSP project evidence was effortless. Three-quarters of the SW-CMM requirements for Level 2 were automatically satisfied simply by the project following TSP." This SSEPG lead is another TSP convert at AV-8B, along with her appraisal teammates.

TSP Project Evolution

The second build cycle delivered a testable product with some functionality. At their last re-launch, this seven-person TSP project had completed a 41-week development effort in 45 weeks, or within 10 percent of their original estimate. The defect density of the product at system integration test was 2.1 defects/thousand lines of code. Perhaps the most significant quality-related observations have come from the systems and test engineers. They are astounded by the robustness of the application and its 100 percent up time. For them, this is a first.

After that the project continued with planning their work and working their plans with the TSP. After the May 2001 appraisal, the project conducted two additional re-launches associated with the third build cycle of their product.

The first of these was in December 2001; it planned the project through to the following June 2002. Of particular interest was the fact that during this re-launch another mini re-launch was planned. This was because the team felt a mid-course correction would be needed due to the fact that new work with no historical basis for planning was starting up. This meant that it would be in their best interest to stop and re-plan. This re-launch was conducted in February 2002 and was very effective in quickly allowing the team to apply recently gathered metrics and re-plan accordingly for the rest of the third build cycle.

The project is now underway after its most recent launch, conducted in June 2002, that planned the fourth build cycle.

Filling in the Organizational Gaps and Overlaps

The TSP and the SW-CMM are complementary by design. However, since the TSP concentrates on project issues, it does not address the broader organizational aspects of the SW-CMM. Even if all teams in an organization were using the TSP, there is still the need for an additional thin layer of organizational support. That organizational support is more obvious at SW-CMM Level 3, where projects are expected to use a common set of documented and approved management and engineering processes. In addition to the gaps, there are also overlaps between the TSP and the SW-CMM to consider.

Both the gaps and the overlaps have challenged the AV-8B JSSA. Filling the biggest gap meant creating a developmental change control board, which the TSP assumes is in place organizationally. For many lower-maturity organizations, creating this board may be a TSP project effort that the organization can then adopt as a standard approach. That was the case at AV-8B.

The second challenge was an overlap between the TSP project roles and organizational roles. Both the TSP project and the organization have duplicate roles that are responsible for processes, configuration management, and quality assurance. Negotiating the roles, responsibilities, and functional touch-points for these duplicate sets of roles takes time, effort, and patience.

The good news is that the AV-8B JSSA is a stronger, more effective organization for filling the gaps and eliminating the overlaps. NAVAIR's lead TSP coach will also be able to further leverage this experience by sharing AV-8B's lessons learned with other NAVAIR TSP projects.

Conclusions

The TSP launches and executes projects with individuals trained in the PSP. These teams follow standards contained in a disciplined, automated process framework. It is important to understand that the PSP and TSP frameworks are flexible and should be evolved based on the team and organization's needs. The primary vehicle for this evolution is the process improvement proposal – a fundamental element of the TSP.

The AV-B JSSA is on its own unique evolutionary TSP path. It is plowing new ground by integrating its TSP tools with its EVMS, which is one of only two EVMS systems in NAVAIR currently certified. It is also reshaping the TSP for application to its maintenance software projects. With strong support for the TSP coming from all levels of the organization and results that speak for themselves, the AV-8B JSSA feels their process improvement initiatives will continue on an accelerated course.◆

References

1. Davis, N. "Using The TSP to Implement The CMM." CrossTalk Sept. 2002: 30.
2. Davis, N., and J. McHale. Relating the Team Software Process to the Capability Maturity Model for Software (CMU/SEI-2002-TR-008). Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2002.
3. Vu, J.D. "Process Improvement Journey (From Level 1 to Level 5)." European Software Engineering Process Group, 2001.
4. Software Engineering Institute. Process Maturity Profile of the Software Community – 2001 Year End Update. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 2002.

Additional Reading

1. Humphrey, Watts S. Introduction to the Team Software Process℠. Boston: Addison-Wesley, 1999.

High Maturity Impacts of TSP Noted During the Appraisal

Maturity Level 4 and 5 Key Process Areas

- The Team Software Process℠ (TSP℠) is beginning to provide the organization a foundation for software quality management activities.
- Quality goals are being established using TSP launch scripts and defects are being tracked using the Personal Software Process℠ and the TSP.
- The TSP is helping to establish organizational defect prevention activities.
- The TSP collects and consolidates defect data for current and future use, which supports awareness by this project, and use of defect data by other projects.
- Data to support continuous process improvement are becoming available through TSP by capturing and acting on process improvement proposals. TSP post-mortems are scheduled and held to identify improvement opportunities.

About the Authors

Bill Hefley, Ph.D., is a senior lecturer at Carnegie Mellon University. He is a lead assessor for the Capability Maturity Model® (CMM®)-based Appraisal for Internal Process Improvement, Standard CMMI℠ Assessment Method for Process Improvement, and People Capability Maturity Model methods. Hefley is co-author of "People CMM" and "People CMM-based Assessment Method." He was instrumental in launching the Software Engineering Institute's Software Process Improvement efforts.

IT Services Qualification Center
Institute for Software Research Int'l
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
Phone: (412) 268-4576
E-mail: bill.hefley@cs.cmu.edu

Jeff Schwalb has been employed by the Naval Air Systems Command at China Lake, Calif. since 1984. He spent the first 10 years of his career developing various embedded real-time range instrumentation systems. Schwalb became a Personal Software Process℠ instructor in 1995 and has taught the course more than 10 times. He has also been involved in the Team Software Process℠ launch of five projects. Schwalb has a degree in computer science from California State University, Chico.

NAWCWD
Code 4.1 Systems Engineering Dept.
Knox Rd, Bldg. 1494
China Lake, CA 93555-6100
Phone: (760) 939-6226
Fax: (760) 939-0150
E-mail: schwalbj@navair.navy.mil

Lisa Pracchia leads software process improvement initiatives at the AV-8B JSSA. Her software background consists of process improvement, business analysis, project management, product life cycle management, and product marketing in a wide range of industries (discrete product manufacturing, international publishing, telecommunications, and weapons systems development/support for both the government and private industry). Pracchia has a master's degree in management from the University of Redlands in California.

NAWCWD
41K200D (Pracchia)
Bldg. 20000A, Room 302
1 Administration Circle
China Lake, CA 93555
Phone: (760) 939-2188
DSN: 431-2188
E-mail: pracchialm@navair.navy.mil

WEB SITES

Team Software Process: Software Engineering Institute

www.sei.cmu.edu/tsp

To have a high-performance software organization you must have high-performance teams, staffed with high-performance software engineers. The Personal Software Process℠ (PSP℠) and the Team Software Process℠ (TSP℠) provide a road map for organizations and individuals to follow on the road to high performance. The TSP provides specific guidance about how PSP-trained engineers can work as effective team members on a high-performance team. The PSP provides specific guidance on how individual engineers can continually improve their performance.

Software Productivity Consortium

www.software.org

The Software Productivity Consortium provides reduced-to-practice technology for the development of systems and software, and is a vehicle for members and affiliates to adopt, implement, and improve their processes, methods, and technologies for developing software-intensive systems. Its structure as a consortium fosters a collaboration to leverage pooled resources among members, share lessons-learned, and develop targeted technologies that meet fundamental and common needs of all members, and are experienced with a multitude of process and framework models.

Earned Value Basics

www.acq.osd.mil/pm/evbasics.htm

Earned value is a management technique that relates resource planning to schedules and to technical cost and schedule requirements. All work is planned, budgeted, and scheduled in time-phased "planned value" increments constituting a cost and schedule measurement baseline. There are two major objectives of an earned value system: to encourage contractors to use effective internal cost and schedule management control systems, and to permit the customer to be able to rely on timely data produced by those systems for determining product-oriented contract status. The benefits to project management of the earned value approach come from the disciplined planning conducted and the availability of metrics, which show real variances from plan in order to generate necessary corrective actions.

Software Technology Support Center

www.stsc.hill.af.mil

The Software Technology Support Center is an Air Force organization established to help other U.S. government organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

How the TSP Impacts the Top Line

Robert Musson
Software Engineering Institute

With the Team Software Process℠ (TSP℠), developers are reporting significant productivity improvements. But what does this mean to the profitability of the corporation? How do these productivity improvements affect the balance sheet? This article compares the development costs associated with teams in a traditional test-based organization to those costs measured on TSP teams. It also presents product and quality data from several TSP projects at one industry organization.

When thinking about productivity improvements, engineering management does not usually have precise data from which to determine the implications to the profit-and-loss statement of the organization. Deploying a new technology such as object oriented design is often done because everyone else in the industry is doing it, and the engineers want to use the latest hot technology. It is unclear how such process changes will impact the balance sheet; there is often no way to actually measure improvements. As a result, many organizations go year after year without knowing if meaningful improvement has occurred. Few organizations know the actual dollar cost of a thousand lines of code (KLOC); they think about head-count on a project, or person-years of effort, and so on. But when attrition is high and team turnover is not accurately measured, development costs become clouded.

This article will present a model of determining process effectiveness using the Team Software Process℠ (TSP℠). The costs to produce a KLOC using the TSP will be compared to a more traditional process focused on testing in quality. Finally, total lifetime costs of the TSP will be compared to the test-based process.

Software Development Cost Model

A simple model of a development process is used as the basis for comparison. The model will have five major phases: requirements (REQ), high-level design (HLD), implementation (IMPL), test (TEST), and release (REL). The detailed phase breakdown is shown in Table 1. These phases represent the work required of a software development project and whether or not a process actually produces these products. All systems have requirements; however, not all processes produce them.

To determine cost, two pieces of information are needed: how much time is spent in each phase and the cost of an engineer's time. The problem is that most software organizations do not have this information available. Therefore, it is necessary to take known aspects of development and infer other information in order to determine how much time is spent in each activity.

For example, many organizations have measured the time to find and fix defects in various test activities, along with the expected numbers of defects found in these activities [1, 2]. Then this information can be used to determine the time spent in test. Using such data, Table 2 (see page 10), on percentage of time spent in phase, was created.

"The cost to find and fix defects once the software has been released to the customer represents a significant portion of the total development cost ... By eliminating most of the time required to fix defects in a released product, an organization can focus its resources on new opportunities."

The numbers for the traditional test-based process agree with studies that indicate testing can take two-thirds of the development effort [2, 3]. The TSP model agrees with data from one TSP organization [4]. The TSP costs are roughly 25 percent less than the costs required to produce one KLOC using the traditional test-based approach. But this is only the cost to get the software into a state sufficient for first-customer use. In a traditional development process, once the software escapes from system testing, it undergoes extended periods of field trial and beta testing.

Defect repair costs typically soar up to 10 times the cost of integration and system testing. In fact, as much as half the cost to produce software is contained in repairing the software after the system test phase (termed release in this model, it includes beta trial, acceptance test, and customer use). Table 3 (see page 10) includes the costs associated with the field test and warranty periods. These numbers are based on the author's experience with several actual TSP development projects and are consistent with data from other organizations [4, 5].

Table 1: *Cost Model Phases*

Major Phase	Detailed Phase Breakdown
	Planning
Requirements (REQ)	Requirements
	System test plan
	REQ inspection
High-Level Design (HLD)	High-level design
	Integration test plan
	HLD inspection
Implementation (IMPL)	Detailed design (DLD)
	DLD review
	Test development
	DLD inspection
	Code
	Code review
	Compile
	Code inspection
	Unit test
TEST	Integration test
	System test
Release (REL)	Field trial
	Acceptance test
	Release

Traditional Process					Team Software Process				
Phase	**% Total Effort**	**Detailed Phase Breakdown**	**% Total Effort**	**Total Hours**	**Phase**	**% Total Effort**	**Detailed Phase Breakdown**	**% Total Effort**	**Total Hours**
	0.99%	Planning	0.99%	1.3		4.76%	Planning	4.76%	4.4
REQ	4.22%	Requirements	2.22%	2.9	REQ	19.87%	Requirements	9.93%	9.3
		System test plan	0.89%	1.1			System test plan	4.97%	4.6
		REQ inspection	1.11%	1.4			REQ inspection	4.97%	4.6
HLD	4.22%	High-level design	2.22%	2.9	HLD	18.06%	High-level design	9.03%	8.4
		Integration test plan	0.89%	1.1			Integration test plan	4.52%	4.2
		HLD inspection	1.11%	1.4			HLD inspection	4.52%	4.2
IMPL	24.63%	Detailed design	2.22%	2.9	IMPL	41.25%	Detailed design	8.21%	7.7
		DLD review	0.00%	0.0			DLD review	4.12%	3.8
		Test development	0.00%	0.0			Test development	4.11%	3.8
		DLD inspection	0.00%	0.0			DLD inspection	3.27%	3.1
		Code	11.65%	15.0			Code	7.43%	6.9
		Code review	0.00%	0.0			Code review	3.71%	3.5
		Compile	2.22%	2.9			Compile	1.26%	1.2
		Code inspection	0.78%	1.0			Code inspection	3.27%	3.1
		Unit test	7.77%	10.0			Unit test	5.87%	5.5
TEST	65.94%	Integration test	17.15%	22.1	TEST	16.05%	Integration test	7.20%	6.7
		System test	48.80%	62.8			System test	8.85%	8.3
Sub-total	**100%**		**100%**	128.7	**Sub-total**	**100%**		**100%**	93.3

Table 2: *Cost to Market*

The total lifetime cost for the TSP is less than 40 percent of the total cost of the traditional process. The total code produced by a traditional development organization is about four KLOC per engineer per year, assuming 1, 000 useful hours per engineer year divided by 252 hours per KLOC. This number agrees with those measured by various studies [5, 7, 8]. A TSP organization will produce just over 10 KLOC per engineer year, again assuming 1,000 useful hours divided by 95.6 hours per KLOC.

In the traditional process, early phase activities of requirements and architectural design tend to be cut short. Additionally, inspection activities take noticeably less time than in the TSP process for three reasons: They are eliminated under time pressure, engineers do not use sound data-based review methods, and work products are often not in a format that can be reviewed. The traditional process produces code very quickly, and then the real work begins in test. In contrast, the costs for the TSP tend to be front-loaded in early phases. More emphasis is placed on personal review and team inspections. This causes almost half of the effort to be expended before any code is even written. Figure 1 shows this graphically.

Notice that the phase investment of the TSP is relatively constant over the development cycle. Traditional processes have a higher cost in coding, and exponentially growing costs in test. The crossover point where the investment is equal in both processes occurs near the end of integration testing. At this point, we are indifferent to the two processes from an investment point of view; costs are roughly the same. Unfortunately, a traditional process has only just begun to pour dollars into the effort at the start of the system test.

Table 3: *Lifetime Software Development Costs*

Traditional Process					Team Software Process				
Phase	**% Total Effort**	**Detailed Phase Breakdown**	**% Total Effort**	**Total Hours**	**Phase**	**% Total Effort**	**Detailed Phase Breakdown**	**% Total Effort**	**Total Hours**
	0.51%	Planning	0.51%	1.3		4.65%	Planning	4.65%	4.4
REQ	2.15%	Requirements	1.13%	2.9	REQ	19.40%	Requirements	9.70%	9.3
		System test plan	0.45%	1.1			System test plan	4.85%	4.6
		REQ inspection	0.57%	1.4			REQ inspection	4.85%	4.6
HLD	2.15%	High-level design	1.13%	2.9	HLD	17.64%	High-level design	8.82%	8.4
		Integration test plan	0.45%	1.1			Integration test plan	4.41%	4.2
		HLD inspection	0.57%	1.4			HLD inspection	4.41%	4.2
IMPL	12.58%	Detailed design	1.13%	2.9	IMPL	40.28%	Detailed design	8.02%	7.7
		DLD review	0.00%	0.0			DLD review	4.03%	3.8
		Test development	0.00%	0.0			Test development	4.01%	3.8
		DLD inspection	0.00%	0.0			DLD inspection	3.19%	3.1
		Code	5.95%	15.0			Code	7.25%	6.9
		Code review	0.00%	0.0			Code review	3.63%	3.5
		Compile	1.13%	2.9			Compile	1.23%	1.2
		Code inspection	0.40%	1.0			Code inspection	3.19%	3.1
		Unit test	3.97%	10.0			Unit test	5.73%	5.5
TEST	33.69%	Integration test	8.76%	22.1	TEST	15.67%	Integration test	7.03%	6.7
		System test	24.93%	62.8			System test	8.64%	8.3
Sub-total	**51.09%**		**51.09%**	**128.7**	**Sub-total**	**97.64%**		**97.64%**	**93.3**
REL	48.91%	All post TEST	48.91%	123.3	REL	2.36%	All post TEST	2.36%	2.3
Total	**100%**		**100%**	**252.0**	**Total**	**100%**		**100%**	**95.6**

Payback Time

The TSP does not come for free. The training class requires two weeks of classroom time and several homework assignments. Most engineers complete training in 100 (plus or minus 20) hours, or about three weeks. Additionally, a TSP coach/Personal Software Process[SM] instructor is needed for every 50 to 100 engineers. That is the equivalent of another week per engineer for the whole organization to support the coach. Therefore, the incremental cost to deploy per engineer is about three weeks of fixed training cost, plus one week of variable cost per year. A payback graph of this appears in Figure 2.

In this model, this cost is paid back in 1.6 KLOC. This compares well with the 1,200 lines of code as reported by Teradyne [4].

Conclusions

The total cost to get software to market using the TSP is much less than the cost of using a process focused on testing. However, this represents only a portion of the total lifetime cost to develop software. The cost to find and fix defects once the software has been released to the customer represents a significant portion of the total development cost. The elimination of this cost results in the bulk of

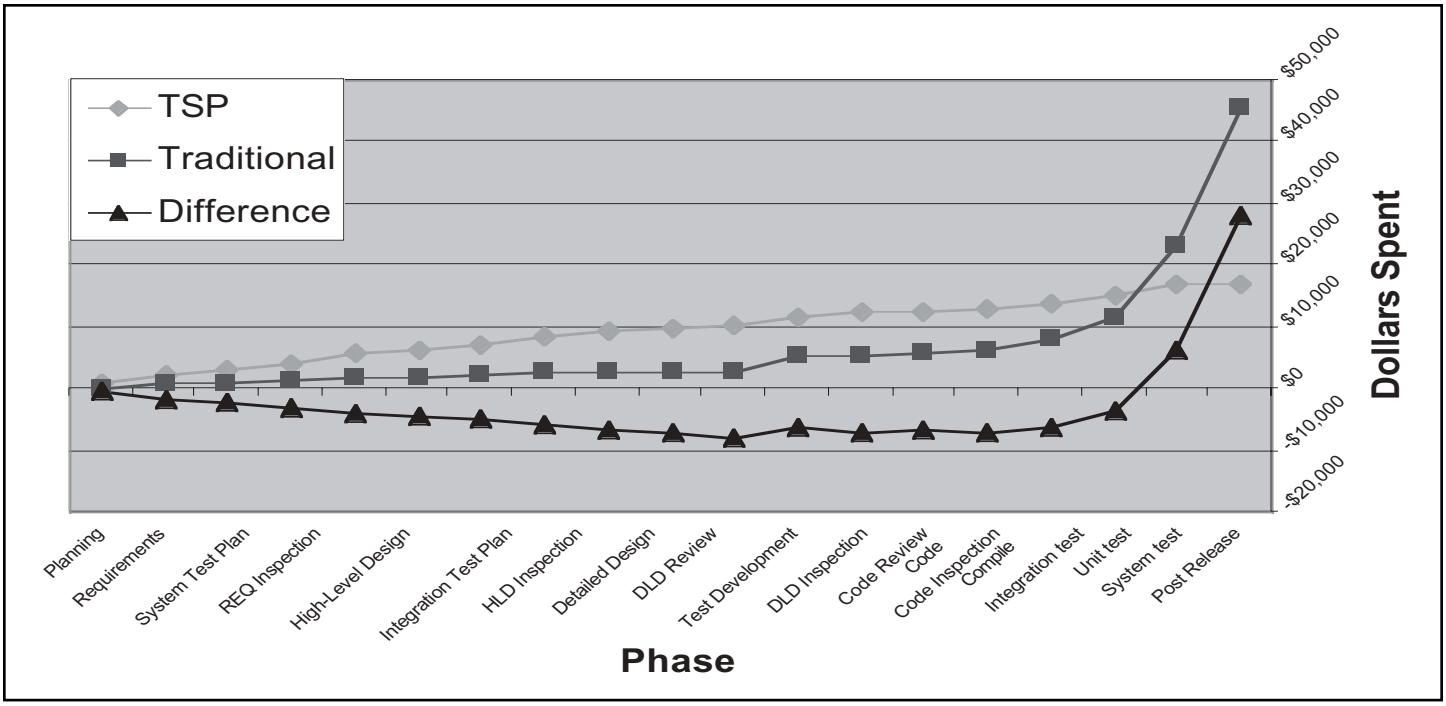


Figure 1: *Cost per Phase*

the savings for an organization using the TSP. By eliminating most of the time required to fix defects in a released product, an organization can focus its resources on new opportunities. The only investments required are training and a willingness to change.◆

References

1. Davani-Chulani, Sunita. Modeling Software Defect Introduction. University of Southern California. Center for Software Engineering, 1998.
2. Cole, Oliver E. The Cost of Debugging. OC-Systems White Paper. Available at: <www.ocsystems.com>.
3. Colburn, Timothy R., James H. Fetzer, and Terry L. Rankin, ed. Program Verification: Fundamental Issues in Computer Science. Kluwer Academic Publishers, 1993.
4. Musson, Robert. "The Results of Using a Team Software Process." Presentation at the Software Engineering Symposium. Software Engineering Institute, Sept. 1999. Pittsburgh, PA.
5. Reifer, Donald J. "Let the Numbers Do the Talking." CrossTalk Mar. 2002: 4-9.
6. Hatton, Les. "How Do We Satisfy Customers in the Long Run?" ESCOM 2001.
7. Caron, Michael. "Cost Justifying a Test Coverage Analyzer Tool." Newsletter of the Boston Software Process Improvement Network. Nov. 1995.

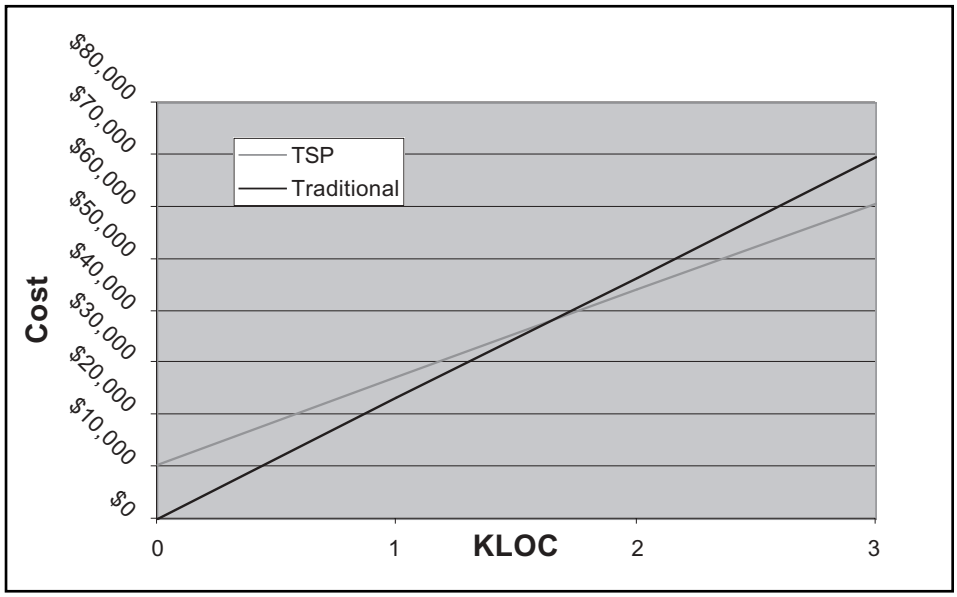


Figure 2: *Payback on Training*

About the Author

Robert Musson has more than 25 years of software experience as a development engineer and in various management positions. He spent 15 years at Teradyne helping bring to market a variety of products for the telecommunications industry. While there, he helped deploy the Team Software Process℠ (TSP℠) to the first industry site. He was vice president of business strategy at a small start-up before becoming a member of the TSP Initiative at the Software Engineering Institute. He has a master's degree in computer science from Illinois Institute of Technology and a master's degree in business administration from Northwestern University's Kellogg School of Management.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Phone: (412) 268-9130
Fax: (412) 268-5758
E-mail: ram@sei.cmu.edu

All the Right Behavior

David R. Webb
Software Division, Hill Air Force Base

Software projects using the Team Software Process℠ (TSP℠) have an unusually high rate of on-time completion. One of several key factors contributing to this accomplishment is the effective way TSP teams make use of earned value techniques to iteratively refine their plan as they work it. Because earned value is reviewed weekly, and because no value is earned either at the personal or the team levels until a task is fully completed, software engineers are highly motivated to perform good earned value practices. This article expounds upon this principle and examines how TSP teams succeed with earned value.

In a world where software projects can typically expect a 100 percent schedule slip, projects using the Team Software Process℠ (TSP℠) have an unusually high rate of on-time completion (Figure 1). In fact, I have been on three TSP teams that have experienced tremendous success in meeting or exceeding schedule. One of several key factors contributing to this accomplishment is the effective way TSP teams make use of earned value techniques to iteratively refine their plan as they work it.

Unlike many teams using traditional earned value methods, TSP teams understand what their data mean, they trust their data, and they actually use their data to guide them in a way that most projects cannot. They succeed for the following five reasons:

1. TSP earned value is based upon properly decomposed tasks.
2. TSP earned value is measured at the personal level.
3. TSP earned value is based on true task completion.
4. TSP earned value is defined in terms of task hours, not dollars.
5. TSP teams review their earned value data and update their plans each week.

A well-known TSP coach recently summed this up when he said, "TSP earned value drives all the right behavior." This article will expound upon this simple, but profound statement, examining how TSP teams succeed with earned value, and why this approach does indeed drive all the right behavior.

Breaking It Down

Earned value is simply a way of measuring progress. The following is a very simple (and unrealistic) example: If a project had 10 major tasks and each task was estimated to take 10 days to complete, the project would have a 100-day schedule and each task could be assigned a *value* equal to 10 percent of the whole (Table 1). As each task is completed, that value is *earned* by the project.

> *"Unlike many teams using traditional earned value methods, TSP teams understand what their data mean, they trust their data, and they actually use their data to guide them in a way that most projects cannot."*

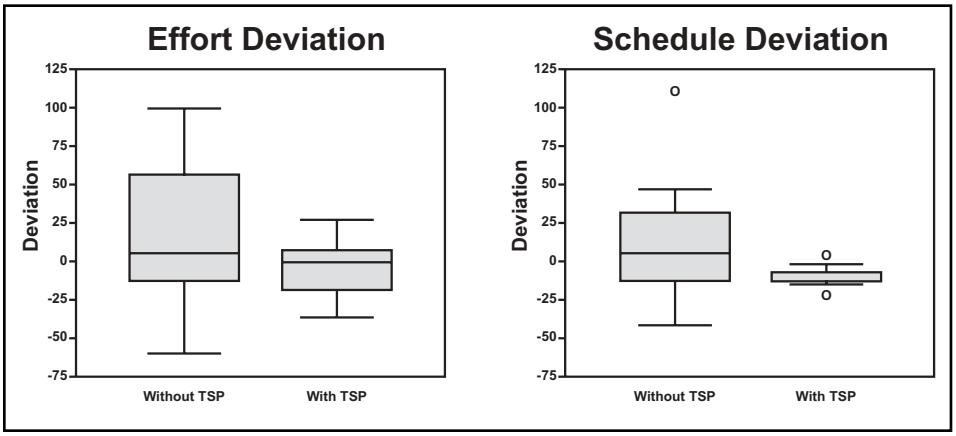
Traditional earned value throws another curve at the project and equates each task to a dollar value. In the 100-day example, if the total project costs were estimated at $1,000, each task would have a value of – you guessed it – $100. This is called the Budgeted Cost of Work Scheduled (BCWS).

Assuming a month has 20 working days, you can quickly estimate that two tasks should be completed each month, and the entire project should take five months (Figure 2). Once you have that baseline estimate, you can begin to add actual data to the chart as each task is completed, displaying the Budgeted Cost of Work Performed (BCWP) and the Actual Cost of Work Performed (ACWP) (Figure 3). (If you are getting confused at the proliferation of acronyms, do not worry that your IQ level has dropped. This is a common problem with traditional earned value. Later, I will show how TSP makes this easier, or at least reduces the number of acronyms you will need to know.)

As stated earlier, this example is far too simple for the real world. Let us look at a more realistic situation. Figure 4 (see page 14) details the earned value progression of a fictional project called *Project Genesis* after one year of work. This is a software intensive project and has been using classic earned value since its inception. Project Genesis is a firm believer in true *all-or-nothing* earned value and does not put any value on the earned value chart until a task is completed. The project was originally scheduled to take 18 months to complete at a cost of just over $1 million. A few earned value calculations would tell you that the project, as depicted in Figure 4 is right on schedule and a bit over budget, which you can tell by looking at the chart.

Now, look at Figure 5 (see page 14). This is the same project four months later. By earned value definition, the project is exactly on schedule and, though somewhat over budget, is pretty much in line with what you would expect. Looking at this chart as a manager or customer, you may be tempted to say, "Well, they were behind

Figure 1: *Comparison of Effort and Schedule Deviation with and without the TSP*



for a few months, but now they've caught up."

Right here we have nailed one of the major problems with the misuse of traditional earned value, because that statement is dead wrong. In fact, Project Genesis is at least three months behind schedule and is in serious trouble. It will deliver months late at a very high cost. If you find this statement confusing, you are not alone. Many customers have told me they feel like they are being duped by doubletalk when they see earned value charts, because they know by experience that despite good-looking charts, projects often fail to meet their schedules. Let us go over a few reasons why Project Genesis's good-looking charts are unintentionally hiding the truth.

(Right now, any earned value gurus reading this are hopping up and down and shouting at the page. That is because there *are* earned value calculations – such as Estimate at Completion – that can tell you whether or not Project Genesis is truly behind schedule, even if the chart looks good. Unfortunately, you have to *be* an earned value guru to know this; most managers and customers are not gurus.)

One reason that our fictional Project Genesis has incorrectly determined schedule performance is that the team did not properly break down its tasks. An important rule of earned value is that you must plan to see progress *each time* you report. If you do not, you really cannot tell when you are getting behind.

Look at Figure 5 again. This project was in trouble way back in October, but did not feel it until January because the slope of the planned earned value line was zero. The result of this lack of proper planning is that they had no feedback on their progress for several months.

If Project Genesis had been using the TSP, it would have avoided this pitfall entirely. TSP teams do not (or should not) allow long stretches of zero slope on their planned earned value charts. They can break the tasks down into very small increments, usually less than one week in duration, so small that they can be measured much more frequently than a month at a time. In the next few paragraphs, I will explain how this works, and why it is so beneficial.

Personal Earned Value

One prerequisite a team must meet prior to beginning the TSP is that all software developers on the team must be trained in the Personal Software Process℠ (PSP℠). There are numerous articles and books on the PSP [1], its tenets, and its numerous benefits. I will only mention here that PSP trainees learn how to plan and track their work at a personal level. PSP-trained software engineers know how to estimate in pieces, break their personal work down into measurable tasks, and gather minute-by-minute data on their progress (Table 2, see page 14). For people who have not had PSP training, this may seem like a ludicrous activity at the individual level. Those who have tried it out, though, have found it is really only a matter of personal engineering discipline and takes no more time than software development performed using the traditional ad hoc approach.

	Estimated Days	Value	Dollar Value
Task 1	10	10%	$100
Task 2	10	10%	$100
Task 3	10	10%	$100
Task 4	10	10%	$100
Task 5	10	10%	$100
Task 6	10	10%	$100
Task 7	10	10%	$100
Task 8	10	10%	$100
Task 9	10	10%	$100
Task 10	10	10%	$100
Totals	**100**	**100%**	**$1,000**

Table 1: *Simple Earned Value Breakdown*

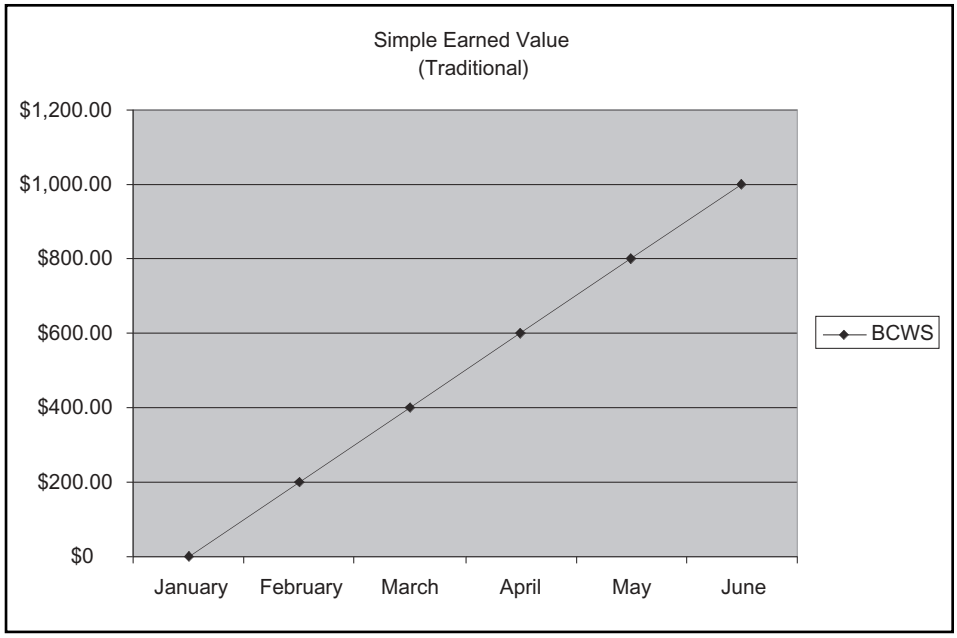


Figure 2: *Simple Earned Value Example – Estimates*

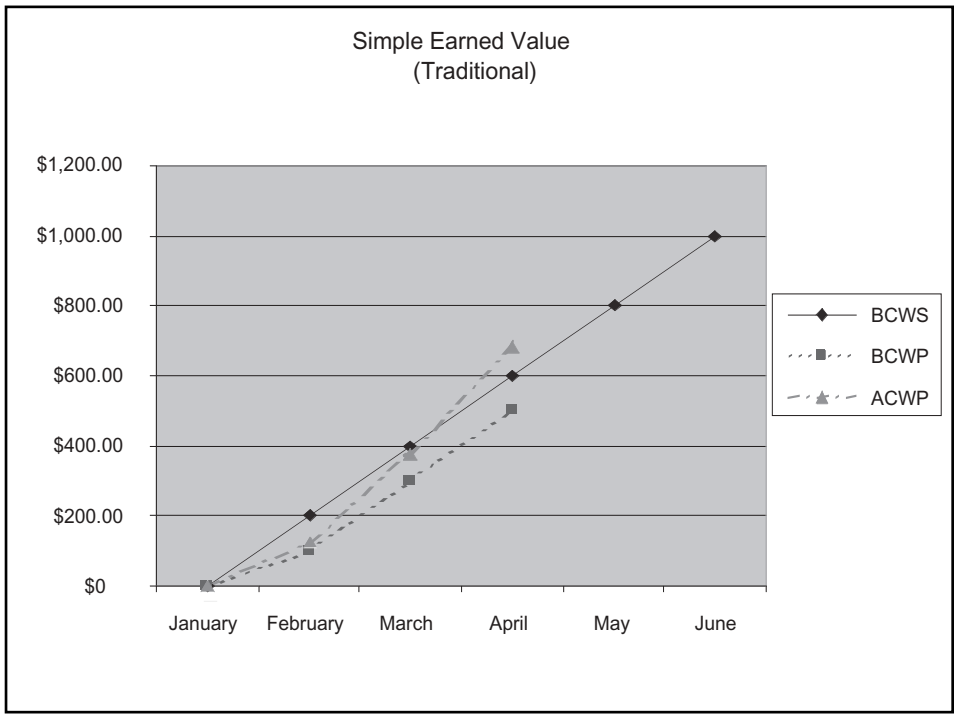


Figure 3: *Simple Earned Value Example – Actuals*

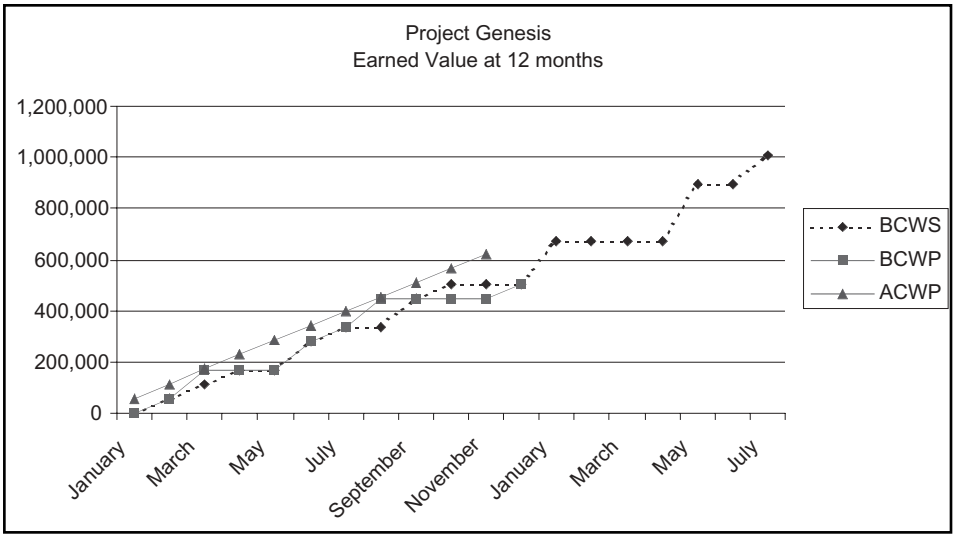


Figure 4: *Project Genesis Chart, Number One*

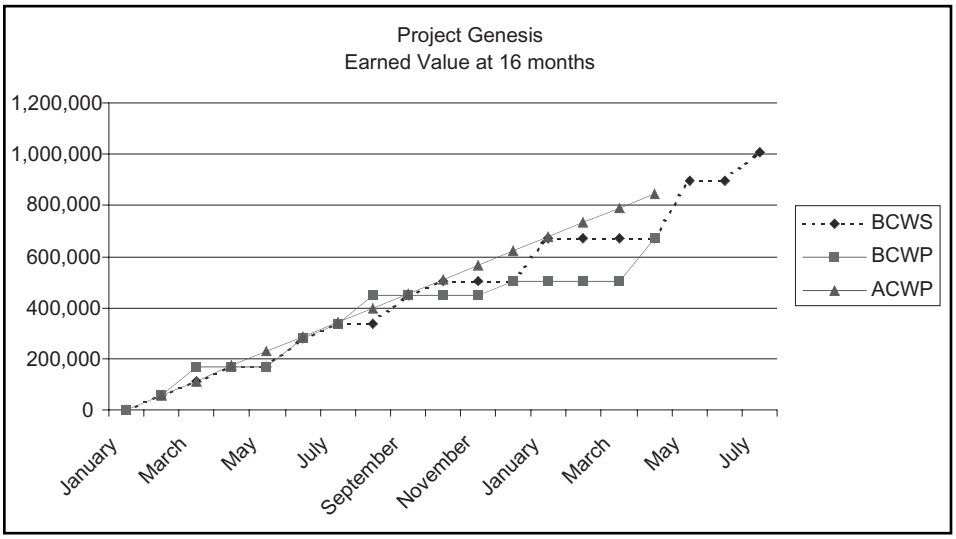


Figure 5: *Project Genesis Chart, Number Two*

PSP2 Project Plan

Student	David Webb
Program	List Sort
Instructor	Humphrey/Over

Time in Phase (min.)	
Planning	86
Design	86
Design Review	28
Code	28
Code Review	35
Compile	8
Test	41
Postmortem	30
Total	**322**

Table 2: *Portions of a PSP Planning Worksheet*

One of the many practical applications of this data-centric discipline is that an individual's tasks and estimates, broken down to a very fine granularity, are readily available to any team using the TSP. In addition, because of personal data gathering, TSP teams have real, measurable data on task completion, an absolutely essential element of tracking true earned value.

In other words, TSP team members can break tasks down and gather very accurate data on each task. This personal approach is what makes the type of earned value used by TSP teams possible.

Time Is Money, or Is It?

Another hallmark of TSP earned value is its lack of coupling between dollars spent and task completion. The reason for this ties back to the personal tracking methodology. TSP team members track *task time* in addition to *project time*. Task time is defined as the actual amount of time (in minutes) spent performing the specific tasks identified as key to project completion, minus any interruptions, which can consist of such mundane things as telephone calls, e-mails, meals, and trips to the restroom. However, it is important to note that any time spent on activities not typically identified on task lists also count as interrupts. This means meetings, work-related discussions, equipment setup, and even travel to other rooms or buildings are not counted as task time.

In a typical week, TSP engineers may track fewer than 20 hours of total task time. Of course, this is very different than the 40 hours of project time they have tracked since all of those *non-task items* must be done to support any project. As a result, the actual cost of a project in dollars is far different from the task hours earned. Rather than being a drawback, however, this personal tracking approach filters out unnecessary data and cleans up the earned value information. TSP team members know exactly how much time (not money) they are spending on the tasks that matter and exactly when those tasks are completed. This kind of precision leads to clearly understood earned value charts without reference to a budgeted or actual cost of anything (Figure 6).

Iterative Refinement

Finally, what sets TSP earned value apart from all other approaches – and frankly makes it work – is the frequency at which the data are reviewed[1]. Each TSP project begins with a *launch* (Figure 7). During the initial launch, tasks are defined at a very high level and estimated using gross measurements such as historical productivity (the number of lines of code a team typically can produce per hour). Using these high-level estimates, the team produces a detailed earned value plan.

This is typically the point when most software teams stop – that is as detailed as their plan becomes. The TSP team, however, then determines what the *next phase* of the project will be and uses PSP techniques to break that next phase into very detailed tasks of fewer than 10 task hours each. Using this level of detail, the team reviews progress against this earned value plan weekly. That is right, once a week, *not* once a month. To paraphrase TSP developer Watts S. Humphrey, projects do not slip a month at a time, they slip a day at a time, an hour at a time, and even a minute at a time.

In order to get insight into project issues at the earliest possible moment, project data must be reviewed much more frequently than the traditional one-month milestone. During each weekly meeting, it is immediately obvious to TSP teams which tasks and team members are ahead of or behind schedule, and which tasks or team members need assistance. It is

information like this that gives the team members the insight to make adjustments to task assignments, renegotiate functionality with the customer, or perform replanning activities to keep the project on track.

It is this combination of detailed planning, meticulous data gathering, and frequent reviews that makes the TSP's iterative refinement of project commitments possible. In fact, it is this ability, in combination with predictable test times due to the exceptionally high quality of the products they produce, that makes TSP teams so successful in on-time deliveries [2].

Let us take another look at Project Genesis, and this time let us assume the team had launched using the TSP. At first, the project earned value plan would look similar to the old plan they made before using TSP. Then, the team would determine that the next 10 weeks will constitute their *next phase*. This phase will consist of incorporating elements one through 10 as listed in Table 3. The team then would make an earned value plan for that period (Figure 8, see page 16).

Unfortunately, they still end up with a two-week period (Figure 8, circled) during which they cannot accurately determine their progress. This is due to a single program element – Element 2 – that is estimated to require more than the 30 task hours the project has estimated it will complete each week. Although this occurs early in the process, it could have a devastating impact on the outcome of the schedule. Using PSP techniques, the team members of Project Genesis break the large task down into its component elements (Table 4, see page 16).

Using this more refined estimate of Element 2, the earned value can be recalculated, producing the earned value chart in Figure 9 (see page 16). Notice that with this refinement, the *flat line* on the earned value chart has disappeared. Now the project has a detailed earned value plan that can be reviewed each week to determine if the project is actually meeting schedule.

All the Right Behavior

When the TaskView project first launched into the TSP at Hill Air Force Base in 1998 [3], the team did not have a lot of experience with this earned value methodology. Prior to the launch, the team had defined what they thought was a very thorough plan with more than 30 separate tasks. To do the launch and get the earned value chart using the TSP method, they doubled the number of tasks, which they felt was very thorough indeed. This did, however, leave three or four flat-line areas on their earned value chart similar to the one in Figure 8.

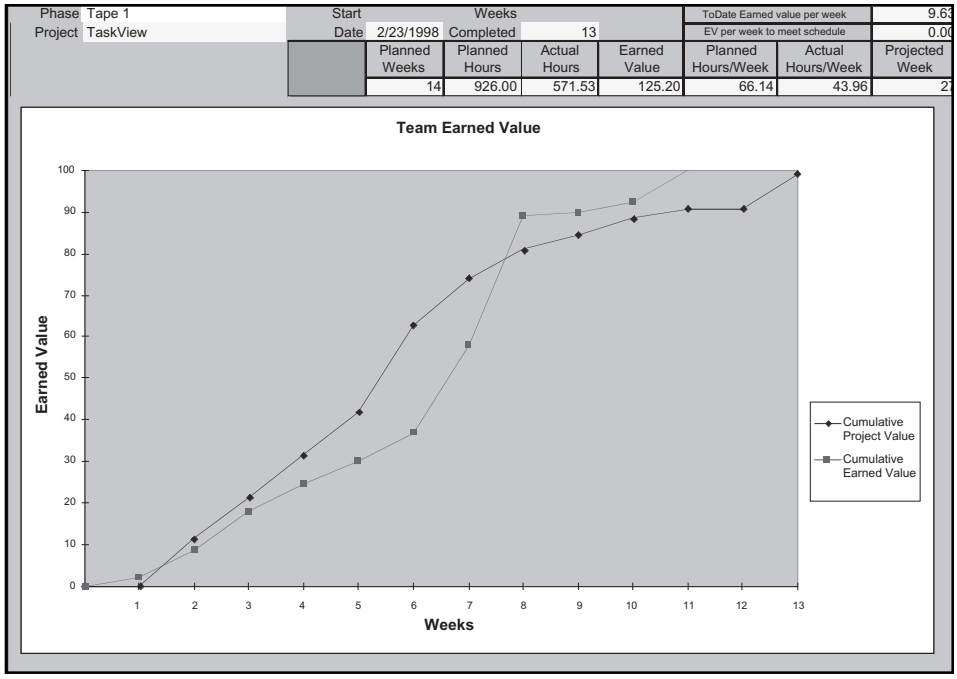


Figure 6: *An Actual TSP Earned Value Chart from the TaskView Project*

> *"My experiences show that traditional earned value, while an effective tool, is rarely used correctly to predict and manage project performance and, as such, is usually incomplete."*

I was serving as project leader at the time and a few weeks after our launch the engineers came to me with a complaint: "Our tasks aren't broken down enough to earn value every week!" So, with the insistence of the engineers, we used PSP phases (as shown in Table 2) to further refine the plan until our tasks were small enough (10 task hours or fewer) to show earned value each week. This activity increased the number of tasks to 204, and the engineers were happy about it. Those are the kinds of engineers TSP teams produce!

The launch coach was correct when he said that TSP earned value "drives all the right behavior." Because it is reviewed each

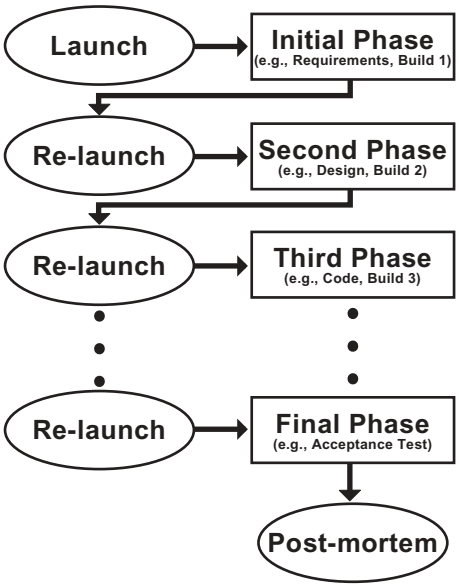


Figure 7: *The TSP Launch Process*

Task	Estimated Hours
Element 1	28
Element 2	85
Element 3	12
Element 4	26
Element 5	22
Element 6	5
Element 7	26
Element 8	29
Element 9	19
Element 10	23

Table 3: *Project Genesis Tasks for the "Next Phase"*

Task Element 2	Estimated Hours 85
Planning	10
Design	33
Design Review	17.5
Code	12
Code Review	6
Compile	0.5
Test	4
Post-mortem	2

Table 4: *Breakdown of Element 2*

week, and because no value is earned either at the personal or the team levels until a task is fully completed, software engineers are highly motivated to perform these good earned value practices:

- Follow a strictly defined process with very specific entry and exit criteria as well as well-defined tasks.
- Break large tasks into small pieces that can more easily be estimated and tracked and shows regular progress.
- Project forward to see if their progress will meet the current schedule.
- Re-plan when unplanned events arise.
- Do the right work, in the right order, at the right time.

My experiences show that traditional earned value, while an effective tool, is rarely used correctly to predict and manage project performance and, as such, is usually incomplete. In fact, many customers not only feel overwhelmed with acronyms like BCWS, BCWP, and ACWP, they do not trust charts that have all of those data and more splattered across them because so often they have seen results contrary to those charts. The TSP earned value techniques work because they collect data at the right level, they are simple, and they are measured each week.

TSP earned value works. It does indeed drive all the right behavior.◆

References

1. Humphrey, Watts S. "Making Software Manageable." CrossTalk Dec. 1996.
2. McAndrews, Donald R. The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices. Pittsburgh: Software Engineering Institute, Nov. 2000. 30.
3. Webb, David R., and Watts S. Humphrey. "Using the TSP on the TaskView Project." CrossTalk Feb. 1999: 3-10.

Figure 8: *Project Genesis Using TSP*

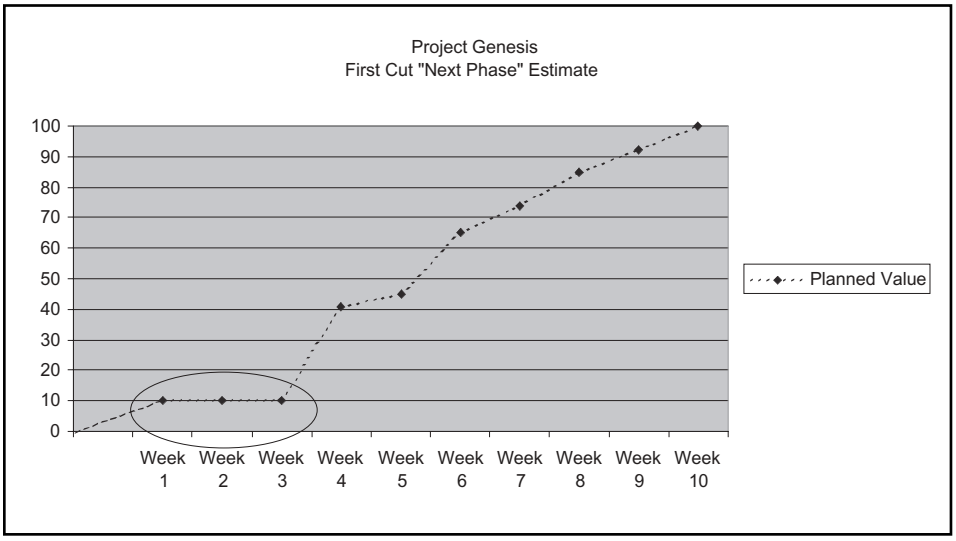
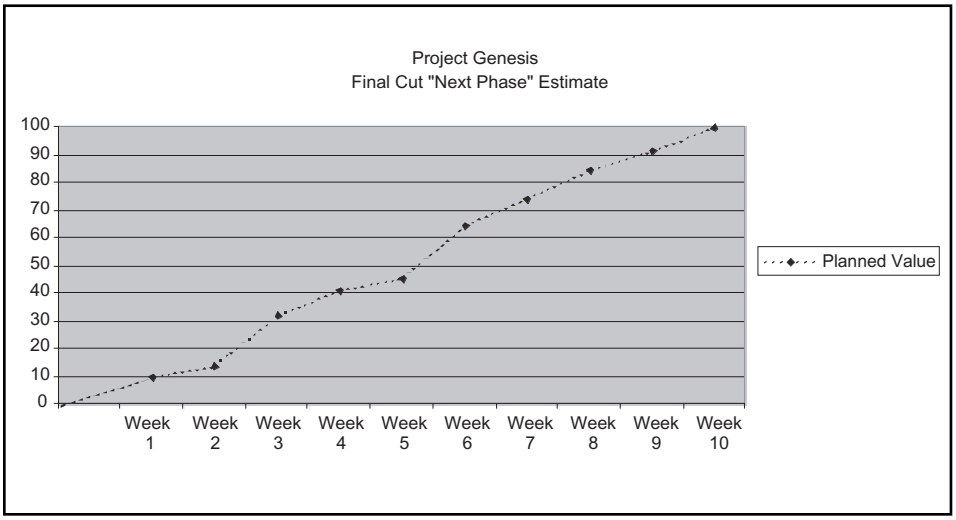


Figure 9: *Project Genesis "Final Cut"*



Note

1. Since schedules are living documents and must be renegotiated with the customer during development as new situations arise, this statement does not imply TSP teams always meet the original schedule set forth at project inception. I was involved with three TSP projects that met or exceeded the negotiated schedule. In one case, the customer shortened the original schedule; in another, the due date was extended; in the third, the acceptance test group was not ready to receive the product so more functionality was added during the *down* time. However, the ability to accurately renegotiate schedules *on the fly* and early in the process, to a customer's satisfaction, is one of the great strengths of TSP earned value and its iterative approach.

About the Author

David R. Webb is a project management and process improvement specialist for the Software Division of Hill Air Force Base in Utah, a Capability Maturity Model® for Software Level 5 software organization. He has 14 years of technical, program management, and process improvement experience with software in the Air Force. Webb is a Software Engineering Institute-certified instructor of the Personal Software Process℠ and a certified Team Software Process℠ launch coach. Webb has a bachelor's degree in electrical and computer engineering from Brigham Young University in Provo, Utah.

7278 4th Street
Software Division, Bldg. 100
Hill AFB, UT 84056
Phone: (801) 777-9737
Fax: (801) 775-3023
E-mail: david.webb@hill.af.mil

Managing a Company Using TSP Techniques

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

This article describes the experience of using techniques from the Team Software Process℠ (TSP℠) to manage a small software consulting company. The company's management team used TSP techniques to run the organization. The authors describe how the TSP has been adjusted and the lessons learned from this experience.

The Team Software Process℠ (TSP℠) is designed to facilitate superior performance of software development teams. Although the TSP is designed for the software domain, it is so well defined that it can be used in other domains. For example, the management team of QuarkSoft, a start-up software company, decided to apply TSP techniques to run the company. In particular, they have been using TSP techniques as the baseline process for planning, controlling, and performing the activities of all the members of the management team.

QuarkSoft's management team members have been using the TSP approach for more than 10 months and are very pleased and excited with the results. They are convinced that the TSP has been fundamental for having effective company management. For example, the strategy and objectives for the company are well defined, risk management has been implemented, communication problems have been reduced, realistic plans have been built, and important problems are addressed in a timely manner. Their experience shows that the TSP is a powerful team process that can be customized to improve the performance of teams beyond the software domain.

This article explains the motivation for using the TSP as an executive management process, how the TSP has been adjusted to fit the QuarkSoft management team's needs, and the lessons learned from this experience. The authors assume the reader knows the main concepts and products of the TSP. Please refer to [1, 2] for a description of the TSP.

Background

QuarkSoft is a small start-up company whose core business is outsourcing of software development. Since QuarkSoft's differentiator is quality software development, the company decided to base its operations on the Capability Maturity Model® for Software (SW-CMM®)[3] and chose the Personal Software Process℠ (PSP℠) [4] and the TSP as the means for implementing the Capability Maturity Model®.

QuarkSoft's staff is organized into a management team that runs the company, a group of software engineers that develop the software, and a small administrative staff that attends to administrative and operational issues.

The management team (MT) is composed of the chief executive officer (CEO), the chief operations officer (COO), the chief financial officer (CFO), the research and development officer, and the software engineering process group (SEPG) chief engineer. The MT is responsible for all business and operation decisions. Major decisions are made by the consensus of the MT members.

> *"Although the TSP is designed for the software domain, it is so well defined that it can be used in other domains. For example, the management team of a start-up software company decided to apply TSP techniques to run the company."*

Since the company is relatively small, each MT member has additional duties as compared to traditional companies. Besides overseeing all the operations of the company, the CEO is responsible for marketing, contacting new clients, and closing contracts. In addition to assuring that all projects are on-time, the COO participates in quoting projects, does TSP coaching to internal software development teams, trains new hires in PSP, and ensures that all the engineers are following the TSP. The COO is a Software Engineering Institute-authorized PSP instructor and TSP launch coach. In addition to handling financial issues, the CFO oversees legal, human resources, and daily administrative issues. Finally, in addition to being in charge of processes, the SEPG chief engineer coordinates quality assurance and configuration management activities, helps train new hires, and makes sure that data are collected.

The Need for TSP

When QuarkSoft was created, the MT used typical management practices. A business plan and a strategic plan were elaborated. The mission, vision, strategic projects, and indicators were defined. These practices worked fine for setting up the company. However, after a couple of months, the business plan no longer represented reality. On the day-to-day activities, the MT started to operate in firefighting mode. They would solve the main problem of the day with no planning at all. The MT realized that without day-to-day planning it was easy to be caught by the urgent problems of the day. This problem-driven management style did not leave time to work on fundamental problems. A new management strategy was necessary.

Start-up companies need to respond rapidly to changes in the business environment to be able to survive [5, 6]. Many outside factors such as market changes, competitors, financial valleys, cancelled contracts, and delayed payments demand prompt attention. Very often, the strategy, planning, and objectives have to change according to these external factors. The business environment demands certain skills from MTs such as good communication, good planning, constant feedback, adaptation to changes, and prompt and accurate information to make decisions.

The TSP's main concepts and techniques provide the foundation to assemble and guide a team with such skills. The decision to use the TSP as the base process for organizing and running the

MT was somewhat natural and aligned with the company culture for two reasons. First, the company has experience using TSP (i.e., TSP is used in all software development projects). Second, most of the MT members (except the CFO) are TSP and PSP trained. Thus, running and managing the company was seen as a project, and the MT as the team to perform it. Making this decision was not difficult; the real challenge was adjusting the TSP to the needs of the MT and to start using it.

Implementation

The first task was to define the general process. The MT agreed on dividing the project into one-month cycles that corresponded to calendar months. Each cycle would start with a re-launch and would end with a post-mortem. The re-launch would take two to four days. During the cycle, the MT would have one-hour weekly status meetings.

The MT has used this TSP approach since July 2001 (cycle one), including six cycles in 2001 (i.e., July to December) and five cycles in 2002 (i.e., January to May).

Adapting the TSP for the MT needs has been a gradual process. During the first two re-launches, the MT focused on making detailed plans for each MT member. Few objectives were set, and no risk analysis was performed.

In cycle three (September 2001), the MT started writing the minutes from weekly meetings, did some data analysis, and started holding post-mortem meetings. In addition, a general meeting was added at the end of the re-launch agenda. All QuarkSoft employees attend this meeting in which the MT presents the current status of the company, the short and long-term plans, and the status of issues and problems of general interest.

By cycle six, the re-launch process had been tailored to meet most of the MT needs. Risk analysis was recognized as an important part of the re-launch, and several forms and standards to report and analyze data had been developed.

Adjusting the TSP for Management

The MT uses the following definitions to facilitate communication, planning, and data collection: *Overhead* is the time spent in unplanned activities. *Available time* is the time a MT member is supposed to be in the company (e.g., 40 hours per week). *Available task time* is the time that each MT member has for planning purposes. As the TSP advocates, the available time is not the same as available task time. Part of the available time is used for answering e-mail and telephone calls, coffee breaks, interruptions, etc.; this time is called *dead time*. *Task time* is the time spent on planned activities. *Direct total time* is the sum of overhead and task time.

Launch and Re-Launches

A regular TSP project starts with a launch for the initial cycle and continues with re-launches for each subsequent cycle. As mentioned before, the MT decided to perform only re-launches. However, in January 2002, the MT began the practice of performing a launch for a yearlong period followed by re-launches for each calendar month. During the launch, the MT revisits the company mission, vision, and general strategy, and defines the objectives, strategy, and milestones for the year. Re-launches are for detailed planning for every month.

A typical re-launch lasts three days. It follows the general structure of a TSP launch, i.e. nine sequential meetings. However, the MT re-launch includes only the equivalent of the TSP meeting numbers two (objectives), three (strategy), four (general plan), six (detailed plan), and seven (risk analysis). In addition, other meetings have been added as illustrated in the following bullet points. A typical agenda for a re-launch consists of the following meetings:

> *"During the launch, the MT revisits the company mission, vision, and general strategy, and defines the objectives, strategy, and milestones for the year. Re-launches are for detailed planning for every month."*

- Review personal issues.
- Review financial information/reports and forecasts.
- List important issues that have not been addressed or that came up in the previous cycle.
- Define objectives and priorities (both for the company and for the MT).
- Define the strategy, perform a risk analysis, and determine milestones and important dates for the cycle.
- Identify the activities for the cycle, activities for future cycles, and responsibility for each activity.
- Resolve dependencies.
- Define the date and time for weekly meetings and next re-launch.
- Detailed planning (individually).
- Prepare the presentation for the general meeting.
- Hold the general meeting.

During the review of personal issues, each member of the MT describes personal issues that might be or will be affecting his/her performance such as feeling burnout, a wedding, a vacation, or a new baby.

Risk analysis is fundamental because there are many risks that could lead to bankruptcy or company dissolution. The risk analysis process follows the TSP approach. Risks are sorted according to likelihood and impact. Top risks are considered and activities to mitigate them are defined. Each risk has a responsible person who tracks the status of the risk. Contrary to a regular TSP re-launch, the MT decided to perform risk analysis before detailed planning because the risk analysis might produce a change in the strategy of the cycle.

Running the Plan

After the re-launch, every MT member has a list of the tasks that he or she will perform during the cycle. Each MT member records the time that he/she has spent in each of the planned tasks. When a planned task is finished, the MT member that performed the task gets earned value for it. If the MT member performs an unplanned task, he/she records it as overhead.

Several administrative tasks have been detected. Examples include consolidate MT data; make agendas for weekly, post-mortem, and re-launch meetings; back-up documentation; and keep the project notebook (i.e., a binder with hard copies of all the documents produced) up to date. These tasks have been distributed among members of the MT.

Weekly Status Meetings and Post-Mortem

Weekly meetings follow an agenda. Typical roles for the weekly meetings include the discussion leader for each agenda item, the timekeeper, and the recorder. Minutes are written during the meeting and e-mailed to the MT after the meeting is finished.

The MT modified the TSP weekly status meeting agenda. The major topics of

the agenda are as follows: personal issues, report of each of the MT members (i.e., each MT member summarizes the status of his/her area), follow-up of objectives and risks, status of the individual and team plan (e.g., overhead, earned value), next-week plan, summary, and meeting wrap-up.

The post-mortem of this cycle is done just before the re-launch of the next cycle and takes a couple of hours. During post-mortem, the analysis is focused on estimation errors, overhead and direct total hours per week, and problems with collecting and interpreting data. Process improvement proposals and new processes needed are addressed, too.

Byproducts

The MT has produced several products such as a process for preparing post-mortem data, forms to summarize data from the cycle (e.g., overhead, task time, uncompleted tasks), calculation of indicators (e.g., estimation errors, overhead vs. task time), standards for collecting data, agendas and minutes, and a checklist to submit data for consolidation.

It has been necessary to define some policies. These include rules for setting deadlines to submit weekly and post-mortem data, rules to determine the time an agenda has to be distributed before the meeting, and rules for canceling or delaying a meeting.

Lessons Learned

The Process

TSP team member roles (e.g., design manager, customer interface manager, etc.) do not apply to the MT context. The MT tried to define new team roles but it was not worthwhile. The MT realized that managing the company is their project. Consequently, the roles correspond to the job positions (i.e., the CEO, the COO, etc.). Thus, there was no need to redefine them. Nevertheless, there are several administrative activities necessary to manage the team. As mentioned before, these activities were assigned to MT members.

Work balance is not done as it is typically done in the TSP because the MT members have very specific activities. There are few tasks that can be performed by more than one team member. Nevertheless, the MT does some redistribution of tasks in situations when one MT member is overloaded and the activities that he/she has to do are of high priority.

Weekly meetings have been exceptionally helpful to improve MT communication. As the company grows, it is difficult for every MT member to be aware of the current status of each area of the company. During weekly meetings, each MT member presents a summary of major decisions, initiatives, problems, etc., in his/her area. This practice has increased the levels of awareness about the status of the company and the issues that each MT member is dealing with.

One aspect that has not been resolved completely involves timing and meeting time commitments. Examples of this issue include submitting data for consolidation on time, starting meetings on time, and scheduling three full days in a row for a re-launch. Due to the nature of the work that the MT members perform, it is difficult to force them to meet these types of time commitments.

Collecting time data has been challenging. The MT uses the prototype TSP tool that is provided by the Software Engineering Institute to collect data. This prototype is implemented in Microsoft Excel, which means that the tool is not very accessible. For example, when MT members go away from headquarters, which is very often, it is impossible to carry the tool to keep an accurate time log. This mobility problem has been addressed by writing down the time log on a piece of paper, or using a simple time log tool that runs on hand-held computers, or more drastically, estimating the times. The inconvenience in all these solutions is re-typing the time log into the TSP tool.

> *"During the early cycles, a considerable amount of time in re-launch was invested in planning. Now, the MT spends more time defining the strategy and attending to urgent issues."*

Re-Launches

Re-launches are very effort demanding; 10-hour workdays and pizza dinners are common. At the end of the three days, the MT is really tired. It has been proposed that one day be added to the re-launch. However, getting three consecutive full days from an executive is difficult. Getting four days is unrealistic. Other strategies have been tried with different levels of success such as having a social activity at the end of day two, stopping work at 6 p.m. or starting at 10 a.m. on one of the three days.

During the early cycles, a considerable amount of time in re-launch was invested in detailed planning. Now, the MT spends more time defining the strategy and attending to urgent issues. For example, there are situations when the plan for the cycle depends on getting a contract. Thus, performing a good strategy and risk analysis should be the priority.

One important difference between the MT and a software team is that there are fewer interdependencies in the activities of the MT members. Detailed planning can be done individually because just a few interdependencies have to be cleared out.

Many issues and action items are produced during weekly meetings and re-launches and are recorded in the minutes. However, there are so many issues and action items that this approach is no longer adequate. There is a need for improving the process of tracking and prioritizing issues and action items.

General

Characterization of quality work has been an unresolved issue. Quality management is a fundamental component of the TSP (e.g., the fifth meeting of a TSP launch). Moreover, one of the QuarkSoft's driving ideas is to work with quality. The MT has searched for a way to include quality management in the TSP-adapted process. But, what does quality mean in the MT work context? Unfortunately, the MT has not found a satisfactory answer to this question.

MT members have had difficulties in stabilizing their task time estimates. Specifically, the CEO time estimates have varied greatly because among other factors, he performs many different activities throughout the cycles. This reduces the chance to collect historical data on the same activity. Moreover, some of the CEO's tasks do not seem to behave consistently. For example, closing a contract has a wide range of variability.

The CEO has approached this problem by shifting from fine granularity estimation to coarse granularity estimation in closing-a-contract estimating. Instead of estimating the entire time for closing each contract in full, he uses historical data from previous contracts to assign a number of hours to this activity per week.

The COO has adopted a similar

approach. He uses historical data to calculate the average time devoted to supervising a project, then uses this average to estimate the weekly amount of time he would plan for each of the projects he supervises.

The CFO is the only member of the MT who has no previous background on processes. Although she comes from a managerial background, she has been very receptive to the concepts of the TSP and on collecting data. She shows a commitment to improving her estimates. She comments that having detailed plans allows her to better organize her days.

When beginning the second cycle, the MT agreed that its members needed to be up to date in their area of expertise (e.g., the CFO needed to be current in tax reforms, and the COO needed to be current in new technologies). Thus, the MT started planning time every week for what was called continuous education and actualization. However, it did not work; the workload for daily activities was too much. As a result, the MT started using the time scheduled for continuous education for other more important activities. Continuous education still is a major concern, so the MT changed strategies. Nowadays, each MT member is required to give a seminar every cycle that is open to all employees. He/she presents a paper that he/she has read. This practice has been working fine so far.

Workload

Detailed planning has been very useful in detecting important issues such as excessive workload, unimportant tasks, identification of critical weeks, company milestones, and major company turning points.

It has been particularly helpful to have data on workloads and direct total time of each MT member. This data has helped put the effort required to run the company in perspective. For example, there have been weeks when a team member has worked more than 50 direct total hours per week (this figure means that he/she has spent at least 60 to 70 hours a week doing company-related activities). It is easier to see when the company is overshadowing the MT member's life. Working more than 50 direct total hours means that the MT member has had no time for family, social activities, personal care, etc. This fact is important because stressed MT members are less effective. But knowing how much time the MT must commit to the company is still a debatable issue.

Overhead

For planning purposes, the MT used an available time of 40 hours per week; a utilization factor of 75 percent, that is, to use 30 hours per week for direct planned tasks (i.e., available task time); and a time of 10 hours for overhead and dead time. After a few cycles, it was clear that each role behaved differently; planning based on those figures was not resulting in accurate estimations.

For example, historical data showed that the CEO invested about 50 percent of his/her direct total time in overhead. For the CFO, COO, and SEPG, the figure was about 30 percent to 40 percent overhead. These numbers were critical, because many important planned activities were not performed. The earned value of the team was consistently below 70 percent and, in some weeks, below 45 percent.

"Having the entire team make decisions has been a major advantage. Every MT member knows and decides on the most convenient time to do important things, such as making major investments and purchases, hiring engineers, and scheduling vacations and training."

It was decided that the CEO would make his plans with a utilization factor of 50 percent, that is, allocating 20 hours per week for planned activities and the rest for overhead and dead time. For the rest of the team, a utilization factor of 65 percent was decided (i.e., 26 hours per week of available task time). The MT has been producing more realistic plans since these changes were implemented. The team's weekly earned value has improved, and the overhead decreased.

One interesting issue derived from using these utilization factors is free time. Assume that in a certain week a MT member finished all his/her planned tasks and he/she has no overhead. In other words, he/she has some free time in that week. The MT decided that each member should maintain a pool of tasks. This pool contains tasks that are important but have been delayed for future cycles. Thus, the team member with free time can check his/her pool of tasks and begin doing the one with highest priority according to the cycle's objectives and strategy.

Another interesting issue that arose from overhead analysis is that there are different types of overhead. In particular, the CEO and COO started collecting transportation data and detected that they invested a lot of travel time visiting customers to conduct negotiations, project supervision, and meetings. They found that in some weeks they invested as much as six hours on transportation (30 percent of the direct task time for the CEO). From this the question arose, "Should the MT charge the customer for this time?" This question is relevant, especially when the client delays a meeting or cancels it at the last minute.

Conclusions

MT members produce a wealth of information that can be used effectively to make decisions and to improve team performance. The plan can be adjusted quickly, according to business needs, priorities, and risks. For a start-up company, short-term objectives might change rapidly. The TSP approach to management has allowed the MT to make rapid adjustments to these changes. In addition, having a detailed cycle strategy makes it easier to plan for such things as vacations, conferences, and business trips, or detecting warnings of employee burnout and recommending the best time for vacations for overwhelmed team members.

There has been only one cycle without a re-launch. The re-launch of February 2002 had to be cancelled. The result was a management nightmare. MT members went back to fire-fighting mode, the total direct time increased, several unattended issues caused several important problems, and one MT member declared himself burned out. Moreover, re-launch for cycle three took four days. After this experience, the commitment to do re-launches as planned and to work based on plan has strengthened.

Having the entire team make decisions has been a major advantage. Every MT member knows and decides on the most convenient time to do important

things, such as making major investments and purchases, hiring engineers, and scheduling vacations and training. In addition, TSP has helped the MT to focus on important things and on aligning MT activities to the strategy and priorities of the cycle.

The MT considers that the time invested in doing all the administrative TSP activities is reasonable. The MT is investing about 14 percent of its time in re-launches and less than 5 percent in management activities (e.g., weekly meetings, preparing post-mortem data, etc.).

So far, this TSP experience has been successful. The MT is very pleased with the results. MT members are enthusiastic about the data they collect and their performance findings. They are continuously looking for ways to improve their team process and plan to keep using and improving the TSP approach to management.

This experience in managing a company using TSP techniques shows that the TSP is a very powerful process that can be tailored for other domains besides software development. This experience also suggests that the ideas behind the TSP can be used as the foundation for any teamwork.◆

Acknowledgments

We want to thank the QuarkSoft management team for sharing with us their experiences and information.

References

1. Humphrey, W. Introduction to the Team Software Process℠. Addison Wesley Longman, 2000.
2. Humphrey, W. "The Team Software Process℠." Technical Report CMU/SEI-2000-TR-023, 2000: 51.
3. Paulk, M., et al. The Capability Maturity Model: Guidelines for Improving the Software Process. Boston: Addison-Wesley, 1994.
4. Humphrey, W. A Discipline for Software Engineering. Boston: Addison-Wesley, 1995.
5. Fayad, M., M. Laitinen, and R. Ward. "Thinking Objectively: Software Engineering in the Small." Communications of the ACM 43.3 (2000): 115-118.
6. Paulk, M. Using the Software CMM in Small Organizations. The Eighth International Conference on Software Quality. Portland, Oregon. 13-14 Oct. 1998.

About the Authors

Carlos Montes de Oca, Ph.D., is a research professor in the Department of Computer Science at the Center for Mathematical Research (CIMAT). He is a Software Engineering Institute-authorized Personal Software Process℠ (PSP℠) instructor and Team Software Process℠ (TSP℠) launch coach. He has more than 10 years experience in software development and management. Dr. Montes de Oca is involved in several TSP and PSP projects in both academia and industry. His current research interests include software process improvement and software quality. Montes de Oca has a doctorate degree in computer science from Louisiana State University.

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

Miguel A. Serrano, Ph.D., is a researcher in the Department of Computer Science at the Center for Mathematical Research (CIMAT). He is a Software Engineering Institute-authorized Personal Software Process℠ instructor and Team Software Process℠ launch coach. His current research interests include software process improvement, statistical process control, and software quality. Dr. Serrano has master's degrees in information systems and decision sciences and in system science, and a doctorate degree in computer science from Louisiana State University.

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

Coming Events

September 24-27
Software Test Automation Fall Conference
Boston, MA
www.sqe.com/testautomation

October 7-10
MILCOM Military Communications Conference
Anaheim, CA
www.milcom.org/2002/

October 14-16
20th Annual Pacific Northwest Software Quality Conference
THE PACIFIC NORTHWEST SOFTWARE QUALITY CONFERENCE
Portland, OR
www.pnsqc.org

November 3-6
3rd Annual Amplifying Your Effectiveness (AYE) Conference 2002
Phoenix, AZ
www.ayeconference.com

November 4-8
Software Testing Analysis and Review Conference
Anaheim, CA
www.sqe.com/starwest

November 11-14
National Defense Industrial Association
Denver, CO
www.ndia.org

November 18-21
International Conference on Software Process Improvement
Washington, DC
www.software-process-institute.com

April 28-May 1, 2003
Software Technology Conference 2003
STC
Salt Lake City, UT
www.stc-online.org

SEI CMM Level 5: Lightning Strikes Twice

Gregory P. Fulton
Boeing Integrated Defense Systems

In December 2001, the Boeing Military Aircraft and Missiles Seattle Site (AMSS) organization (within the former Boeing Military Aircraft and Missiles) achieved a Level 5 rating using the Software Engineering Institute's Capability Maturity Model® (CMM®). This rating was achieved just 12 months after receiving a Level 3 rating in December 2000. While making such a rapid progression from Level 3 to Level 5 is uncommon, it was not unprecedented within Boeing. In 1995-1996, the Boeing Space Transportation Systems organization (within the former Boeing Space and Communications) achieved a Level 3 to Level 5 transition in approximately six months. This article describes three essential factors that were common to both organizations that enabled such a rapid progression from CMM Level 3 to Level 5.

In the August 2001 "Process Maturity Profile of the Software Community 2001 Mid-Year Update" [1], the Software Engineering Institute (SEI) reported that the median time to move from Capability Maturity Model® (CMM®) Level 3 to Level 4 is 33 months, followed by 18 additional months to reach CMM Level 5. Obviously, accomplishing the same objective in six to 12 months is a remarkable achievement. To do it twice – as two organizations within The Boeing Company did – is not only a credit to the organizations involved, but to the underlying principles that were present in both efforts.

In December 2001, just 12 months after receiving CMM Level 3, the Boeing Military Aircraft and Missiles Seattle Site (AMSS) organization achieved CMM Level 5. Previously in 1996, the Boeing Space Transportation Systems (STS) organization transitioned from CMM Level 3 to Level 5 in approximately six months.

It is not a mystery that any successful improvement effort – regardless of its timeline – requires sponsorship, practitioners' involvement, and a focus on the organization's business needs. But those elements by themselves do not necessarily equate to an accelerated timeline, which begs the question: What additional factors must be present in an organization in order to achieve high-maturity in a relatively short amount of time?

Although the Boeing AMSS and STS achievements[1] were separated by approximately six years and involved different personnel, they shared the following three fundamental elements in their approach to improvement: software engineering process group (SEPG) composition, a strong tie to the business case, and projects that had institutionalized a data-driven approach to management. The following sections will describe each of these common elements and how they contributed to each organization's success.

SEPG Composition

Sponsorship is a commonly cited reason for why improvement efforts succeed or fail. Both AMSS and STS addressed the issue of sponsorship by carefully crafting their SEPG. The first common element was SEPG leadership.

> *"Including senior managers and program executives in hands-on roles in the improvement effort was the first key to securing project participation."*

Senior managers (i.e., a manager of other managers) or executive managers (i.e., project managers) were appointed to chair the SEPG and were allowed time to do so as a part of their job descriptions. The STS software engineering manager chaired the STS SEPG with membership that included all project software managers, key leads, key domain experts, and process focal points. For the AMSS SEPG, an executive manager was appointed chairperson, with membership that included the chief software engineers from each project, domain experts, and process focal points. In both cases, managers accounted for 33 percent to 50 percent of total SEPG membership.

In addition, AMSS had a steering committee that included program executives and business unit functional managers. The steering committee's role was to establish AMSS objectives, commit resources, and monitor the progress of the AMSS SEPG.

Including senior managers and program executives in hands-on roles in the improvement effort was the first key to securing project participation. SEPG leadership activities went beyond declarations of intent, writing policy, and attending weekly meetings. SEPG leaders and members had a stake in the outcome of the improvement efforts because it affected products they had responsibility for producing.

Additional benefits included the following:

- The individuals who were accountable for producing the overall system set the process improvement goals, which aligned improvement efforts with the product and business needs, not a model.
- The individuals who had responsibility for building the software dictated what the processes were and how they should be used. There was a clear relationship between a process and its impact on day-to-day work.
- The managers who had authority over budgets and personnel resources put plans into action. As a result, improvement efforts were an integral part of the mainstream software development activities rather than activities isolated from the business concerns.

Improvement efforts were closely tied to the bottom line; the people making the decisions were accountable for the products being produced by the processes.

Staffing the SEPG in this way established the group with responsibility, authority, and accountability. Oftentimes, SEPGs are staffed with individuals who have the responsibility for managing the improvement effort but have little or no actual authority to change behavior, no personnel or budget authority, and no

direct accountability for cost, schedule, or quality. The key factor that separated the Boeing AMSS and STS organizations was the insistence that SEPG leaders and members have a stake in the outcome that was directly tied to the bottom line of the projects within the organization.

Another common practice on both teams was the commitment to charter the SEPG based on continuous process improvement, not to simply achieve a CMM level. Gary Wigle and George Yamamura described this and other successful practices of the STS SEPG in their 1997 article, "Practices of an SEI CMM Level 5 SEPG" [2]. The position of STS was that chartering an SEPG based on continuous improvement would align the business case for improvement with the activities of process definition, process change management, technology insertion, process evaluation, training, process improvement support, and regular assessments. A SEPG charter based on the business needs of the organization, combined with personnel that have the authority to make changes, were key to the success of STS.

AMSS adopted a similar philosophy early on by selecting the STS charter as its model. This approach to SEPG composition and function provided the foundation for understanding the business case for high maturity practices, and assured that key stakeholders were involved.

This approach has proven successful on two separate occasions in two completely different business units. The fact that the model proved successful in completely different domains, with different personnel involved, and with approximately six years separating the two efforts underscores the value of chartering and staffing an SEPG in this manner.

A Strong Tie to the Business Case

The second key factor was developing an understanding of what areas were critical to producing successful, high-quality products in order to focus improvement efforts. In the case of STS, the Inertial Upper Stage (IUS) project had a business objective to achieve a 100 percent mission success rate. Over time, they developed a clear understanding of the relationship between key development practices and mission success. Defect prevention was strongly emphasized long before the CMM was ever published because software errors during flight could rapidly lead to mission failure.

For AMSS, the Boeing F-22 project had established a history of outstanding product quality, cost, and schedule performance. Maintaining a careful balance of cost, schedule, and quality performance while reducing cycle time has become a primary business objective. Before either organization made an effort to achieve Level 5, the underlying business goals and criteria for success were established and communicated to everyone in the organization. Consequently, processes and metrics were inherently aligned to provide the necessary insight into critical processes and product quality.

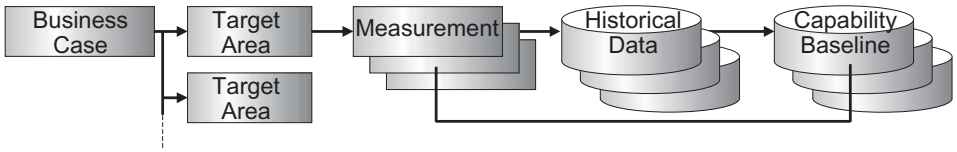


Figure 1: *Business Case Driven Approach*

As an example, the former Military Aircraft and Missiles Business Unit used five balanced measures in the areas of cost, schedule, quality, cycle time, and inventory/backlog. Four of these five were deemed relevant to software and flowed down to the AMSS organization (cost, schedule, quality, and cycle time). At the project level, data supporting these four areas had already been collected and used for a number of years. The data were researched and analyzed to establish historical baseline capabilities for process and quality. This established a quantifiable understanding at the project and organization levels in areas that were already tied to the business case of the organization and the business unit. This also provided a commonality among metrics in use by all projects of the organization (see Figure 1).

> *"... the pursuit of high maturity practices was rooted in a quantifiable understanding of the impact improvement efforts would have on the bottom line."*

In both cases, the pursuit of high maturity practices was rooted in a quantifiable understanding of the impact improvement efforts would have on the bottom line. In fact, the application of the CMM-based approach had little to do with the CMM itself; rather, it was the act of putting these practices into place that eventually improved the project in terms of cost, schedule, and quality. Most importantly, that understanding was shared with senior managers and executives who were accountable for mission success.

Project Culture and Historical Data

The existence of historical data that had been consistently collected over several years was the final contributing factor to achieving Level 5 in such a short time. In both cases, the organizations valued a data-driven approach to software management; both had used a consistent set of indicators for a number of years. Each organization had at least one project (IUS for STS, and Boeing F-22 for AMSS) that, through the necessity of meeting business and mission objectives, had long since initiated a data-driven approach to software management.

Making the transition from Level 3 to Level 5 amounted to taking what the managers understood as intuition, experience, and instinct and adding the quantitative understanding as revealed through analysis of the historical data. Managers on both IUS and Boeing F-22 had been making mental quantitative interpretations of the data for a number of years. Providing a historical context based on statistical analysis was a logical extension to the existing mindset of the software managers. Deployment was further accelerated by the fact that a quantitative understanding of process and quality was introduced in areas where data had been collected and analyzed for a number of years. While the presentation and usage of the historical data was new, the practice of collecting, reporting, and acting on the data had been long-since established.

For both organizations, introducing a quantitative understanding of the data in use was treated as an extension of an existing practice, not a new practice. Once this technique was understood and the benefits quantified, the value of applying this practice to other areas of software development became obvious. Each organization prioritized its efforts based

on the data that directly tied to its core business needs. This resulted in a quantitative metric set that included eight to 12 metrics distributed across process and quality.

Both organizations used metrics relating to cost and schedule that included earned value, cost and schedule variance, budget, actuals, and product release performance. STS used several defect-related measurements that ranged from defect removal during peer reviews to defect profiles (quantity and density) for each software product with clear traceability to ongoing defect prevention efforts. AMSS also used defect density measurements but with an increased emphasis on cycle time, and used measurements such as build cycle time to help manage improvements without sacrificing cost or quality.

Using eight to 12 metrics proved to be both meaningful and manageable. Some organizations can fall into the trap of trying to produce more Level 4 metrics than can possibly be used in an effective manner. In addition, new metrics may get invented that provide interesting information, but have no real significance in understanding the things that are vital to running the business. Both AMSS and STS avoided that trap because of SEPG composition and the tie to the business case. The individuals who used the metrics dictated what areas were meaningful.

Summary

The combination of hands-on participation by senior managers and executives, a clear tie to the business case, and the availability of mature historical data all contributed to making a rapid and almost intuitive transition to the high maturity practices. A common reaction from achieving Level 5 by both STS and AMSS was that it validated long-standing business practices that had been refined and elevated as best practices. Instead of having to overcome the not-invented-here syndrome, members of each organization were proud to say these processes were invented here.◆

References

1. Software Engineering Measurement and Analysis Team. Process Maturity Profile of the Software Community 2001 Mid-Year Update. Pittsburgh: Software Engineering Institute. Carnegie Mellon University. Aug. 2001.
2. Wigle, Gary B., and George Yamamura. "Practices of an SEI CMM Level 5 SEPG." CrossTalk Nov. 1997.

Note

1. Both Space Transportation Systems (STS) and Boeing Military Aircraft and Missiles Seattle Site (AMSS) assessments were conducted using the CMM-Based Appraisal for Internal Process Improvement method with external lead assessors from the Software Engineering Institute, STS, and Q-Labs (AMSS).

About the Author

Gregory P. Fulton is currently the software process improvement lead for Boeing's F-22 program, and Capability Maturity Model® Level 4/5 focal point for the Aircraft and Missiles Seattle Site organization. Fulton has 12 years of software development and process improvement experience, including an assignment as Space Transportation Systems Software Engineering Process Group lead. He is a former Air Force officer with seven years active duty experience. Fulton has a bachelor's of science degree in computer science from the University of Portland and a master's of science degree in computer science from the University of Nebraska at Omaha.

Boeing Integrated Defense Systems
F-22 Program
P.O. Box 3707, MC 43-14
Seattle, WA 98124-2207
Phone: (206) 544-1674
Fax: (206) 662-3301
E-mail: gregory.p.fulton@boeing.com

A Web Repository of Lessons Learned from COTS-Based Software Development[1]

Dr. Ioana Rus, Dr. Carolyn Seaman, Dr. Mikael Lindvall, Dr. Victor Basili, and Dr. Barry Boehm
Center for Empirically Based Software Engineering

By their natures, commercial off-the-shelf (COTS) software development and in-house software development are very different. Building a body of knowledge of lessons learned in COTS-based development would be very beneficial. Thus the authors have built a Web-based repository of such lessons learned for free use.

The development of commercial off-the-shelf (COTS)-based software is different in many respects from in-house software development. Since COTS requires different activities and skills, we need to build a body of knowledge about COTS-based software development. Thus, the authors have built a Web-based repository of lessons learned, seeded with about 70 lessons extracted from literature, including journal articles [1], workshop presentations [2], and government reports [3, 4]. The authors also organized online eWorkshops [5] and are using these discussions to synthesize new lessons and refine existing ones[2]. They are also consolidating the repository with an unpublished set of lessons learned from the Software Engineering Institute.

The lessons are described in the repository by a set of attributes, the most important describing the context in which the lesson was learned and could be applied (such as type of system, type of company, number and type of COTS). Other attributes refer to type of data (qualitative or quantitative), recommended audience, or pertinent life cycle phase. Most of the attributes were chosen based on a bottom-up effort to characterize and differentiate the lessons learned in the initial repository. Others were added simply because they seemed to reflect issues of interest to potential practitioner users (e.g., impact on cost, quality, and schedule).

Users can interact directly with the main components of the system, the COTS Lessons Learned repository and browse or search and retrieve lessons based on text searches over all attributes. Users are also encouraged to contribute to the community experience by using the online submission form provided on the main page, available at: <http://fc-md.umd.edu/ll/index.asp>.

User feedback is encouraged. Examples of useful feedback are: "this lesson applies differently in my environment because ... ," or "I experienced the same situation in a similar project," etc.

The feedback and new lessons go first to a buffer, and are examined and validated before being uploaded to the repository. An administrator maintains the repository, and an analyst is responsible for the repository's evolution. A component of the system (currently under development and available soon) will allow dialogues between users and experts, providing concrete support for problems. The logs of these dialogues will be captured and used for extracting new lessons.

For guidance on the use of the repository, there is a set of frequently asked questions (FAQs) accessible from the main page.

The repository's content is growing organically by contributions from users and as a result of analysis, synthesis, and refinement of the existing lessons by experts. The attributes used to characterize and classify the records will also evolve over time. The repository has a built-in facility for tracking various metrics related to the repository's usage, which can be used to tune the repository based on usage patterns.

The repository is available at no cost at: <http://fc-md.umd.edu/ll/index.asp>.◆

References

1. Basili, Victor, and B. Boehm "COTS-Based Systems Top 10 List." IEEE Software. May 2001: 91-93.
2. Fox, Steve, and M. Moore. "EOSDIS Core System (ECS) COTS Lessons Learned." 25th Annual NASA Goddard Software Engineering Workshop, Nov. 2000. Available at <http://sel.gsfc.nasa.gov/website/sew/2000/SEW25_final_program.htm>.
3. Albert, C., and E. Morris. Commercial Item Acquisition: Considerations and Lessons Learned. Available at: <www.dsp.dla.mil/documents/cotsreport.pdf>.
4. Lewis, Patrick, P. Hyle, M. Parrington, E. Clark, B. Boehm, C. Abts, R. Manners, and J. Brackett. "Lessons Learned in Developing Commercial Off-the-Shelf (COTS) Intensive Software Systems." FAA SERC Report, 2001.
5. Basili, Victor, R. Tesoriero, P. Costa, M. Lindvall, I. Rus, F. Shull, and M. Zelkowitz. Building an Experience Base for Software Engineering: A report on the first CeBASE eWorkshop. Proceedings of the 3rd International Conference on Product Focused Software Process Improvement. PROFES2001. Kaiserslautern, Germany, Sept. 2001.

Notes

1. This work is partially sponsored by the National Science Foundation grant CCR0086078, <http://cebase.org>, to the University of Southern California and the University of Maryland, with subcontract to the Fraunhofer Center, Maryland.
2. Many thanks to our CeBASE research partners from University of Southern California, Chris Abts and Dan Port, for their help in gathering the published lessons and running the first COTS eWorkshop.

About the Authors

The authors are members of the Center for Empirically Based Software Engineering (CeBASE), a National Science Foundation-funded project. Dr. Ioana Rus (irus@fc-md.umd.edu) and Dr. Mikael Lindvall (mikli@fc-md.umd.edu) are scientists at Fraunhofer Center for Empirical Software Engineering Maryland (FC-MD). Dr. Victor Basili (basili@fc-md.umd.edu) is the director of this center and a professor at University of Maryland. Dr. Carolyn Seaman (cseaman@umbc.edu) has a research position at FC-MD and is also an assistant professor at University of Maryland, Baltimore County. Dr. Barry Boehm (boehm@usc.edu) is a professor at University of Southern California.

TSP: Process Costs and Benefits

Jim McHale
Software Engineering Institute

The Team Software Process℠ (TSP℠), like other process improvement paradigms, is often challenged on the grounds that it adds overhead to already burdened developers. However, the TSP's overhead is readily quantified and justified by published results. One might also question the use of the term "overhead" when referring to necessary project tasks.

A question that often arises when attempting to convince a management team or an engineering staff to adopt the Team Software Process℠ (TSP℠), or any other process improvement program is: "How much overhead will it add?"

This is generally not an easy question to answer since it is unlikely that the questioner knows how much overhead, beyond perhaps a general accounting figure, is associated with an organization's current practices, particularly at the project-team level. Thus, the question of how much overhead will be added cannot be answered since it is not known how much overhead there was in the first place; a large part of that team-level overhead will likely be replaced by the TSP. Also, it seems somewhat ironic that this question arises from the same mindset that trims administrative and support staff, allowing managers and developers to answer their own phones and make their own copies, which seems like true overhead indeed.

For the sake of argument and part one of this article, let us suppose that the question is legitimate at face value. Part two of this article will address a few of the relevant benefits of using the TSP. Finally the question of process overhead itself will be examined.

Part One: The TSP Overhead

We need a counting standard to begin the analysis of the amount of the TSP overhead. We will assume 52 weeks per year, five working days in a week, and eight hours per day for a 40-hour week. That gives us 260 days or 2,080 hours per ideal developer-year.

The TSP calls for an all-hands team planning session (called a launch) that lasts for four days at the beginning of a project. Re-launches of up to three days each happen every three or four months. Let us assume that a full four-day launch happens annually, and a full three-day re-launch every quarter thereafter. This totals 13 days per developer, exactly 5 percent of the ideal developer-year.

At the end of every launch phase, the TSP calls for a post-mortem meeting to consolidate the data gathered and compare it against the launch estimates. The post-mortem also allows the team to figure out how the processes that they used helped or hindered in getting the job done, and to identify process adjustments to be implemented the next time. Post-mortems should last a day or less. Let us assume a full day for the entire team once a quarter, or four days per year. This is slightly more than 1.5 percent of a developer-year.

Weekly status meetings are also required. If the TSP team is reasonably efficient in running its meetings, most teams of, say, 10 to 12 people, can conduct them in an hour or less. Smaller teams can take less time, and larger teams can take a little longer, but if a meeting is running more than 90 minutes, either the team needs some training in meeting facilitation, or the team is just too big. Weekly meeting time equals one hour per week on average, or 2.5 percent of a developer-year.

> *"If you question the value of removing defects early via inspection, you should not even consider using the TSP or any other improvement paradigm based on CMM principles."*

The amount of time spent gathering data for the TSP is often questioned, so let us examine that. An entry in the time log, if done by hand on paper, takes about 15 seconds, counting start time, stop time, and interrupts as a single entry on one line. A person probably makes between 10 and 12 entries per day on average. It is a lot less tedious if you are using the SEI-supplied TSP tool or a freeware/shareware program, and there are some nice time-tracking programs available for your favorite hand-held device. Some organizations have developed their own tools for this purpose. [In case you were wondering, the Software Engineering Institute (SEI) does not care which tool you use, as long as you report specified summaries of the gathered data back to the SEI.]

If you are using anything besides a tool that allows direct consolidation with the rest of the team's data, it should take about five minutes to transfer your time log entries daily. (If you wait until the end of the week, it tends to be tedious and inaccurate. Do not do that!) Set transfer time at 10 minutes a day, 12 minutes to make the math easy. Five days times 12 minutes per day equals one hour per week. That is another 2.5 percent, but it is that high only if you do it by hand first and then transfer, less time otherwise.

Estimating the time spent logging defects can be tricky. Defects found in personal reviews or team inspections tend to take less time to log since, by definition, these are for things that you are looking for specifically. Set 30 seconds or less to log a defect found in reviews/inspections or by the compiler, which after all is telling you what the defect is. In integration and test phases, admittedly it can take a little longer to log a defect, but since you have relatively few of these (due to the great job you did in your reviews and inspections), the time spent here should not be too onerous. Even at two minutes per defect, that is plenty of time. On average, my informed guess is about one minute to log a defect. At 10,000 lines of code (LOC) per programmer-year (a fairly productive person) and an average of 100 defects per 1,000 LOC (KLOC), that is about 1,000 defects or about 1,000 minutes. To make the math easier, let us round up to 1,200 minutes or 20 hours per year, or slightly less than 1 percent overhead attributable to defect data gathering.

The time spent on the TSP role manager tasks is difficult to estimate, in part

because the actual duties of each role are very idiosyncratic to a particular team on a particular project in a particular organization. SEI guidance is for one to two hours per week. Two hours a week seems fairly high as an average weekly value, but even at that we are talking about another 5 percent overhead.

This analysis does not accept the premise that personal reviews or team inspections should be treated as overhead. If you question the value of removing defects early via inspection, you should not even consider using the TSP or any other improvement paradigm based on Capability Maturity Model® (CMM®) principles. CMM was developed as an instantiation of total quality management methods applied specifically to software development [1]. The basic tenet is that it is generally faster and cheaper to find defects earlier in the process rather than later. Also, if reviews and inspections are done properly, most defects should be found at that time and logged when they are fixed, and we certainly should not count defect-logging time twice. Therefore, in the TSP implementation of CMM principles, reviews and inspections are an integral part of the process, and not overhead.

The overhead numbers are summed up in Table 1. Is 17.5 percent a lot? I can't say. I contend that the question is irrelevant unless you know what you get in return.

Part Two: The TSP Upside

The most recently published example that I can find on the benefits of using the TSP is from a Honeywell presentation at the 2002 Software Engineering Process Group conference [2]. Pavlik and Riall claim a better-than-70-percent software productivity increase from one release of an avionics control system to the next, with a total of 22 percent savings in total systems and software effort.

While their delivery was on time, even more significant is that the quality of their delivered product was 10 times better than the previous release, while the delivered functionality was three times what was originally planned. I would say that a 17.5 percent process overhead for the TSP was more than worth it to Honeywell.

At the same conference, a presentation by John Ciurczak of EBS Dealing Resources claimed a "37.5 percent reduction in execution stage cycle" [3]. The execution stage, in this instance, refers to the time from when the business case for the project was approved, until the time that the product was tested and ready to ship. That reduction was entirely attributable to fewer defects in the EBS certification test phase, which is required for the foreign currency exchange services that EBS provides. Ciurczak also showed that the development activities prior to certification testing took just about as long with TSP practices as without. EBS probably cannot decide if that 17.5 percent was overhead, or just a normal and acceptable cost of doing a project.

In 2000, Don McAndrews of SEI published a summary of early results of using the TSP and its companion technology, the Personal Software Process℠ [4]. Unfortunately, productivity was not one of the numbers available for comparison. However, the before-and-after comparisons for cost and schedule deviation, defect density in test, and system test time per KLOC are usually cited by me and my SEI colleagues to ask a different question: "Can you afford not to implement TSP, overhead and all?" Certainly, the other numbers cited above leads one to the same question.

"TSP overhead performs necessary project functions regardless of the size of the organization ..."

Part Three: The Upside of Overhead

Finally, let us look at this accounting fiction called *overhead*. The battered dictionary that lives on my desk defines it this way: "*overhead* n.: business expenses not chargeable to a particular part of the work" [5].

I like this definition for a couple of reasons. First, it justifies my earlier tirade on not counting reviews and inspections as overhead, since one clearly must be inspecting a "particular part of the work." Second, it arguably removes the defect logging time from the calculations since that too can be attributed to particular parts of the job. The same argument applies for most of the time logging, since time usually is logged only against tasks that are traceable back to a specific part of what the TSP team is building. But for the continued sake of the argument, let us not fiddle with the 17.5 percent number, even though it may be high by a few percentage points. Let us talk instead about a subtle connotation of the word overhead.

Launches and re-launches	5.0%
Post-mortems	1.5%
Weekly team meetings	2.5%
Time logging	2.5%
Defect logging	1.0%
Role manager tasks	5.0%
Total	**17.5%**

Table 1: *Total Overhead Amounts*

In current usage, overhead conveys the sense that work so charged is somehow not entirely necessary. Is planning unnecessary? What about evaluating the effectiveness of the resulting plan during the last three months and figuring out how to do better the next time? What about knowing on a weekly basis just how effectively that plan is guiding the work, and how much of the work you have actually accomplished? What about gathering the data necessary to make these judgments about the plan? What about having the data available to evaluate your own performance objectively, and to deploy the team's resources most efficiently? These are exactly the purposes of the launches, post-mortems, weekly meetings, and data-gathering activities of the TSP.

Is it necessary for a project team to keep track of changing requirements? What about having common standards for design representation and coding? What about maintaining a view towards testing the system throughout the life cycle? These are all activities for some of the TSP role managers.

I have seen organizations create entire staffs to plan or to evaluate effectiveness, or to track time and defects, or to perform many of the other functions questioned above. While having such *overhead* positions may indeed be necessary in a particular organization, I like the TSP overhead model, which features the people who are doing the work also doing that little bit extra that actually helps them do the work more effectively and efficiently. TSP overhead performs necessary project functions regardless of the size of the organization, and can also help the people responsible for planning, evaluating, and tracking do their own jobs more effectively by freeing them to deal with the cross-project and other organizational issues that they are intended to address.

On balance, I prefer the overhead that the TSP brings to the table. A project team managing and measuring its own work against its own commitments, and getting results like those previously cited, seems to have no need to apologize for its practices or to justify what percentage of the

time they take to execute.◆

Acknowledgements

My thanks to Jim Porter of Tyco Electronics whose e-mail initially launched part of the preceding rant. My teammate on the TSP Initiative team at the SEI, Marsha Pomeroy-Huff, also provided the odd prod or two and edited my composition perhaps a little too gleefully. Finally, Anita Carleton of the SEI provided just the right push at just the right time to get me started, which is always the hardest part.

References

1. Paulk, Mark, B. Curtis, M. Chrissis, and C. Weber. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.
2. Pavlik, Rich, and C. Riall. Integrating PSP℠, TSP℠ and Six Sigma at Honeywell. Software Engineering Process Group 2002 Conference Proceedings (CD-ROM). Carnegie Mellon University, 2002.
3. Ciurczak, John. The Quiet Quality Revolution at EBS Dealing Resources, Inc. Software Engineering Process Group 2002 Conference Proceedings (CD-ROM). Carnegie Mellon University, 2002.
4. McAndrews, Donald. The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices. CMU/SEI-2000-TR-15. Carnegie Mellon University, 2000.
5. The New Merriam-Webster Pocket Dictionary. Simon and Schuster, 1971.

About the Author

Jim McHale joined the Software Engineering Institute in 1999. He has more than 20 years of experience, mainly in real-time control and supervisory systems in the transportation, steel, plastics, machine tool, and power generation industries. He has acted as software engineer, systems engineer, hardware engineer, project leader, and product manager. Since 1996, McHale has been a Capability Maturity Model® (CMM®)-Based Appraisal for Internal Process Improvement assessment team member, Software Engineering Process Group member, Personal Software Process instructor, and Team Software Process℠ (TSP℠) launch coach. Currently he is focusing on using the TSP to accelerate CMM-based improvement, and on adapting the TSP to enhance the effectiveness of other SEI initiatives, including commercial off-the-shelf systems. McHale has a bachelor's degree in electrical engineering from the University of Pittsburgh.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Ave.
Pittsburgh, PA 15213-3890
Phone: (412) 269-3948
E-mail: jdm@sei.cmu.edu

LETTER TO THE EDITOR

Dear CROSSTALK Editor,

I just wanted to thank the CROSSTALK staff and the authors for the April 2002 edition "Risky Requirements." I read every article with diligence. This is a very important issue that focuses on the most important piece of the software puzzle, software requirement.

Yet, in my opinion, it is the most neglected. You may have the best of everything: management, technical staff, resources, budget, schedule, customers, and even CMM® Level 5 processes. But, if you do not have a good set of well-defined validated requirements that are understood and agreed to by all stakeholders, you have absolutely NOTHING.

Did I say nothing? Well, you do have something. You have a whole lot of something: re-work, missed schedules, low quality, failed projects, irate management, customer dissatisfaction, canceled projects and additions to failure statistics.

We all know that there are no silver bullets for software development. But, if you have a good set of well-defined, validated requirements that all stakeholders can drive a stake into in the early stages of development, then that is the closest you will come to that silver bullet. Everything else will fall easily into place for the remainder of the life cycle. I know. I have experienced both phenomenons.

Al Florence
MITRE Corp.

Experts Present Views on 21ST Century

Pamela Bowers
CrossTalk

Software engineering research moves the industry forward. Some of this research was presented recently at the International Conference on Software Engineering (ICSE) in Orlando, Fla. This article reports on keynote talks at the ICSE 2002.

Technology continues to shrink our world. Today's customers know everything about products; almost every business has a Web storefront, said Jim Cassell, group vice president at Dataquest Research at the 2002 International Conference on Software Engineering.

Global competition causes business to become very customer centered, said Cassell, including operating and responding in real time to customer desires. "We have to have dynamic, interactive, collaborative interaction in the supply chain to have zero latency enterprise (ZLE)," he said. While ZLE technology is in place, the decision-makers in business have not adopted it, he noted. "There is no standard of communication among parties in the supply chain ..."

Ubiquitous computing is the solution, according to Cassell. "Operators are looking to operate 24 hours a day, seven days a week. They must be able to accept a wide variety of computing supplies, including laptops, phones, palm pilots, etc.," he said. "We need to reduce inefficiencies by 'virtualizing' resources, setting standard operating parameters, switching workloads, and resource management."

Meanwhile, 21st century systems engineering demands robust use of the systems approach, said Donna H. Rhodes, director, Process and Quality at CSG Systems. Given the challenges of this century, Rhodes said, systems engineering must be an essential engineering discipline. As it becomes a more integral part of product development, the character of the systems engineering discipline expands, and the associated research agenda takes new shape, she said. While software engineering and systems engineering share many methods and practices, each has a different world view.

Bob Balzer, chief technical officer of Teknowledge Corporation said researchers should focus more on commercial off-the-shelf (COTS) and on assisting users vs. developers. Opportunities include wrappers to add user functionality and safety, he said. New tools or add-ons understand what the user is doing within a COTS tool, and infer user goals in order to provide usage guidance and scripting assistance, he said.

Balzer cited user-programmable extensions like Safe-Email in which each opened attachment spawns a new process that is wrapped for safety, and Editor, which allows users to make late authorization decisions by transparently redirecting operations to a virtual system. He also mentioned integrity-marked documents that build a history of all changes made to a document. "There is a lot of opportunity to make use of the COTS function that is out there."◆

Using the TSP to Implement the CMM

Noopur Davis
Software Engineering Institute

Organizations using the Capability Maturity Model® for Software (SW-CMM®) to guide their software process improvement efforts often struggle with implementation details. The Team Software Process℠ (TSP℠) was designed to implement high maturity processes for projects. This article examines the relationship between these two complementary technologies by analyzing the degree to which the CMM is addressed by the TSP. An overview of the relationship between the TSP and the CMM is presented first. This is followed by a description of how the TSP addresses each CMM key process area.

The Capability Maturity Model® for Software (SW-CMM®) is a descriptive model of the characteristics of an organization at a particular level of software process maturity [1]. The Team Software Process℠ (TSP℠) is a prescriptive process for projects. It contains an adaptable set of processes, procedures, guidelines, and tools for projects to use in producing high-quality software on time and on budget. It also includes an introduction strategy for building management sponsorship, training managers and engineers, and coaching and mentoring TSP practitioners.

The CMM and the TSP are complementary by design [2, 3, 4]. After guiding the development of the CMM, Watts Humphrey went on to develop the TSP as a way to apply CMM principles at the individual and project levels [5]. The CMM describes what an organization at a particular maturity level should be doing, while the TSP prescribes how high maturity practices are implemented at the project level.

This article explores the relationship between the TSP and the CMM by describing how the TSP addresses each key process area of the CMM, and by showing the number of CMM key practices that are addressed by the TSP. Further details about the relationship between the CMM and the TSP are available in a Software Engineering Institute (SEI) technical report [6].◆

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

From Performance-Based Earned Value to the CMMI

Paul J. Solomon
Northrop Grumman Corporation

Earned Value Management (EVM) can be a process thread to enable effective process integration and improvement during transition to the Capability Maturity Model® Integration℠ (CMMI℠). Organizations that already use EVM can reduce their transition costs and increase the effectiveness of EVM by following the guidance in this article. Other organizations should consider implementing EVM during transition to the CMMI. Quantitative project management, with the effective use of performance-based earned value, will reduce the risk of failing to achieve a project's cost, schedule, and technical objectives.

Many organizations are planning to transition their framework for process improvement efforts from the Capability Maturity Model® for Software (SW-CMM®) to the Capability Maturity Model® Integration℠ (CMMI®). The CMMI is generally consistent with the guidelines of the primary external industry benchmarks for Earned Value Management (EVM), including the Electrical Industries Association standard EIA-748-A, "Earned Value Management Systems" [1] (EVM standard). However, the CMMI is more stringent than the EVM standard regarding objective measurement and more focused on requirements.

Those organizations that already use EVM can develop efficient process improvement plans and minimize transition costs, including appraisal costs if they utilize the relationships between the CMMI and external industry benchmarks, and address the gaps between their EVM practices and CMMI goals.

Other organizations should consider implementing EVM as a process improvement. In the CMMI context, EVM is a process thread that crosses many discipline boundaries and is critical to effective process integration. Consequently, implementation of EVM during the transition will be more efficient if it is part of an overall plan to improve and integrate processes.◆

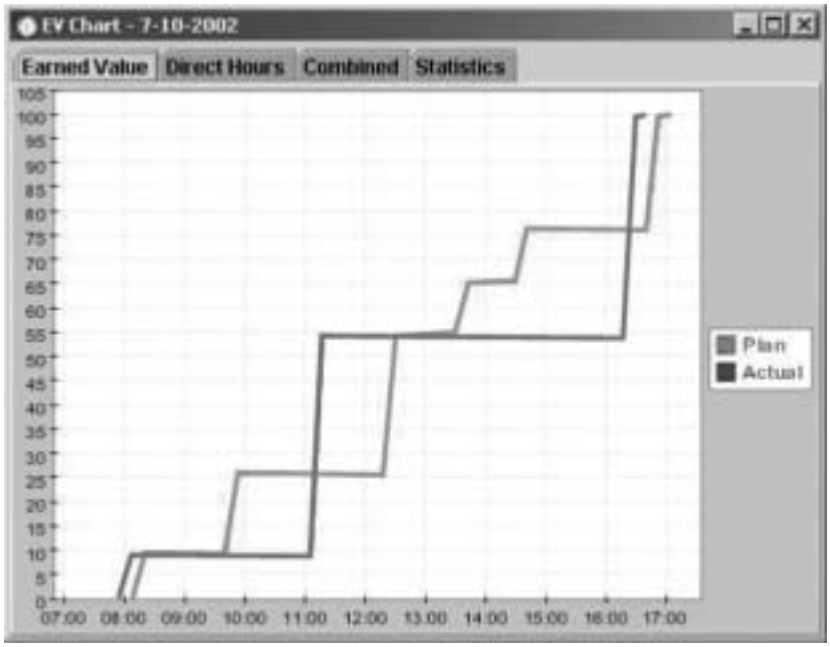
Due to space constraints, CrossTalk *was not able to publish this article in its entirety. However, it can be viewed in this month's issue on our Web site at <www.stsc.hill.af.mil/cross talk> along with back issues of* CrossTalk.

PSP Addicts Anonymous

Hello. My name is David Webb, and I am a PSP[SM] addict.

PSP is the acronym for Personal Software Process[SM], developed by Watts Humphrey. It is a software engineering discipline that trains software professionals to plan, track, and improve themselves like marathon runners. I am addicted to it. Let me give you some examples of my addiction.

I plan everything. Note, everything! I plan my days, each and every day, in great detail. I identify every meeting I'm to attend, every document I'm to produce, and all the activities I'm to perform. I estimate the relative size of each event (from very small to very large). I assign a time value to each of the tasks based upon my historical data for the type of task and its relative size. (A medium-sized meeting, for example, is typically 60 minutes long.) Then, I put the tasks in order and create an earned value tracking plan for my day. As I work the tasks, I record the actual time in minutes it took me to complete each one. Every time I do this, I update my actual earned value and compare it to my plan. I then adjust my plan throughout the day to account for unexpected events.



David Webb's Daily Earned Value Chart

At the end of the day, I conduct a postmortem to record my historical data, compare my estimates to my actual times, document any lessons learned, and prepare the tracking form for the next day.

Impressed? Scared, maybe? Wait, there's more!

Another symptom of my addiction is that I have a documented process for everything. Of course, I have a process for creating software and for planning and tracking my day, but I also have a process for conducting meetings, writing meeting minutes and documents, updating project metrics, and all the routine things I do throughout the week. I even have a process for balancing my checkbook.

Oh, yes. This addiction extends to my home life as well. My family has gotten used to seeing charts on the wall. Weight loss and family budget charts are the most common, but I'm currently working on a weekly earned value chart for household chores and another to track savings toward our summer vacation. I have to hide these charts in the bathroom where most visitors won't see them. (PSP addicts are not well understood by nonbelievers.)

Which brings me to this question: Why do folks take an instant and extreme disliking to those three little letters? PSP addicts often suffer from open hostility. For example, people have stormed out of my PSP classes (Did I mention I'm also an instructor?), started shouting matches over it, and have even called me a liar. Last month, a well-respected software professional called the PSP practice of tracking to the minute "#*@%-ing ludicrous!" He then spent several minutes telling me why it was ludicrous and why the use of the expletive was the kindest possible way to put it.

I think a major reason people dislike it is that PSP is threatening to their way of life. Outside of the sports world, people generally do not like measuring themselves. (When was the last time you really looked at the bathroom scale?) A basic assumption of PSP is that we must measure ourselves. Another PSP tenet is that current methods for producing software are not only inadequate, they are dangerous. This does not sit well with people who have been working that way for a couple of decades. PSP is also a direct departure from the prevalent hack-and-slash culture and treats the creation of software not as an art form, but as a discipline.

PSP requires us to be marathon runners, to know the length of the track we're running and to use a stopwatch to time ourselves. Some people feel that this takes the fun out of coding. PSP addicts find the fun in meeting a schedule and producing code that compiles and tests perfectly the first time. This attitude actually scares some people.

So, what do we do with weirdos like me who actually like PSP? (Oh, yes, there are others!) Do we really want to set up a booth at COMDEX to collect donations to eradicate this debilitating and unpleasant disease?

You know, there are advantages to working with PSP addicts: They give you accurate estimates, they let you know early on if those estimates need adjusting, they know their strengths, they know their weaknesses, and they know how to improve themselves. While it may seem that the overhead of PSP data collection will slow you down, PSP has been proven time and again to actually increase productivity.

So, rather than find a cure for PSP addicts, maybe we should send them to support groups. These groups already exist and, where properly implemented, are effective outlets for the PSP addict: Team Software Process[SM] (TSP[SM]) groups. These are teams of PSP addicts who work together to produce nearly defect-free code on time and under budget. And just like other support groups, TSP groups have weekly meetings. It's a good thing, too – finally we have someone to show our charts to!

— **David R. Webb**
Software Division, Hill Air Force Base

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

Published by the Software Technology Support Center (STSC)

CrossTalk / MASE
7278 4th Street
Bldg. 100
Hill AFB, UT 84056-5205

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