Carnegie Mellon
Software Engineering Institute

news @ sei

http://interactive.sei.cmu.edu

Editor's Message

Improvements to the licensing of SEI products and services are the focus of the article "The SEI Partner Network: Providing Authentic SEI Services and Training." Until recently, organizations licensed to deliver SEI products and services had been called Transition Partners, but the newly designated SEI Partner Network emphasizes the role of the SEI Partners as providers of official SEI-brand services. Through the changes detailed in the SEI Partner article, the SEI intends to help users of SEI technologies understand where and how they can get authentic SEI services to support their software engineering efforts. Users who buy SEI courses and services from an SEI Partner can be sure that they are getting the official version, from someone who has been trained and approved by the SEI.

As a part of its efforts to strengthen the SEI Partner Network, the SEI recently instituted a code of conduct that defines the ethical business practices it expects from its licensees, with the possibility of license revocation in the event of an investigated and confirmed infraction. At the time *news@sei* was heading to press, the code of conduct was being finalized. For current information on the code of conduct or to get a copy of the code, please visit www.sei.cmu.edu/partners/?si.

Many SEI Partners offer services related to process management and process improvement. Two examples of SEI work in these areas are showcased in this issue of *news@sei*. "Using Six Sigma in Software Development" describes how the SEI is investigating the ways Six Sigma can benefit software and systems development, as Six Sigma evolves from an improvement framework for the manufacturing sector to one that can be applied across all levels of an enterprise. Accelerating the adoption of the Capability Maturity Model® Integration (CMMI®) framework through use of the SEI Team Software Process℠ (TSP℠) is the focus in "TSP Accelerates Software Quality Improvements at NAVAIR."

Conferences are another important channel for delivering information about software engineering best practices, and a place for SEI Partners and others to share information. Process management was the focus at the 2004 SEI Software Engineering Process Group Conference (SEPG℠ 2004), which was just held in March in Orlando. (Look for an article about SEPG in the next issue of *news@sei*.) Another important conference co-sponsored by the SEI is the International Conference on COTS-Based Software Systems (ICCBSS 2004) which is summarized on page 37. Be sure to read this article if you weren't able to attend, and go to *www.iccbss.org* for more details and for information about next year's conference.

Janet Rex
Editor in Chief

:

Columns

The Architect

Integrating Architecture Methods: The Case of the ATAM and the CBAM

RICK KAZMAN AND ROBERT L. NORD

In a previous column (“Rethinking the Software Life Cycle”, Vol. 6, No. 3, Q3, 2003), we took a look at the traditional software development life cycle (SDLC) in the context of the architecture-centric methods that we have developed at the Software Engineering Institute over the past 10 years, including the Architecture Tradeoff Analysis Method, or ATAM, which helps a system’s stakeholders understand the consequences of architectural decisions with respect to the system’s quality-attribute requirements [1, 2].

The common wisdom is that the earlier in the SDLC that problems are found, the easier and cheaper it is to fix them. The problems that can be found by an architecture evaluation include unreasonable requirements, performance problems, problems associated with potential future modifications, and many others related to quality attributes. As we have gained experience from conducting ATAM-based architecture evaluation exercises, we have found the need to develop methods that extend even earlier into the SDLC. In this way, rather than simply reporting on problems following an architecture evaluation, we can anticipate these problems at requirements or design time. The Quality Attribute Workshop, or QAW, provides a method for eliciting quality attribute requirements [3]. The Attribute-Driven Design, or ADD, method provides a way of defining a software architecture by basing the design process on the quality-attribute requirements of the system [4].

Looking at later stages of the SDLC, the Cost Benefit Analysis Method, or CBAM, is a method for performing architecture-based economic analyses of software-intensive systems [1, 5]. The CBAM can be used to help the system’s stakeholders choose architectural alternatives during the design phase, but it can also be used to determine what architecture choices to make in enhancing the system, during the maintenance phase of the SDLC.

Although these methods share a common heritage and a common set of concepts and activities, they have not thus far been explicitly integrated with each other or integrated into common life-cycle models, such as the Rational Unified Process, or RUP [6]. Integration of this kind is essential

if organizations are to reap the benefits of adopting an architecture-centric approach to software development. We call this integration "architecture-centric development," as shown in Figure 1.

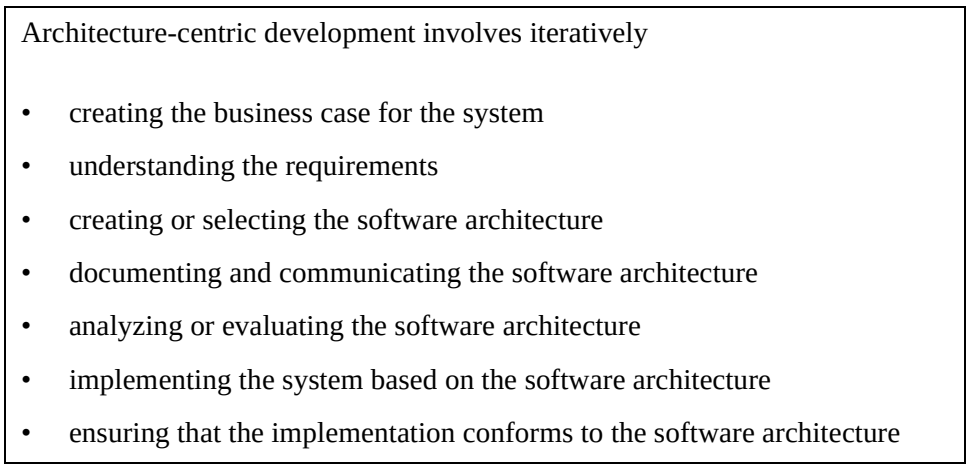

Figure 1: Architecture-Centric Development

We have already made some progress toward integrating architecture-based methods into the SDLC. For example, some organizations have applied more than one architecture-based method at different life-cycle stages—the ATAM followed by the CBAM, or the QAW followed by the ATAM, or the ADD followed by the ATAM. But the full benefits of this integration have not been realized.

For example, in our interactions with NASA's EOSDIS project over several years, we integrated the ATAM and CBAM activities. (EODIS is the Earth Observing System
Data and Information System, whose purpose is to acquire, archive, manage and distribute Earth observation data to a diverse group of users.) The results of applying the ATAM and the CBAM to the EOSDIS project have been documented, but each method is described separately [1, 7, 5]. The integration was piecemeal and opportunistic, rather than planned and organic.

Representations of the inputs, outputs, and participants of the ATAM and the CBAM are shown in Figures 2 and 3.

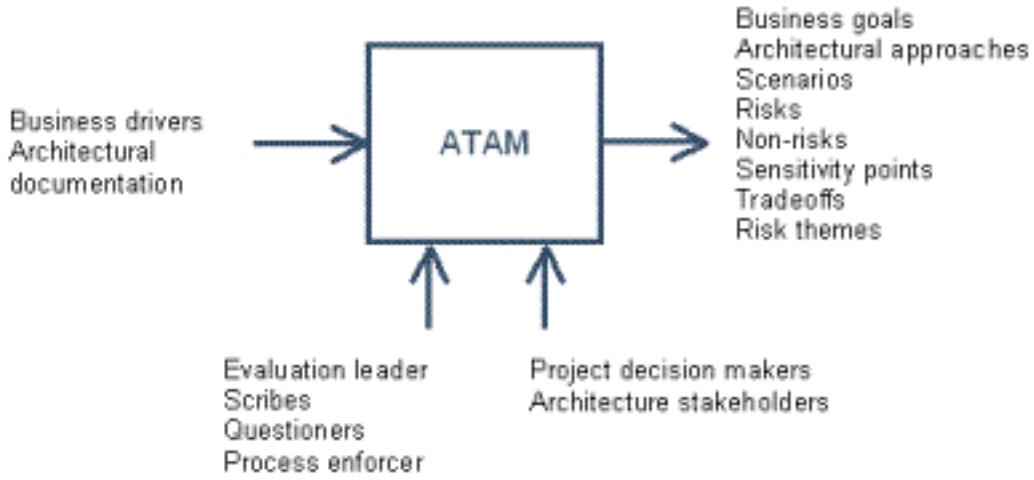

Figure 2: ATAM Inputs, Outputs, and Participants

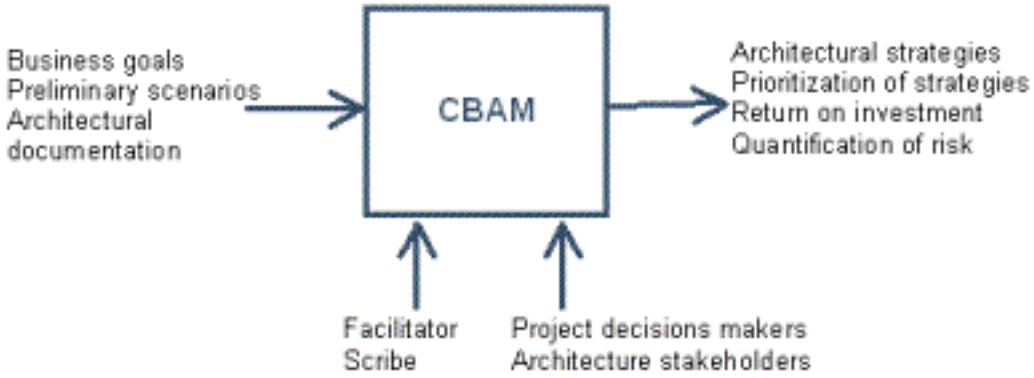

Figure 3: CBAM Inputs, Outputs, and Participants

A cursory observation shows that the two methods align well—many of the inputs, outputs, and stakeholders are aligned. But there are differences in the methods that prevent them from being simple to integrate.

In a forthcoming technical note [8] we address this by focusing on enhancements to the ATAM and the CBAM specifically designed for their integration within an SDLC process. The aim is to create a combined method that will help organizations design and manage the evolution of complex software-intensive systems by making a series of informed architecture-design decisions based on business factors.

The integrated ATAM/CBAM evaluates the software architecture and prepares for the next design iteration by considering new business goals and requirements. A CBAM exercise provides a

natural follow-up to an ATAM exercise for those wanting an integrated process that provides more informed results. This combined approach provides an integrated method for both technical and economic analysis of a software architecture and offers the following benefits:

- The integrated method optimizes the process. The results from the ATAM exercise can be used in the CBAM exercise and don't have to be duplicated. Several CBAM steps can be shortened or eliminated, saving time and requiring fewer meetings for stakeholders.
- The integrated method improves the quality of the results. The combined method is more capable and makes better use of the stakeholders' time than either method applied independently.
- The integrated method provides more guidance in eliciting artifacts. The integrated method more clearly differentiates the analysis of existing architecture versus development and analysis of new architectural approaches. It makes explicit the documentation needed to record new architectural decisions.

Clearly integration of architecture-centric methods with each other and with existing life-cycle models is the way of the future. Organizations cannot afford to develop an architecture that is less than technically capable, nor can they afford to make ill-advised or ill-timed economic decisions. Organizations need guidance in allocating their limited budgets for designing, implementing, maintaining, and augmenting a system over its lifetime. A combined ATAM/CBAM does just that, and does so better than either method could do on its own.

References

[1] Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice*, 2nd ed., Boston, MA: Addison-Wesley, 2003.

[2] Kazman, R.; Klein, M.; & Clements, P. *ATAM: Method for Architecture Evaluation* (CMU/SEI-2000-TR-004, ADA382629). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tr004.html>

[3] Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; Wood, W. *Quality Attribute Workshops (QAWs)*, 3rd ed. (CMU/SEI-2003-TR-016), 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>

[4] Bachmann, F.; Bass, L.; Chastek, G.; Donohoe, P.; & Peruzzi, F. *The Architecture Based Design Method* (CMU/SEI-2000-TR-001, ADA375851). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tr001.html>

[5] Kazman, R. Asundi, J. & Klein, M. *Making Architecture Design Decisions: An Economic Approach* (CMU/SEI-2002-TR-035). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports/02tr035.html>

[6] Kruchten, P. *The Rational Unified Process: An Introduction*, 3rd ed., Boston, MA: Addison-Wesley, 2003.

[7] Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*, Boston, MA: Addison-Wesley, 2002.

[8] Nord, R.; Barbacci, M.; Clements, P.; Kazman, R.; Klein, M.; O'Brien, L.; & Tomayko, J. *Integrating the Architecture Tradeoff Analysis Method with the Cost Benefit Analysis Method* (CMU/SEI-2003-TN-038) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. <http://www.sei.cmu.edu/publications/documents/03.reports/03tn038.html>

About the Authors

Rick Kazman is a senior member of the technical staff at the SEI, where he is a technical lead in the Architecture Tradeoff Analysis Initiative. He is also an adjunct professor at the Universities of Waterloo and Toronto. His primary research interests within software engineering are software

architecture, design tools, and software visualization. He is the author of more than 50 papers and co-author of several books, including a book recently published by Addison-Wesley titled Software Architecture in Practice. Kazman received a BA and MMath from the University of Waterloo, an MA from York University, and a PhD from Carnegie Mellon University.

Robert L. Nord is a senior member of the technical staff in the Product Line Systems Program at the Software Engineering Institute (SEI) where he works to develop and communicate effective methods and practices for software architecture. Prior to joining the SEI, he was a member of the software architecture program at Siemens, where he balanced research in software architecture with work in designing and evaluating large-scale systems. He earned a Ph.D. in Computer Science from Carnegie Mellon University. Dr. Nord lectures on architecture-centric approaches. He is co-author of Applied Software Architecture and Documenting Software Architectures: Views and Beyond.

CMMI in Focus

Myths about SCAMPI Appraisals

MIKE PHILLIPS

At the CMMI Technology Conference this past November, Dave Kitson, the SEI's chief architect of the SCAMPI V1.1 Method, presented "Ten Myths about SCAMPI." (He started with 10, but by the time he actually made his presentation, there were 14….) With minor elaboration and Dave's kind permission, I'm including them in this column, as the concerns they raise continue to be heard.

The myths are presented in no particular order, and were "discovered" without any particular scientific or statistical methodology. The myths originated from one or more of the following:

- SEI observations of SCAMPIs
- inquiries made to the SEI (phone, email)
- questions from students attending SEI classes (Intro to CMMI, Intermediate CMMI, SCAMPI Lead Appraiser Training)

Myth 1: Findings are required to be reported at the sub-practice level.

One of the difficult construction decisions in developing the CMMI was determining what level of inquiry was appropriate for CMMI guidance in appraisals. We determined that while goal satisfaction was "rated," (all goals in a process area [PA] must be "satisfied" in order for the PA to be "fully satisfied" for maturity level or capability level ratings) specific and generic practices (the next level down in the models) would be examined to check if they were being implemented – but subpractices would not be. However, findings *may* be reported at the sub-practice level if requested by the appraisal sponsor.

Typically, however, findings are reported as goal-level statements that summarize gaps in practice implementation. These statements must be abstracted to the level of the organizational unit, and cannot focus on individual projects (unless the option for project-specific findings has been agreed upon during planning).

Myth 2: The Appraisal Disclosure Statement (ADS) is optional.

The ADS is a required part of the SCAMPI method; it is typically prepared by the SCAMPI lead appraiser and must be provided to the appraisal sponsor as well as the SEI.

The SCAMPI lead appraiser affirms that the information in the ADS is accurate and that the appraisal was conducted in full accordance with the requirements of the SCAMPI V1.1 Appraisal Method and the provisions of his or her authorization as a SCAMPI V1.1 lead appraiser. The ADS

is intended to serve as a common basis for the disclosure of SCAMPI appraisal results, and is now the basis for any SEI cataloging of appraisal results that the sponsor agrees to make publicly available.

Myth 3: SCAMPI appraisal team members must be trained in CMMI within 60 days of the appraisal onsite.

SCAMPI appraisal team members must receive Introduction to CMMI model training from an SEI-authorized instructor at any time prior to participation as a team member. (This can be provided by the lead appraiser if—and only if—the individual has also qualified as an instructor.) Appraisal team training is typically provided by the lead appraiser, except in some large organizations that provide organizational team training for their appraisal team members. There are currently no requirements for renewal or refresher training.

Myth 4: SCAMPI cannot be used in discovery mode.

Discovery mode, in which little or none of the needed objective evidence is readily accessible by the appraisal team, is an acceptable mode of use for SCAMPI. However, this mode of use is the most expensive and time consuming.

Further, excessive or repeated employment of this mode of use may suggest an organization that is not managing process improvement in a way that provides sufficient visibility to management. The whole idea of practice implementation indicators is that these constitute a valuable resource for the organization itself, whether it ever conducts a formal appraisal or not.

Myth 5: Having two direct artifacts obviates the need for indirect or affirmation objective evidence.

This myth is a result of misunderstanding the significance of a direct artifact as well as the purpose of the requirement for indirect or affirmation objective evidence. The number of direct artifacts pertinent to a practice depends on the practice as well as its implementation in an organization. One direct artifact might show software planning, for example, while another might represent the systems engineering master plan.

The requirement for indirect or affirmation objective evidence meets the Appraisal Requirements for CMMI (ARC) requirement for corroborating objective evidence. These assure that the direct artifacts were created within an understood process, or are being used and maintained.

Myth 6: Alternative practices must be one-for-one replacements for model practices.

In the CMMI Product Suite the term “alternative practice” is defined as “a practice that is a substitute for one or more generic or specific practices contained in CMMI models that achieves

an equivalent effect toward satisfying the generic or specific goal associated with model practices. Alternative practices are not necessarily one-for-one replacements for the generic or specific practices."

Thus, it may be that an organization is able to achieve a goal with a different number of practices than is expected by the CMMI models.

Myth 7: If we conduct a SCAMPI appraisal using the staged representation, we must report a maturity level.

The SCAMPI method requires that, as a minimum, goal ratings be performed; there is no requirement that any further rating activities be performed. Furthermore, even if additional ratings are rendered by the appraisal team, the appraisal sponsor can decide how to communicate these ratings within the organization.

Myth 8: If we use the staged representation, we must examine all process areas up to and at the target maturity level.

The model scope of a SCAMPI appraisal is governed by the appraisal outputs selected by the appraisal sponsor and the inherent relationships among CMMI process areas. *If a maturity level rating is not selected as an appraisal output*, the model scope could be as small as one process area, irrespective of the representation chosen.

Myth 9: Appraisal team members retain full productivity until 4:00 am.

No one's interests are served by agreeing to unrealistic appraisal plans. The SCAMPI lead appraiser has primary responsibility for ensuring that the appraisal plan is realistic. The SCAMPI documentation does not "require" the appraisal to be finished in a specific amount of time, so it may be necessary to change the plan if excessive hours are being spent on the appraisal.

Myth 10 (1/2): If all of the goals are satisfied during an ARC class B, we should be able to render a maturity level rating.

First, ARC class B appraisals do not produce goal ratings; that is, they do not sanction goal ratings as an output of a class B appraisal. Ratings are reserved for ARC class A appraisals, and the CMMI community has invested significantly in a standard appraisal method for this purpose—the "Class A" SCAMPI.

Second, class B appraisal methods are exempt from some key ARC requirements pertaining to data consolidation and validation, thus providing less rigor than an ARC class A method in the accuracy of the resulting findings. Thus, the CMMI Product Development Team decided that the rendering of ratings is not warranted for class B or C methods.

Finally, the team believed that allowing appraisal methods with significantly varying degrees of rigor to produce ratings would eventually result in an erosion of confidence and support for ARC class B and C methods, thus depriving these classes of appraisal methods of their useful role in process improvement.

Many organizations execute high-fidelity class B appraisals that assure readiness for class A. If they have not been planned and executed as a class A, however, they simply cannot count for the purpose of rendering a maturity level.

Myth 11: There are only 10 SCAMPI myths.

Just a little humor!

Myth 12: Periodic progress reviews are mandatory for maintaining a maturity level rating.

A variation of this myth is that a SCAMPI appraisal must be repeated every two years to be "valid." At this time, there is no SEI position on duration for SCAMPI appraisal outputs.

While there are currently no formal requirements for periodic progress or status reviews (or appraisals), such reviews are a common practice in related methodologies (e.g., ISO 9000); furthermore, if widely adopted, they could provide a rational basis for continuing confidence in the results of a SCAMPI appraisal over time without incurring the cost of repeated SCAMPI appraisals. Thus while there is no requirement for periodic reviews, they do represent a best practice.

Myth 13: Formal structured Federal Acquisition Regulation (FAR) interviews carry more weight in a SCAMPI than informal opportunistic interviews.

It is less disruptive and far less expensive to conduct interviews as needed. This mode is encouraged by the SCAMPI method.

Myth 14: Every SCAMPI lead appraiser has "tenure" in the SEI system.

First, SCAMPI lead appraisers must maintain their skills to be re-licensed. This currently requires them to participate on at least two SCAMPI Class A appraisals and lead at least one during the authorization period. Licenses do expire, and certain actions may result in a license being withdrawn. A listing of currently authorized lead appraisers is available at http://www.sei.cmu.edu/managing/scampi.html?si.

About the Author

Mike Phillips is the Director of Special Projects at the SEI, a position created to lead the Capability Maturity Model® Integration (CMMI®) project for the SEI. He was previously responsible for transition-enabling activities at the SEI.

Prior to his retirement as a colonel from the Air Force, he managed the $36B development program for the B-2 in the B-2 SPO and commanded the 4950th Test Wing at Wright-Patterson AFB, OH. In addition to his bachelor's degree in astronautical engineering from the Air Force Academy, Phillips has masters degrees in nuclear engineering from Georgia Tech, in systems management from the University of Southern California, and in international affairs from Salve Regina College and the Naval War College.

Eye on Integration

ISIS and the Goal of Interoperability

ED MORRIS

For the last several years, the *COTS Spot* column has addressed a number of topics important to the use of commercial off-the-shelf (COTS) products in software-intensive systems. Although the focus of the column was intentionally narrow, we recognized that selecting and incorporating COTS products was only one part of the challenge of building large software-intensive systems.

What was apparent from the outset was that large systems – even those with a COTS product at its foundation – incorporate many other types of software. This other software may include legacy-system code, code developed for organizational purposes (e.g., Federal non-developmental-item, or NDI, code), shareware, custom-built adaptation code that extends core COTS capabilities using vendor-provided hooks and languages, and custom code that provides unique capabilities. Constructing a system that integrates these various types of software into a cohesive solution is a daunting challenge.

Also apparent from the outset was the fact that no large-scale software system stands alone in today's computing environment. Software systems are increasingly expected to use information from and provide information to other systems. For example, satellite ground-support software must interact with mission software; radar systems must share track information; and personnel systems must provide information to financial systems. This assembly of large software-intensive systems into still larger systems of systems has placed intense focus on integration to achieve interoperability[1].

To respond to a growing call to solve the wide range of integration problems evident when diverse systems are incorporated into systems of systems, the SEI has created a new Integration of Software Intensive Systems (ISIS) initiative. To reflect the transition to our new, broader focus, we have renamed the *COTS Spot* column *Eye on Integration*.

Part of the foundation of ISIS was established by the SEI's Systems of Systems Interoperability (SOSI) project in 2003. In this project, we sought to understand the state of the practice for achieving interoperability among systems and to identify areas in which the SEI could help improve the practice.

1. For our purposes (at least for now) *interoperability* is defined as the ability to provide services to and accept services from other systems and to use the services to operate effectively together. *Integration* is the activity performed to achieve interoperability.

The SOSI team held two workshops[1], interviewed several interoperability experts, and surveyed important literature on interoperability. The team identified both successes and failures. In general, interoperability is hard won and expensive. The SOSI team identified a wide range of problems.

- New systems designed and constructed to interoperate with existing and other new systems fail to interoperate as expected. Reasons for the failures vary, but incomplete requirements, unexpected interactions, and unshared assumptions were common.
- Planned interoperability among new systems is sometimes scaled back to maintain compatibility with older systems that cannot be upgraded without major rework.
- Even strict specification of standards (e.g., Link-16) proved insufficient for achieving desired levels of interoperability, because organizations constructing "compliant" systems interpret specifications in different ways, thus creating different, and sometimes incompatible, variants.
- Policies sometimes promote a single point of view at the expense of other points of view. For example, policies that enhance the levels of interoperability that can be achieved in one domain are sometimes generalized to additional domains, where they unduly constrain organizations or enforce unhelpful standards.
- Funding and control structures within the DoD are not yet sufficient for providing the incentives necessary to achieve interoperability. For example, even though there is increased emphasis on the joint nature of many systems, funding for most programs still flows through service sponsors.
- Tests constructed to verify interoperability sometimes fail to identify interoperability shortfalls. In other cases, systems are approved for release in spite of failing interoperability tests.
- Even when interoperability is achieved by systems of systems, it is difficult to maintain as new versions of constituent systems are released. These new system versions sometimes break interoperability.

The SOSI project extracted the majority of its lessons from DoD sources. However, lest anyone think that the interoperability problem is limited to DoD systems, consider this commercial banking-industry scenario:

> After deregulation of the banking industry, a commercial bank built new capabilities in investment management, insurance, and other areas through acquisition of other firms. The previously independent firms became divisions of the commercial bank and were treated as separate cost centers to encourage retention of staff and motivate high achievement. Following the consolidation cycle, the now larger institution recognized that in addition to diversifying into new market areas, it had acquired dozens of information systems that provided inconsistent, but in many

1. Proceedings from the first workshop can be found in the technical note CMU/SEI-2003-TN-016 at http://www.sei.cmu.edu/publications/documents/03.reports/03tn016.html. A technical report detailing the complete SOSI findings is forthcoming.

> cases duplicate, services. These systems ran on diverse platforms, used different databases, and had unique user-interface and communication requirements. To become more competitive, the institution would have to consolidate some systems and develop mechanisms for interoperation between others. Unfortunately, initiatives to identify a common architecture and shared data semantics for interoperability were disrupted due to several factors, including the diverse hardware and software technologies represented, and the differing demands of the managers of the various divisions. Division managers, who were encouraged to minimize cost and disruption to their existing computing infrastructure and systems, fought hard to avoid making the changes to achieve interoperability.

The banking example highlights several important features of the interoperability problem:

1. Interoperability problems increasingly involve many systems, and cannot be effectively solved by developing point-to-point solutions between pairs of systems. This is particularly the case for the next generation of systems of systems such as those being developed for e-commerce and for the DoD's network-centric warfare.

2. Many situations that call for interoperability among systems are unknown at the time that the involved systems are specified. In this case, the value of interoperability among the banking systems was discovered long after the systems were in use. In general, no matter how carefully requirements for interoperable systems are identified, new opportunities for interaction among systems will emerge.

3. Individuals and organizations are motivated by many, often competing incentives that may be conflict with achieving interoperability goals. In the banking industry example, disruption to existing systems and costs associated with changes discouraged the compromises necessary for interoperability. In general, organizations are influenced by many incentives that discourage interoperability.

These and many other problems that occur in achieving interoperability are the result of the inherent complexity of managing and building systems of systems. As Fred Brooks[1] pointed out more than 15 years ago, technical innovations intended to improve software engineering have not successfully solved the problems represented by this inherent complexity. We believe that such is still the case today, and building and maintaining interoperating systems is becoming an even more complex task.

1. In his classic article, "No Silver Bullet: Essence and Accidents of Software Engineering" (*IEEE Computer, 20* (4), 10-19), Brooks suggested that factors such as complexity of systems, lack of conformity between organizations building them, changeability of expectations, and invisibility of details are essential characteristics of software that makes building software difficult.

The SEI is building on the SOSI project. Like SOSI, the SEI will focus on a full range of management, technical, and operational barriers to achieving and maintaining interoperability among complex systems. Keep your eye on *Eye on Integration* to track our progress.

About the Author

Ed Morris is a Senior Member of the Technical Staff at the Software Engineering Institute, assigned to the Integration of Software-Intensive Systems (ISIS) Initiative. He is currently investigating approaches to achieving technical interoperability between complex systems and programmatic interoperability between the organizations that build and maintain them. Previous activities involved improving processes and techniques for the evaluation and selection of COTS products, and the development of the COTS Usage Risk Evaluation (CURE) technology. Before coming to the SEI, Ed developed custom operating systems for embedded microprocessors along with support tools to predict and monitor the performance of real time systems.

Security Matters

What is a Distributed Denial of Service (DDoS) Attack and What Can I Do About It?

LARRY. ROGERS

Have you ever tried to make a telephone call but couldn't because all the telephone circuits were busy? This may happen on a major holiday and often happens on Mother's Day. In fact, in the United States telephone companies used to air commercials on television and radio that suggested you avoid peak calling times by making your calls early or late in the day.

The reason you couldn't get through is because the telephone system is designed to handle a limited number of calls at a time. That limit was determined by weighing the cost of having all calls get through all the time with the amount of traffic the system receives. If the total number of calls is always high, it makes economic sense for the telephone company to provide more capacity to match that demand. However, if the number of calls is low compared to the holiday peaks, then the telephone company will build networks that accommodate only the lower off-peak number of callers and advise its customers to avoid peak calling times. It's a basic matter of supply and demand.

Imagine that an intruder wanted to attack the telephone system and make the system unusable by telephone customers. How would he or she do this? One way would be to make call after call in an attempt to make all circuits busy. This type of attack is called a ***denial of service***, or ***DoS***, attack. In essence, the intruder has caused the telephone system to deny service to its customers. It is not likely that one caller working alone can tie up all telephone circuits. To do that would require making as many calls as possible from as many telephones as possible. This is called a ***distributed denial of service***, or ***DDoS***, attack.

Computer systems can also suffer DoS and DDoS attacks. For example, sending an extraordinary amount of electronic mail to someone could fill the computer disk where mail resides. This means that people who use the computer with the full disk cannot receive any new email until the situation changes. While this is an older style of DoS attack, it is still popular today.

In addition, intruders have turned their efforts toward denying people the services provided by networked computers. Examples of frequently attacked services are the World Wide Web[1], file-sharing services and, more recently, the Domain Name Service.[2] Because so many of our computers are connected through the Internet, attacking one of these services can have a significant effect on the whole Internet community. For example, by launching a DoS attack on a

1. http://www.computerworld.com/networkingtopics/networking/story/0,10801,60501,00.html

2. http://www.computerworld.com/securitytopics/security/story/0,10801,75454,00.html

popular merchant during a high sales period, the intruder affects not only that merchant's company, but everyone who is then unable to buy its products.

To deny these services to prospective users of a computer service, intruders run specially written computer programs that send extraordinary volumes of Internet "calls" to one of the computers that provides that service, similar to the way that an intruder can tie up the telephone system.

When a computer answers such a call, most often there's no one on the other end, so answering the call turned out to be a waste of time. Unfortunately, the attacked service cannot tell this in advance, so it has to answer all calls placed to it. Answering each call takes time, and there's only so much time available. Again, supply and demand.

In addition, the volume of traffic may be so high that the networks connecting the attacking computers to the victim's computer may also suffer from degraded performance. Just like the telephone system and service computers, these networks cannot handle traffic beyond a certain limit. Users wanting services from computers on those networks are denied those services, too. Those networks are also considered victims of a DDoS attack.

How do intruders wage a DDoS attack against a victim's computer?

First, they build a network of computers that will be used to produce the volume of traffic needed to deny services to computer users. We'll call this an ***attack network***.

To build this attack network, intruders look for computers that are poorly secured, such as those that have not been properly patched, or those with out-of-date or non-existent anti-virus software. When the intruders find such computers, they install new programs on the computers that they can remotely control to carry out the attack.

Intruders used to select the computers that made up the attack network individually. These days, however, the process of building an attack network has been automated through self-propagating programs. These programs automatically find vulnerable computers, attack them, and then install the necessary programs. The process begins again as those newly compromised computers look for still other vulnerable computers. Once a DDoS program has been installed on a computer, that program identifies the computer as a member of the attack network. Because of this self propagation, large attack networks can be built very quickly. A by product of the network-building phase is yet another DDoS attack, because searching for other vulnerable computers creates significant traffic as well.

Once an attack network is built, the intruder is ready to attack the chosen victim or victims. Some information-security experts believe that many attack networks currently exist and are dormant, passively waiting for the command to launch an attack against a victim's computers. Others believe that once a victim has been identified, the attack network is built and the attack launched soon afterward.

To reduce their chances of being discovered, intruders distribute their attack across computers in different time zones, different legal jurisdictions, and with different systems administrators. Intruders also make the electronic traffic they create appear to be from a computer different from the one that actually created it. This is called ***IP spoofing***, and it is a commonly used method to disguise where an attack is really coming from. If the source of the attack is unknown, it is difficult to stop it, giving intruders free reign with a high likelihood of successfully remaining anonymous.

The MyDoom virus is an example of building such a DDoS attack network. In this case, the attack network was built not through technological vulnerabilities but rather through operational vulnerabilities. Computer system users were coaxed into executing a malicious program that was either sent as an email attachment or as a file downloaded through a point-to-point network connection, effectively enrolling their computer systems into the attack network. However, instead of remotely controlling the newly installed malicious program as previously described, the intruder designed it to automatically send significant amounts of traffic to www.sco.com[1] on February 1, 2004 and www.microsoft.com on February 3, 2004. See Technical Cyber Security Alert TA04-0[2]28A for a detailed explanation of MyDoom. This alert also lists steps that can be taken to remove it from an infected computer system.

What can be done about DDoS attacks?

There are no short-term solutions to eliminate DDoS attacks. Today's best practices involve making computers and networks more resilient in the face of an attack. We call this ***survivability.***

All systems have their limits. One way to make a system more survivable is to increase these limits; the more resources there are, the better the chances are that the system will survive an increased demand for use. To increase the telephone system's limits, the telephone company adds more circuits. For a web service, the webmaster might increase the number of connections that a web service can accept; for example, a site could add more web servers. This spreads the increased load over more computers and helps to ensure that no one computer operates too near its limit. The higher the limits of all the potentially affected systems—the network and the computers on that network – the better the chances that network will survive a DDoS attack.

You can do your part to ensure that your computers are never part of a DDoS attack network by following security best practices, such as those in *Home Computer Security*[3]. Then, be alert to changes in your computer or network performance.

1. www.sco.com was changed to www.thescogroup.com in an attempt to defeat the MyDoom virus and is no longer a valid host name.

2. http://www.us-cert.gov/cas/techalerts/TA04-028A.html

3. http://www.cert.org/homeusers/HomeComputerSecurity

Some signs that your computer is part of a DDoS attack

1. Are your computers running noticeably slower than usual?
2. Is your Internet connection slower than usual?
3. Are the activity lights on your high-speed (cable or DSL) modem solid, or on almost all of the time?

Any of these signs could indicate that your computer system may be a participant in a DDoS attack network. If this happens to you, contact your Internet service provider (ISP) and follow its recommendations. Also, you should consider turning off your computer system or your high-speed modem. That will certainly stop the flow of DDoS traffic, though this is only a temporary solution.

If your computer system was a participant in a DDoS attack network, your system was compromised, and attack tools were installed on your computer, you'll need to determine what the intruders did and then repair the damage. The article *There IS an Intruder in My Computer—What Now?*[1] describes how to recover from an intrusion on your home computer.

Distributed denial of service attacks are a significant problem. These attacks will be with us for a while, though there is ongoing research on how to reduce them (see the section below). Until then, DDoS is no (tele)phoney baloney!

Learn more about DDoS attacks

You can find more reading for home users on the CERT Web site (http://www.cert.org/homeusers). If you would like more technical details, you might be interested in reading the following:

- Trends in Denial of Service Attack Technology (http://www.cert.org/archive/pdf/DoS_trends.pdf)
- Papers on the topic of survivability (http://www.cert.org/nav/index_purple.html)
- Tracking and Tracing Cyber-Attacks: Technical Challenges and Global Policy Issues See Part II on future directions to address the DDoS problem. (http://www.cert.org/archive/pdf/02sr009.pdf)

About the Author

Lawrence R. Rogers is a senior member of the technical staff in the Networked Systems Survivability Program at the Software Engineering Institute (SEI). The CERT Coordination Center is a part of this program. Rogers's primary focus is analyzing system and network vulnerabilities

1. http://www.cert.org/homeusers/intruder2.html

and helping to transition security technology into production use. His professional interests are in the areas of the administering systems in a secure fashion and software tools and techniques for creating new systems being deployed on the Internet. Rogers also works as a trainer of system administrators, authoring and delivering courseware. Before joining the SEI, Rogers worked for 10 years at Princeton University. Rogers co-authored the Advanced Programmer's Guide to UNIX Systems V with Rebecca Thomas and Jean Yates. He received a BS in systems analysis from Miami University in 1976 and an MA in computer engineering in 1978 from Case Western Reserve University.

Software Product Lines

"Lunching" and Institutionalizing

PAUL CLEMENTS

In 1999, a small group of us visited a company that manufactures software-intensive aircraft subsystems. We were interested in the software architecture(s) that they used for their systems, and how architecture played a role in their overall culture and development process. We interviewed over 20 people in six sessions over two days, including developers, senior designers, functional area managers, quality assurance/configuration management/tool groups, and systems designers. Although architecture was our quarry for this trip, we were delighted to discover a strong product line culture along the way.

The first interview session was with a group of senior designers with the title of "project engineer." Project engineers are the technical authorities for projects. They drew for us a detailed picture of the common architecture for their group of products, which we will give the collective (and, to protect confidentiality, fictitious) name of "Aircraft Systems," and which we were to learn was indeed a product line of systems at this company. The project engineers told us that projects[1] begin with attempts to reuse, with as little change as possible, the architectural components from previous projects. In fact, the process starts with the software requirements specification (SRS). The Aircraft Systems group maintains a "common" (what we would call a "core") SRS; system-specific SRSs are written as variations of this. They also told us that the project engineers meet weekly to work out common requirements that they can pass on to the groups who maintain the major subsystems and associated software.

We asked what prevented a developer from picking up a component from a previous project and changing it to meet the exigencies of the product he or she was working on at the time. The answer, they said, was that this was strongly discouraged. The functional area managers would not allow it, and the offending engineer would "have to eat by himself in the lunchroom." Beneath the levity of this comment, a picture of strong product line culture was beginning to emerge.

Subsequent interview groups confirmed this. We learned that their development environment (in this case, Rational's APEX) plays a critical role in the sustainment of the product line. For one thing, every project is open to every other project, so that people can straightforwardly search for components to reuse. Each component is stored with its history of usage, its own requirements, its test cases, and its Ada spec, so when retrieved for reuse its entire heritage and artifact accompaniment come with it. APEX reveals what components depend on what other components, and users are able to compare versions to quickly discern differences. And APEX automatically sends email to the functional area manager when a changed subsystem under that manager's

1. By "project" we mean here the organizational unit or team that builds a product.

purview is checked in. We were learning that the functional area managers are the guardians of the product line.

At this point, we did not know who these mystical functional area managers were, or what role they played. But they were clearly important to the product line, and commanded a lot of respect from the engineers and developers. We asked to interview them—they were not originally on our schedule because we did not know they existed—and they revealed the whole product line picture.

About four and a half years ago, this company faced a crisis: three major projects, which were expected to be won in a staggered fashion, came in simultaneously. These projects were to supply systems for three different kinds of United States military aircraft. They realized that they needed to exploit the commonality in all of the systems, and the product line was born. There was apparently no leader from above who ordered the architects into a locked closet and forcefully heeled the organization over to a new tack. Rather, the functional area managers simply began to meet (over lunch, we are told) to work out procedures for exploiting the commonality that they knew existed in previous projects and that would be required in the new ones. Projects apportioned the work and shared each other's components.

Today, the functional area managers control the design and evolution of the subsystems. They are like a core asset group, but unlike other core asset groups, this one has no implementation duties. Rather, the functional area managers are like senior designers who (a) approve or reject design changes requested by individual projects; (b) keep track of changes that are needed by more than one project; and (c) find projects that have the money to make the approved changes. (Sometimes changes are made using independent research and development money as well.) So this "core asset group" has the vision and the change authority, but changes are made by projects. Each of these managers has a team of people that "work" for them and are assigned to individual projects as needed. Each functional area team represents a collection of expertise about a particular functional area (subsystem). Thus, the organization is a matrix in which a developer works for both a functional area manager and (while assigned to a project) a project manager. See Figure 1.

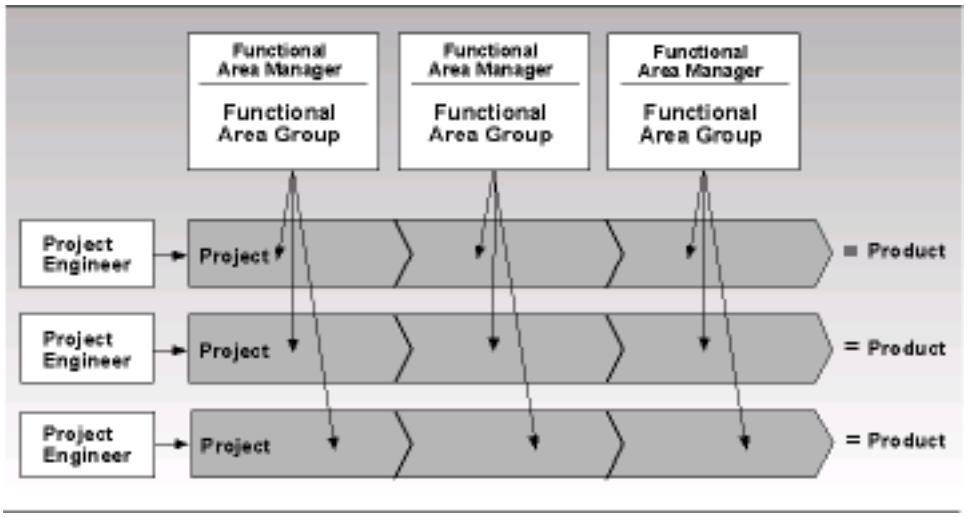

Figure 1: Organization by Functional Area Group

Improvements are continually made to the subsystems (which are their primary software core assets). The functional area managers are always looking for ways to improve the architecture, and then looking for projects that can use (and hence pay for the making of) those improvements. Examples include (a) moving away from direct subroutine invocation as the collaboration mechanism, and toward registration; and (b) simplifying the pattern of dependencies (as manifested by Ada "with" statements) among the subsystems, which at the time of our visit was almost incomprehensibly complex.

Some of the conditions that work in this company's favor include

- a relatively small number of projects so far. The number of versions of any subsystem that a reuser has to browse through is no more than 15-20, and 6 is a much more likely number. Thus, they don't yet have a runaway version-control or component-search problem.
- a domain in which the software architecture naturally mirrors the hardware/system architecture. Most software that they build matches up straightforwardly to electronic boxes that plug into the aircraft, boxes that the company has long manufactured. Other software that doesn't correspond precisely to a box falls out naturally and is treated separately.
- deep domain expertise, which is not a surprise.

Also not surprisingly, their architecture supports the product line view by providing changeability and portability qualities. It is strongly layered, and enforces strict visibility rules among the layers. Subsystems that are present in every product (such as a bus manager that handles putting messages on the communication bus and pulling them off for other subsystems) are distinguished from subsystems that may or may not be present in a particular product. Subsystems have internal structures of their own; the part that displays information is often changed, but the state-machine or controlling part seldom does, and these are well-separated parts of each subsystem. There is an

input/output part of each subsystem that communicates with other subsystems via the bus. Overall, the architecture features minimal use of global data, maintains stable subsystem interfaces, and mandates consistent and disciplined interprocess communication protocols. Platform and environmental dependencies, scheduling policies, user interface, and message formats are all isolated (encapsulated) in separate software components.

This company clearly has a strong culture of product lines and architecture, and this emerged consistently. Everyone who drew the architecture drew the same picture. Everyone who told how the product line works told the same story. But the culture is an oral one. Nowhere was the product line operation codified in writing. Except for the SRSs, people rather than documents seemed to be the authoritative sources for much of the information necessary in development. But the oral culture has advantages. There is a strong and formal mentoring process in place, and it seems to be quite effective. There is also open communication. Developers are free to talk to their project engineers and advocate changes; they either convince them to make the change or are convinced by them that the change is not needed. The project engineers will not hesitate to lobby the functional area managers for changes. Communication flows freely.

And what does this company have to show for its product line efforts?

- They can port to a new processor in about eight days, and have already ported their software to a PowerPC.
- Reuse numbers as high as 94% were cited, and reuse in the 80% range seemed merely nominal for the groups to whom we spoke, earning only a modest shrug of the shoulders.
- Their tool/environment person (whose tools help collect metrics) estimated that what they used to do in 18-38 months they can now do in 8-16 months.

So necessity was once again the mother of invention, a product line scenario we have seen more than once. But under the calm guidance of the functional area managers, who saw what needed to be done and worked quickly and efficiently among themselves to do it, the transition to the product line paradigm was accomplished with a minimum of trauma. But besides starting the product line, the functional area managers have been its guardians, have kept it alive and nurtured it, have protected its conceptual integrity, and have instilled a strong sense of product line culture in the entire Aircraft Systems organization. They are its champions.

Not bad for an idea that started over lunch.

About the Author

Dr. Paul Clements is a senior member of the technical staff at Carnegie Mellon University's Software Engineering Institute, where he has worked for 8 years leading or co-leading projects in software product line engineering and software architecture documentation and analysis.

Clements is the co-author of three practitioner-oriented books about software architecture: *Software Architecture in Practice* (1998, second edition due in late 2002), *Evaluating Software Architectures: Methods and Case Studies* (2001), and *Documenting Software Architectures: View and Beyond* (2002). He also co-wrote *Software Product Lines: Practices and Patterns* (2001), and was co-author and editor of *Constructing Superior Software* (1999).In addition, Clements has also authored dozens of papers in software engineering reflecting his long-standing interest in the design and specification of challenging software systems.

He received a B.S. in mathematical sciences in 1977 and an M.S. in computer science in 1980, both from the University of North Carolina at Chapel Hill. He received a Ph.D. in computer sciences from the University of Texas at Austin in 1994.

What's New

Defective Software Works

WATTS S. HUMPHREY

Over the years, many people have written to me with questions about software quality, testing, and process improvement. Jon Hirota asked how to get organizations to invest in software quality; John Fox asked if I see a movement away from system test and toward quality processes; Bob Schaefer wondered what I thought would happen in the area of software integration and testing; Dan St. Andre asked what software development managers can do to encourage executive management to meaningfully address software quality; and Pete Lakey wondered if and how the software community should use statistical process control techniques. I won't directly answer all of these questions but I will discuss these quality and testing issues here and in my next column. While it has taken me an embarrassingly long time to respond to these letters, they still raise a critical question: "How important is software quality and how should quality practices change in the future?"

First, What Do We Mean by Quality?

While the classical definition of product quality must focus on the customer's needs, in this and the next column, I will concentrate on only the defect aspect of quality. This is because the cost and time spent in removing software defects currently consumes such a large proportion of our efforts that it overwhelms everything else, often even reducing our ability to meet functional needs. To make meaningful improvements in security, usability, maintainability, productivity, predictability, quality, and almost any other "ility," we must reduce the defect problem to manageable proportions. Only then can we devote sufficient resources to other aspects of quality.

The functional and operational characteristics of a software product should and will continue to be important, but that is where most people now focus, and there is little risk that they won't continue to do so in the future. If a product doesn't have attractive functional content, it won't sell, regardless of how few or how many defects it contains. Unfortunately, many software groups treat the importance of functional quality as an excuse to concentrate almost exclusively on product function and devote little attention to better managing the defect problem.

While there is irrefutable evidence that the current "fix-it-later" approach to defect management is costly, time consuming, and ineffective, don't expect this to change soon. It is too deeply ingrained in our culture to be rooted out easily. However, since I'm an optimist, I'll keep trying to change the way the world views software quality. And by that, I mean the way we manage defects.

Second, How Important is Software Quality?

The key question is: "Important to whom?" Developers are necessarily preoccupied with defects. They spend the bulk of their time trying to get their products to work. And then, even when the

products do work, the developers spend even more time fixing test and user-reported problems. While few developers recognize that their schedule and cost problems are caused by poor quality practices, an increasing number do. This is a hopeful sign for the future.

Not surprisingly, development management tends to view quality as important only when executives do. And the executives view quality as important only when their customers do. When customers demand quality, executives almost always pay attention. While their actions may not always be effective from a quality-management perspective, they will almost always respond to customer pressure. And even if their customers do not demand improved quality, government regulation can cause both executives and development managers to pay more attention to quality. This is true for commercial aircraft, nuclear power plants, and medical devices. There is little question that, with commercial aircraft for example, the close and continuous regulatory scrutiny coupled with painstaking reviews of every safety incident hold this industry to high quality standards.

Third, Why Don't Customers Care About Quality?

The simple answer is: "Because defective software works." The reason it works, however, is because software doesn't wear out, rot, or otherwise deteriorate. Once it is fixed, it will continue to work as long as it is used in precisely the same way. But, as software systems support an increasing percentage of the national infrastructure, they will be used in progressively less predictable ways. When coupled with the explosive growth of the Internet and the resulting exposure to hackers, criminals, and terrorists, the need for reliable, dependable, and secure software systems will steadily increase. If experience is any guide, as these systems are used to perform more critical functions, they will get more complex and less reliable. Unfortunately, this probably means that it will take a severe, disruptive, and highly public software failure to get people concerned about software quality.

Two forces could change this complacent attitude. One is the Sarbanes-Oxley Act, which makes chief executives and chief financial officers personally responsible for the quality of their organizations' financial reports. This has caused executives to inquire into the accuracy of their financial reporting systems. What they find is often disquieting. The general accuracy of such systems is usually reasonably good, but there are many sources of error. While these errors have not been a serious concern in the past, they will become much more important when the senior executives are personally liable.

The second issue is closely linked to the first. That concerns software security. Although this is not yet well recognized, when software systems are defective, they cannot be secure. Executives are also just beginning to realize that software security is important to them because, if their systems are not secure, they almost certainly cannot be accurate or reliable. Since they are now personally liable for the accuracy of their financial systems, they now are beginning to appreciate the need for secure financial systems.

How Defective is Software?

So the basic question concerns defects. First, some facts. The number of defects in delivered software products is typically measured in defects per thousand lines of code (KLOC). Figure 1 shows some data on recent history. Here, Capers Jones has substantial data on delivered product defect levels, and he has compared these with the CMM[1] maturity of the organizations that developed the software [1]. Noopur Davis has converted the Jones data to defects per million lines of code (MLOC), as shown in Figure 1 [2]. For example, organizations at CMM level 1 delivered systems with an average of 7,500 defects per MLOC while those at level 5 averaged 1,050 defects per MLOC. What is disquieting about these numbers is that today most products of any sophistication have many thousands and often millions of lines of code. So, while one defect per KLOC may seem like a pretty good quality level, a one million LOC system of this quality would have 1,000 defects. And these are just the functional defects.

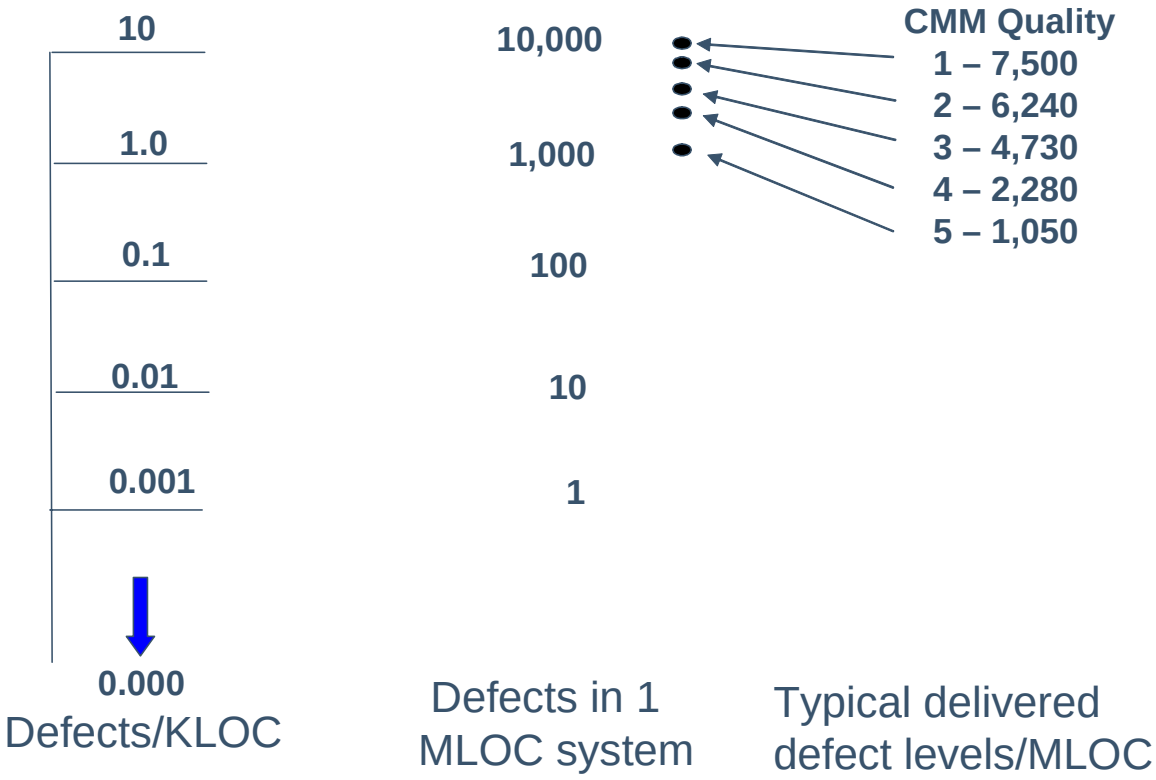

Figure 1: Typical Software Quality Levels – in Delivered Defects

Today, most programmers are unaware of many types of security defects. While some security defects are generally recognized as functional defects, others are more subtle. A security defect is any design error that permits hackers, criminals, or terrorists to obtain unauthorized access or use of a software system. Since many of these defects do not cause functional problems, systems that are riddled with security flaws may work just fine and even pass all of their functional tests. And,

1. CMM is registered with the U.S. Patent and Trademark Office by Carnegie Mellon University.

since many programmers are worried only about getting their programs to function, they are almost totally unaware of their products' potential security vulnerabilities.

In one program that had been supporting a Web site for over two years, a security audit found one functional defect and 16 security defects. So, today's one MLOC systems with 1,000 functional defects could easily have many thousands of security defects, and almost any of these could provide a portal for a hacker, thief, or terrorist to steal money, disrupt records, or to otherwise compromise the accuracy of the organization's financial system. No wonder executives are worried about the Sarbanes-Oxley Act.

Putting Software Quality into Perspective

At present, over 90% of all security vulnerabilities are garden-variety functional defects. This means that our initial goal must be to reduce functional defect levels. From a security perspective, one defect per KLOC is totally inadequate for large systems. But what does one defect per KLOC mean in human terms? One thousand lines of source code, when printed in a listing, typically take about 30 to 40 printed pages. This means that one defect in 30 to 40 printed pages is poor quality. No other human-produced product comes close to this quality level.

But even though one defect per KLOC is far beyond the levels that humans ordinarily achieve, this quality level is totally inadequate. At this quality level, most large systems will contain many thousands of defects, and any one of them could open the door to security problems. So, if 1,000 defects in a one MLOC (million line-of-code) system is inadequate, what would be adequate? That, of course, is impossible to say, but 10 defects per MLOC would be an advance from where we are now. However, I am afraid that even 10 defects per MLOC will not be adequate forever.

Reaching a level of 10 defects per MLOC would mean only one defect in 3,000 to 4,000 pages of listings. That goal seems preposterous. Are such quality levels needed, and is there any hope that we could achieve them? Unfortunately, we almost certainly need such levels today. Criminals, hackers, and terrorists now have and will continue to devise more sophisticated and automated ways to probe our systems for security vulnerabilities. And any single vulnerability could open the door to serious problems.

The problem is not that 10 defects per MLOC is difficult to achieve. The real problem is that, even if it is an adequate security level today, it will almost certainly not be adequate for very long. In the next column I will discuss this problem and explore some of the issues we face in producing systems that have such extraordinary quality levels.

Acknowledgements

In writing papers and columns, I make a practice of asking associates to review early drafts. For this column, I particularly appreciate the helpful comments and suggestions of Julia Mullaney, Bill Peterson, Marsha Pomeroy-Huff, and Carol Woody. The letters from

John Fox, John Hirota, Pete Lakey, Bob Schaefer, and Dan St. Andre were also very helpful and I thank them as well.

In closing, an invitation to readers

In these columns, I discuss software issues and the impact of quality and process on developers and their organizations. However, I am most interested in addressing the issues that you feel are important. So, please drop me a note with your comments, questions, or suggestions. I will read your notes and consider them when planning future columns.

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

References

[1] Jones, C. Software Assessments, Benchmarks, and Best Practices. Reading, MA: Addison Wesley, 2000.

[2] Davis, N. & Mullaney, J. *The Team Software Process (TSP) in Practice: A Summary of Recent Results* (CMU/SEI-2003-TR-014). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003 < http://www.sei.cmu.edu/publications/documents/03.reports/03tr014.html?si>.

About the Author

Watts S. Humphrey founded the Software Process Program at the SEI. He is a fellow of the institute and is a research scientist on its staff. From 1959 to 1986, he was associated with IBM Corporation, where he was director of programming quality and process. His publications include many technical papers and several books. His most recent books are *Introduction to the Team Software Process* (2000) and *Winning With Software: An Executive Strategy* (2002). He holds five U.S. patents. He is a member of the Association for Computing Machinery, a fellow of the Institute for Electrical and Electronics Engineers, and a past member of the Malcolm Baldrige National Quality Award Board of Examiners. He holds a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, and an MBA from the University of Chicago.

Features

3rd International Conference on COTS-Based Software Systems

BILL ANDERSON

ICCBSS: A GLOBAL PRESENCE
ICCBSS DRAWS INTERNATIONAL PARTICIPATION WITH CONFERENCES HELD IN THE US, CANADA, AND EUROPE. ICCBSS 2004 RECEIVED PAPER SUBMISSIONS FROM 17 COUNTRIES AROUND THE WORLD.

The 2004 International Conference on COTS-Based Software Systems (ICCBSS) continued to make significant contributions to the growing field of COTS (commercial off-the-shelf)-based system practice and research. A maturation of the state of the practice was seen in a shift of conference emphasis from selecting a COTS product to engineering and managing the development and sustainment of complex COTS-based systems. Issues as diverse as requirements definition, legal concerns, architectures, safety-critical applications, predictive cost models, interoperability, and security broadened the scope of this year's conference from a focus on COTS products to considerations of system-wide engineering. Rick Hefner, director of process initiatives at Northrop Grumman Mission Systems, said of the conference, "In the past decade, the software community has moved beyond the simplistic view of COTS as a silver-bullet solution to a deeper examination of the critical success factors surrounding COTS usage. More than any other single conference, ICCBSS provides the opportunity to interact with the researchers and practitioners who are defining the state of the practice."

Dr. Rose Gamble of the University of Tulsa was a paper jury panelist, and she commented, "The quality of the presentations [and] the papers I reviewed was superior to previous years." Dr. Gamble's own research on migrating application integrations was a prime example. She reported on her studies of the detailed system changes required to migrate a system from a publish/subscribe COTS middleware architecture to a request/reply architecture, identifying some of the artifacts required to accomplish such a migration successfully. Such sustainment strategies enable a system to adapt to changing technologies and take advantage of COTS market improvements.

Two well-attended workshops produced lively interaction on tools and techniques for incorporating COTS products into software systems and building a framework for COTS

terminology. Participants commented on the high level of interactive discussion and the continuity from conference to conference as issues raised in previous years were explored in detail during this year's workshops. In his closing remarks, Dr. Barry Boehm, of the University of Southern California and this year's program chair, pointed out that ICCBSS is unique in its extended presentation format, allowing this audience participation to continue throughout the conference.

A number of presentations dealt with COTS legal issues. Ignacio Delgado González from Martin & Lawson, Spain, spoke about introducing COTS products into the European Union. The European Software Institute's David Morera's detailed checklist of legal issues when acquiring COTS products, and this author's comments on the challenges of working with shrink-wrapped licenses, illustrate that the legalities associated with COTS-based system acquisition and sustainment are vital to the long-term success of these systems. Turnkey vendors may rely on third-party embedded products without adequate provisions for maintenance of those packages. Warranties, remedies, infringement, and title types are just a few of the areas that need to be well understood when using COTS software.

Use of COTS products in safety-critical systems presents new and significant challenges to certification efforts, integration, and cost estimation. Dr. Tim Kelly of the University of York, UK, spoke elegantly on the topic. He presented an approach to alleviate these difficulties by providing a safety-informed decision on COTS selection. Kelly stated that the "biggest challenges facing spacecraft ground systems developers [are those being faced every day while] architecting mission-critical systems using COTS components." A panel discussion with expert participation from the Boeing Company, the U.S. DoD Joint Program Office, Integral Systems, Aerospace Corp., and Braxton Technologies provided another venue to explore the use of COTS products in critical environments.

Dr. Douglas C. Schmidt, Vanderbilt University, presented an informative keynote on advances in COTS middleware for distributed real-time and embedded systems, providing examples of military and commercial applications that ranged from shipboard resource management to automated stock trading. He concluded his presentation by describing the recent successes of the Bold Stroke Avionics Mission Computing and a real-time retargeting system's infusion of standards-based COTS products, both of which leveraged advances in the quality of service-enabled middleware, distributed resource modeling, and pattern languages.

In the words of Joseph Farrelly, CIO of Aventis Pharma, "Being competent at developing software was once a priority—now you buy that off the shelf, maybe 85% of the time. It's not about developing the best technology anymore. It's about mastering project management in the implementation of technology."[1] The breadth and depth of the material presented at ICCBSS 2004 made a considerable contribution to filling this need.

1. Carr, David F. "Integration Successes and Flops: The Baseline Magazine Perspective," ICCBSS 2004. http://*www.iccbss.org/2004/presentations/carr.pdf*

You can participate in this exchange by attending upcoming conferences and by submitting papers, presentations, tutorials, panels, or posters; ICCBSS 2005 will be held in Bilbao, Spain, February 7-11. Further information, including the entire program and information about how to order the proceedings, is available at the conference Web site at www.iccbss.org.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

World Wide Web
www.iccbss.org

The SEI Partner Network: Providing Authentic SEI Services and Training

JANET REX

A core element of the SEI mission is to broadly disseminate software engineering knowledge and methods to improve the state of the practice of software engineering worldwide. To provide leadership in the transition of new software engineering technology into practice, the SEI licenses specific SEI products and services—typically training courses or assessment services—to organizations that have historically been called "transition partners." To strengthen its relationship with this important constituency, the SEI is working to better inform the software engineering community about these partners. This process is now bearing its first fruit. Transition partners have a new name—the SEI Partner Network—a new, formal code of conduct, and support from a new directory and guide to services that became available in the spring of 2004.

Why the name change? "The SEI has changed the name to emphasize the role of the SEI Partners as providers of official SEI-brand services," says David White, director of technology transition practices. "The SEI's intent is to help users learn where and how they can get authentic SEI services to support their software engineering efforts."

The SEI Partner Network is a group of organizations and individuals that are selected, trained, and licensed by the SEI to deliver authentic SEI services. These services include courses, consulting methods, and management processes that aid in the implementation of the SEI's software engineering technologies. "Organizations that obtain these services from SEI Partners can be sure that the services have the same high quality as those delivered directly by the SEI," says White. "The SEI stands behind the SEI Partner Network and is accountable for the skills and training of partner-sponsored individuals."

There are two key characteristics that distinguish authentic SEI services: permission and qualification. *Permission* means that the person or organization delivering the service is allowed to use the required SEI intellectual property. *Qualification* means that the person delivering the service has the skills and knowledge necessary to do so. The SEI uses the terms "SEI-authorized" and "SEI-certified" to indicate that an individual has the necessary qualifications to perform an SEI service or deliver an SEI course. The rules of permission and qualification vary for each of the SEI services available through the Partner Network.

Benefits of Working with an SEI Partner

- Prerequisites: Authentic SEI courses are required as prerequisites for advanced training programs at the SEI and are sometimes required as prerequisites for performing SEI consulting or evaluation methods.

- Quality: SEI Partners undergo evaluation by the SEI to become partners and are continually reviewed as long as they remain partners. By working with SEI Partners, organizations can have confidence in the quality of the services they are receiving.
- Legitimacy: SEI services can dramatically improve the performance of a software engineering organization or team. Working with SEI Partners will legitimize an organization's improvement efforts and ensure the best possible results.

Quality Assurance and the Code of Conduct

The process for becoming an SEI Partner varies by SEI service, but always includes quality assurance. Potential partners complete an application procedure and review by SEI staff of the applicant organization's capabilities and experience. In addition, the SEI evaluates the credentials and experience of the employees that a partner applicant plans to sponsor for authorization or certification. (More information on how to become a partner can be found at the SEI Partner Network Web site: *www.sei.cmu.edu/partners/?si*.)

"The SEI performs numerous quality checks and implements quality programs that monitor the services delivered by its partners," and, White adds, "The SEI is in the process of integrating the various quality programs to provide even more quality assurance to customers of SEI Partners."

A cornerstone of the quality-integration effort is a newly developed code of conduct. The code of conduct describes the minimum standards of conduct for partner organizations and individuals delivering SEI services. At the time of publication, the code of conduct was being completed. For current information on the code of conduct and to get a copy of the code, please visit *www.sei.cmu.edu/partners/?si*.

Services Available from SEI Partners

Products and services currently available through the SEI Partner Network include

Process Improvement

- The SEI Introduction to Capability Maturity Model® Integration (CMMI®) course
- Standard CMMI Appraisal Method for Process Improvement (SCAMPI℠) Appraisal Services
- The Personal Software Process℠ (PSP℠) and Team Software Process℠ (TSP℠) curriculum
- TSP coaching services

Security and Survivability

- Operationally Critical Threat, Asset, and Vulnerability Evaluation℠ (OCTAVE®) training and evaluation services
- The CERT® course suite on information security

Software Architecture

- The SEI Software Architecture: Principles and Practices course (available from partners in late 2004 or early 2005)

A new SEI Partner Directory and Guide to Services has just been published. This directory provides details on all of the services and training available through the SEI Partner Network. The directory also provides guidance on adopting particular SEI technologies, tips for selecting an SEI Partner, and instructions for purchasing authentic SEI services. To get a copy of the directory, please contact SEI Customer Relations at the phone number or email address below.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

World Wide Web
www.sei.cmu.edu/partners/?si

Using Six Sigma in Software Development

LAUREN HEINZ

Each year at Lockheed Martin, corporate management challenges its Integrated Systems & Solutions business unit (IS&S) to reduce total costs. Each year, IS&S uses Six Sigma tools to make it happen.

"Six Sigma has resulted in significant cost savings," said Lynn Penn, director of quality systems and process management at IS&S. "It's a structured approach that provides more than a checklist—it shows you what's coming next, lets you look at data from different views, and gives you a big picture of your practices for making decisions."

Lockheed Martin is part of a growing number of organizations using Six Sigma to improve software quality and cycle time, reduce defects in products and services, and increase customer satisfaction. As Six Sigma evolves from an improvement framework for the manufacturing sector to one that can be applied across all levels of an enterprise, the SEI is looking at ways that Six Sigma has benefited software and systems development.

More Than a Metric

Six Sigma is an approach to business improvement that includes a philosophy, a set of metrics, and an improvement framework (also called a toolkit). Its philosophy is to improve customer satisfaction by eliminating and preventing defects, resulting in increased profitability. Sigma (S) is the Greek symbol used to represent standard deviation, or the amount of variation in a process. Six Sigma (6S) refers to a measure of process variation (six standard deviations) that translates into an error or defect rate of 3.4 parts per million, or 99.9997 percent. In Six Sigma, defects are defined as any product, service, or process variation that prevents the needs of the customer from being met.

During the 1980s, large manufacturing companies such as Motorola, General Electric, and Allied Signal first used Six Sigma processes to collect data, improve quality, lower costs, and virtually eliminate defects in fielded products. Using both statistical and non-statistical methods, the approach soon spread to several major service industries, and today software practitioners are exploring ways to apply Six Sigma techniques to improve software and systems development.

"Six Sigma is more than just a metric," says Jeannine Siviy, a leading Six Sigma practitioner at the SEI. "Maintaining vigilance about the philosophy, customer satisfaction, and business profitability is crucial to Six Sigma success. Using this philosophy, organizations can define sigma measures and thresholds in customer terms and then link these to engineering measures such as defect density, cost, and schedule performance."

Northrop Grumman Mission Systems, for example, used Six Sigma to help in its move from Maturity Level 3 of the SEI CMMI models to Level 5 in just one year. Rick Hefner, director of process initiatives, said all of their engineers and managers attend a two-week training course in Six Sigma and complete a six-month Six Sigma project. This work is augmented by additional training on the Quantitative Project Management and Causal Analysis and Resolution process areas of CMMI Maturity Levels 4 and 5.

"Six Sigma provided the tools and techniques to get to Level 5 more quickly," said Hefner. "We were able to move from Level 3 to Level 5 in a year, which is directly attributable to the knowledge and culture established by Six Sigma. All the engineers and managers understand process and process variation, as well as the importance of business value, the voice of the customer, and many other Six Sigma tools and techniques. Now everyone is using Six Sigma as part of their everyday job; you hear the terminology at meetings … and no one is shocked or surprised. That's just the way we talk now."

Accelerating Technology Adoption: Six Sigma and CMMI

Siviy leads a research team at the SEI examining how Six Sigma techniques have helped organizations such as Lockheed Martin and Northrop Grumman to adopt and institutionalize best engineering practices. Her team is concentrating on how Six Sigma accelerates the adoption of the CMMI models, as well as its applicability to software architecture practices, systems integration, information technology (IT) operations and security, and acquisition practices.

Accelerating technology adoption is important to many software and systems organizations, says Eileen Forrester, a member of Siviy's SEI team. "We're finding some great reports about faster results from new technologies if an organization is also adept at using Six Sigma. This may be because Six Sigma helps organizations to make better decisions about what to adopt given their business objectives and improves the chances that the adoption will be a success. Also—best of all—use of Six Sigma in tandem with a new technology is likely to produce credible data for evaluating that success."

Because Six Sigma is new to the domains of software and systems development, many organizations, Siviy said, are struggling with implementation. Common questions from organizations include

- How does Six Sigma compare with other improvement approaches, and how does it fit with my organization's other software process improvement initiatives?
- What evidence is there that Six Sigma is applicable to software and systems engineering?
- What will it take for me to implement Six Sigma in my organization, and how do I get started?
- How do I train software engineers in Six Sigma methods when Six Sigma training is largely focused on manufacturing?

Siviy's research aims to help organizations answer these questions. "Using our expertise in software process improvement, our goal is to collect tangible evidence and assets that can be used by other organizations who wish to approach things this way."

Tell Us About Your Six Sigma Experiences

The SEI is interested in your organization's experiences using Six Sigma. Please contact us.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

World Wide Web
www.sei.cmu.edu/?si

TSP Accelerates Software Quality Improvements at NAVAIR

PAMELA CURTIS AND BILL THOMAS

When Jeff Schwalb heard the SEI's Watts Humphrey describe the SEI Personal Software Process℠ (PSP℠) methodology back in 1995, he thought it might be the key to reenergizing the software-process-improvement efforts at the U.S. Naval Air Systems Command (NAVAIR), which develops, acquires, and supports the aircraft and related weapons systems used by the U.S. Navy and Marine Corps.

At the time, Schwalb, who works in the Software Resource Center at NAVAIR's China Lake site, was trying to teach software development teams to use the SEI Capability Maturity Model (CMM) for Software. "It was frustrating," he recalls. "We would teach groups about CMM, and people built the process and the documentation, but then they didn't use it." Schwalb attended the Software Engineering Process Group conference in 1995, hoping to pick up some tips. He attended a presentation in which Watts Humphrey explained a different way for engineers to precisely apply CMM process principles—using PSP. "I saw Watts and I thought: 'This is the way to do it. Don't make groups create a process, rather give them something to start with. Then they can revise the process as needed.'"

Later that year, Schwalb attended PSP training at the SEI and became the first PSP instructor at NAVAIR. When the SEI developed the SEI Team Software Process℠ (TSP℠) methodology to extend PSP's disciplined engineering approach to teams, Schwalb was an early adopter.

The PSP provides guidance on how individual engineers can continually improve their performance. The TSP provides guidance about how PSP-trained engineers can work as effective team members on high-performance teams. These methodologies can work together to allow organizations to produce quality software on schedule and on budget. Also, PSP and TSP can help organizations implement Capability Maturity Models, which focus on what organizations should do, but do not specify how to reach those goals.

Schwalb introduced PSP and TSP to NAVAIR's Software Leadership Team, which devises software-engineering policy and guidance. He also brought in Watts Humphrey from the SEI to make a presentation. The team "bought in to PSP and TSP as a way to quickly implement CMM-based process improvement," Schwalb recalls. That support ultimately resulted in a NAVAIR instruction that includes the statement: "Programs engaged in organic software development should use the Personal Software Process and Team Software Process for personnel training, project initiation, and execution."

Today, eleven TSP projects are underway at NAVAIR, seven are planned, and at least five more are being considered. Some of the projects that have benefited from TSP and PSP include the F-14D Tomcat, its successor the F/A-18 Super Hornet, and the AV-8B Harrier. For example, the AV-8B

Joint System Support Activity at China Lake, CA improved from CMM Level 1 to Level 4—the second-highest level—in only 30 months, less than half the usual time. Also, "the savings from cutting defects in the F-14D project more than paid for the TSP training," Schwalb says. The project's managers expected to save more than half a million dollars from reduction of defects in flight testing alone.

Using TSP has paid other dividends. Recently, NAVAIR managers found it relatively easy to combine two teams, one that had just finished working on the Operational Flight Program component of the last major fleet release of the F-14D and another working on the Hawkeye electronic surveillance aircraft that was understaffed. The key to success was the fact that both teams used the same process and could easily share information. "TSP and PSP gave us a common vocabulary to start with in terms of how we were going to do our work," says Timothy Chick, Hawkeye project co-leader. The methods also "define many of our day-to-day processes, which allows us to focus on communication and the technical challenges before us." Antonio Garcia, the project's other co-leader, likes the fact that with TSP, team project status is always available. "Knowledge of problems or slips with project plans as they occurred has been a tremendous plus," he says. "This was not possible in the past, without the weekly tracking of plans that is done in TSP."

Teams working on upgrades for the Hawkeye patrol plane are using the TSP.[1]

John Mishler, former technical director of NAVAIR's Naval Aviation Systems and Analysis Department, found the results of TSP projects impressive: "My experience with PSP/TSP has made me more aware of how little of what is claimed by numerous management theories is really based on empirical data. By contrast, the PSP/TSP approach is built on a firm foundation of personal and team motivation theory and software engineering research, and provides a sound practical application of statistical theory. The PSP/TSP method is backed by an ever-growing body of research and a practice methodology that is data based and produces consistent results. The TSP enables teams to manage requirements, to plan their work based on empirical data, and to work

1. U.S. Navy photo by Photographer's Mate 1st Class Jim Hampshire.

their plan using best-practice methods, such as earned value management. Projects using the PSP/TSP method have a proven track record of delivering products on time and within budget with dramatic quality improvements."[1]

Presentations given by Garcia, Chick, Schwalb, and other NAVAIR personnel at the September 2003 TSP User Group meeting are available at *www.sei.cmu.edu/tsp/tug-2003-presentations/?si*.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

World Wide Web
www.sei.cmu.edu/?si

1. Kaniss, Al & Crosby, Linda Lou. "CMM, TSP, PSP: A Winning Combination for NAVAIR Systems, Software." *Tester*, April 10, 2003. *www.dcmilitary.com/navy/tester/8_14 /commentary/22557-1.html*