Carnegie Mellon
Software Engineering Institute

SEI interactive

Volume 2 . Issue 4 . December 1999

Full Issue

In This Issue:

Feature

Columns

http://interactive.sei.cmu.edu

Messages

Features

Columns

SEI Interactive

Volume 2 . Issue 4 . December 1999

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

About the SEI

Mission

The SEI mission is to provide leadership in advancing the state of the practice of software engineering to improve the quality of systems that depend on software. The SEI expects to accomplish this mission by promoting the evolution of software engineering from an ad hoc, labor-intensive activity to a discipline that is well managed and supported by technology.

SEI Work

The SEI program of work is grouped into two principal areas:

- Software Engineering Management Practices
- Software Engineering Technical Practices

Within these broad areas of work, the SEI has defined specific initiatives that address pervasive and significant issues impeding the ability of organizations to acquire, build, and evolve software-intensive systems predictably on time, within expected cost, and with expected functionality. Visit the initiative page on the SEI Web site [http://www.sei.cmu.edu] for more information.

From the Director

The CMMI℠ Project Delivers

Stephen E. Cross

The Capability Maturity Model® Integration (CMMI℠) effort has reached a major milestone: On Aug. 31, 1999, the first of the CMMI models was released for public review. We thought that it would be a good time to revisit the CMMI project as a subject for SEI Interactive.

CMMI-SE/SW Version 0.2 is an integrated model for systems and software engineering improvement. With CMMI-SE/SW, organizations using different models for improving software engineering and systems engineering can now use the integrated model to coordinate efforts to improve in both disciplines.

Current work on the CMMI project began in fall 1997, when the Office of the Under Secretary of Defense for Acquisition and Technology requested that the SEI collaborate with government and industry on an integrated framework for maturity models and associated products. The CMMI effort is intended to meet the needs expressed by the systems and software engineering communities by reducing redundancy and eliminating inconsistency from the use of stand-alone models. The goal is to improve efficiency, return on investment, and effectiveness by using models that integrate disciplines that are inseparable in system development, such as software engineering and systems engineering.

CMMI-SE/SW is an improvement over previously released models. It builds on lessons learned about software process improvement since the release of Version 1.1 of the Capability Maturity Model for Software (SW-CMM) in 1991. CMMI-SE/SW includes improved process descriptions for requirements management; it makes risk management an explicit process area; and it introduces concepts such as reuse and product lines at lower levels of maturity than in SW-CMM® Version 2 draft C.

Pilots are now underway in nine organizations with an interesting mix of systems-software, systems-only, and software-only process improvement. By enabling users to select only those features that are needed, the model allows for this kind of organization-specific tailoring.

I thank the 70 men and women from industry, government, and the SEI who worked hard to make the on-time version 0.2 release possible.

In the September 1998 issue of SEI Interactive, our feature topic was the CMMI project. We devoted much of that issue to describing the CMMI effort and providing the rationale behind the integration of Capability Maturity Models (that material is available from the

SEI Interactive archives). In this issue, we delve more deeply into the structure of CMMI models with a look at the two ways that organizations can approach process improvement using CMMI models, the continuous and staged representations. We also gathered many of the CMMI experts who participated in our first Roundtable discussion to talk further about CMMI models, how they will affect the organizations that use them, and what the future holds for process-improvement standards.

From the Editor

Welcome to SEI Interactive

Welcome to the seventh installment of SEI Interactive, and our last issue of 1999. I won't say the last issue of this millennium because, like Arthur C. Clarke, I subscribe to the view that 2000 is actually the last year of the current millennium, and 2001 is the beginning of the next.

Still, the rollover to 2000 offers a good opportunity for some reflection and for making new year's resolutions. My resolution for 2000 is this: I resolve to encourage the readership of SEI Interactive to take greater advantage of this publication's interactivity. Specifically, I hope to build up the traffic in our discussion groups.

Every week, we receive notes from readers who write to our email address, interactive@sei.cmu.edu, asking questions about or providing insights on a software engineering subject. We welcome this correspondence, and we pass the notes along to the appropriate technical staff person at the Software Engineering Institute.

But I propose that you consider sending those notes, instead, to our discussion groups. As we do with email, we alert the columnist or other technical staff person when notes are posted, and usually they respond. The result is an engaging dialog that all readers can see, involving you, the members of the software engineering community, and the top people at the SEI, including Watts Humphrey.

There is a discussion group connected to each major feature and column in SEI Interactive:

- the quarterly feature
- the "This Week" poll and short article
- "The Architect" column
- the "COTS Spot" column
- the "Net Effects" column
- the "Security Matters" column
- the "Watts New?" column

Our readership has been climbing steadily since the first issue was published in June 1998; in fact, it has increased more than five-fold. We use the statistic for "distinct hosts served" each month on the SEI Interactive server as a useful, if inexact, measurement of total readership. In November 1999, our readership reached its highest monthly figure

yet: 5,168. Because that is a monthly figure, and each issue is current for three months, our total readership could be as high as 15,000 per issue, but we're comfortable with the idea that our per-issue readership to date is probably between 10,000 and 15,000. Considering the technical nature of SEI Interactive and its specialized readership, we are thrilled with that number, though naturally we hope it will climb even higher.

Along with our hopes for growing readership, we hope for growing interaction. So keep those notes coming--but please consider posting them to our discussion groups!

Bill Thomas
SEI Interactive editor-in-chief
Software Engineering Institute

The SEI Interactive Team:

Mark Paat, communication design
Bill McSteen
Bill Pollak
Barbara White

Thanks also to Sandy Shrum, the guest editor for this issue's Features section, and to all of our content reviewers, including Steve Cross, John Goodenough, and Mike Konrad.

Introduction

CMMI[SM] Models Revisited

The first model from the Capability Maturity Model® Integration (CMMI[SM]) effort has been released for public review. As SEI Director Steve Cross writes in **From the Director**, CMMI-SE/SW Version 0.2 "is an improvement over previously released models. It builds on lessons learned about software process improvement since the release of Version 1.1 of the Capability Maturity Model for Software (SW-CMM®) in 1991. CMMI-SE/SW includes improved process descriptions for requirements management; it makes risk management an explicit process area; and it introduces concepts such as reuse and product lines at lower levels of maturity than in SW-CMM Version 2 draft C."

CMMI-SE/SW Version 0.2 is available on the Web at http://www.sei.cmu.edu/cmm/cmmi/public-review/public-review.html.

With the achievement of this milestone, we at *SEI Interactive* decided to revisit CMMI models as a feature topic.

Our **Background** article, "CMMI: The Evolution of Process Improvement," provides a brief overview of the history of Capability Maturity Models, the need for integration of those models, and the status of the CMMI effort. Readers might also want to see the September 1998 issue of *SEI Interactive*, available through the **Archives**, in which our feature topic was also the CMMI project. We devoted much of that issue to describing the CMMI effort and providing the rationale behind the integration of Capability Maturity Models.

Our **Spotlight** article, "Continuous and Staged, a Choice of CMMI Representations," examines the two methods by which organizations can pursue process improvement using a CMMI model. The two representations are explained from the standpoint of two imaginary companies that manufacture electronic toys. Foo Toys chooses the continuous representation because it wants to focus improvement efforts in two predefined areas, risk management and the integration of components. Widget Toys chooses the staged representation because it wants to improve the company's overall development capability and wants to compare its process-improvement efforts against those of competitors that use the same model.

In the **Roundtable** interview, SEI staff and members of the CMMI Steering Group engage in a wide-ranging discussion about the CMMI project. Topics include CMMI models versus other improvement models, transitioning systems engineers to CMMI models, CMMI models and international standards, the effect on senior management of

implementing CMMI models for process improvement, and how organizations should and should not use CMMI models.

Finally, our **Links** feature offers a guided tour of information available on the Web about the CMMI effort and CMMI models.

Background:

CMMI: The Evolution of Process Improvement

As organizations struggle to be competitive and profitable, executives look for guidance. They read books about management and strategy and apply the principles that they learn. Sometimes they hire consultants to evaluate the inner workings of their organizations and suggest improvements. One approach to organizational improvement, the Capability Maturity Model® (CMM®) approach, relies on a collection of best practices in a particular discipline, such as software engineering, systems engineering, people management, and software acquisition. Each individual model is associated with assessment methods that can be used to evaluate the effectiveness of an organization's practices.

The SEI and Capability Maturity Models

In 1991 the Software Engineering Institute released the Capability Maturity Model for Software (SW-CMM) consisting of key practices organized into a "roadmap" that guides organizations toward improving their software development and maintenance capability. The SW-CMM approach was based on principles of managing product quality that have existed for the past 60 years. In the 1930s, Walter Shewhart advanced the principles of statistical quality control, which were further developed and successfully demonstrated in the work of W. Edwards Deming, Joseph Juran, and Phillip Crosby. These principles were adapted by the SEI into a foundation for continually improving the software development and maintenance process. Since its release, the SW-CMM approach has significantly influenced software process improvement worldwide.

The SEI also became involved in helping to develop additional CMM approaches in other disciplines, including the Systems Engineering Capability Maturity Model[1] (SE-CMM) and the Integrated Product Development Capability Maturity Model (IPD-CMM). As is the case with the SW-CMM and SE-CMM, the IPD-CMM addresses organizational and project management processes, but with a focus on ensuring the timely collaboration of all appropriate disciplines in the development and maintenance of a product or service.

1. Recently, the Electronic Industries Alliance Interim Standard 731 (EIA/IS 731), Systems Engineering Capability Model (SECM), was published. EIA/IS 731 is based on both the SE-CMM and the Systems Engineering Capability Assessment Model (SECAM) developed by the International Council on Systems Engineering (INCOSE).

The development of multiple capability maturity models for other disciplines was generally greeted positively. Process improvement expanded to affect more disciplines and helped organizations to better develop and maintain their products and services. However, this expansion also created challenges.

Ideally, various capability maturity models should work together harmoniously for the benefit of organizations wishing to apply more than one model to improve product quality and productivity. However, especially with respect to the capability maturity models for software engineering, systems engineering, and IPD, managers have found that overlaps in content and differences in architecture and guidance across these models made improvement across the organization difficult and costly. Training, assessments, and improvement activities often had to be repeated for each specific discipline, with little guidance on how to integrate such activities across these disciplines. Clearly, organizations needed a way to successfully and easily integrate their CMM-based improvement activities. The models themselves needed to be integrated.

The Need for CMM Integration

To respond to the challenges and opportunities created by the demand for a better integration of CMM models, training, and assessment methods, the Office of the Under Secretary of Defense for Acquisition and Technology initiated the Capability Maturity Model Integration (CMMI[SM]) project, which is co-sponsored by the National Defense Industrial Association. Experts from a variety of backgrounds and organizations were asked to establish a framework that could accommodate current and future models. For more on the organization and goals of the CMMI project, see the September 1998 issue in the SEI Interactive Archives.

Since February 1998, industry, government, and the SEI have been working to build an initial set of integrated models covering three disciplines: software engineering, systems engineering, and integrated product and process development. The models chosen as the primary sources for the initial set of CMMI models were

- Capability Maturity Model for Software (SW-CMM) version 2.0 draft C
- Electronic Industries Alliance Interim Standard 731 (EIA/IS 731), Systems Engineering Capability Model (SECM)
- Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98a

CMMI chief architect Roger Bate explains, "Integrating process improvement models is no easy task. The source models for the CMMI project use different approaches and architectures, and they cover some different topics. The CMMI product development team, as diverse as it is, achieved consensus on many tough issues. The result is a set of

process improvement models that can meet the needs of many organizations now and can grow to meet the needs of more organizations in the future."

The team's mission also included ensuring that all CMMI products comply with the International Organization for Standardization (ISO) 15504 standard for software process assessment, and preserving the improvement work achieved by organizations that used CMMI source models.

The CMMI project endeavored to preserve government and industry investment in process improvement and to broaden the application of improvement across the enterprise. In addition to improving the usability of the CMM approach in a wider set of disciplines, the CMMI concept uses common terminology, common components, common assessment methods, and common training materials. All groups developing and reviewing CMMI products consist of representatives from government, industry, and the SEI. An intensive review process ensures that the content of CMMI products is generally accepted by key groups before public review.

CMM Integration Status

The CMMI product suite includes CMMI models, a framework, training materials, and assessment methods. The CMMI framework states the rules and concepts that ensure CMMI products are consistent with each other—that is, that they are capable of being integrated.

The framework will enable users to select one or more of the disciplines available in the CMMI product suite and to choose the representation that best suits the organization. Currently there are two disciplines available, systems engineering and software engineering, and two representations, staged and continuous. In the continuous representation, which comes from systems engineering, process areas span levels rather than being defined with a maturity level, as in the staged representation used in the SW-CMM approach. For more on the staged and continuous representations, see the Spotlight article in this issue.

The first CMMI model available for public review and comment, CMMI-SE/SW v0.2, contained best practices for management, software engineering, and systems engineering. The public review and comment period ended November 30, 1999. This model is available on the SEI Web site at
http://www.sei.cmu.edu/cmm/cmms/cmms.integration.html

The model, assessment method, and introductory training are currently being piloted by volunteer organizations. The product development team is processing change requests

from the public review. These requests will guide the modification of the model. A release of a model for use in process improvement will be available in June 2000.

Jack Ferguson was the CMMI project manager until recently and is currently the Director of Software-Intensive Systems for the Deputy Under Secretary of Defense for Science, Technology, and Logistics. Ferguson says the new CMMI-SE/SW v0.2 model "is not just the combination of models from the software and systems engineering disciplines; it represents an evolution in the focus of the capability maturity model approach. Previous models were intended to improve the processes of individual disciplines. This new model is intended to improve the process of developing or sustaining a product or service, regardless of the discipline. The CMMI effort is a major change in the process improvement landscape, with the potential to dramatically improve all disciplines involved."

Results of Studies

Studies documenting the results of using the SW-CMM approach for process improvement have demonstrated significant cost savings, and the following benefits are expected from the use of the CMMI models:

- reduced costs
- increased predictability of project costs and schedules
- higher quality and productivity
- shorter cycle time
- increased customer satisfaction
- higher employee morale

Spotlight

Continuous and Staged, a Choice of CMMI Representations

Sandy Shrum

What is a CMMI model representation? The answer requires an explanation of the structure of CMMI models. The basic building blocks in every CMMI model are called "process areas." A process area does not describe *how* an effective process is executed (e.g., entrance and exit criteria, roles of participants, resources). Instead, a process area describes *what* those using an effective process do (practices) and *why* they do those things (goals).

In a Capability Maturity Model®, process areas can be organized into one of two "representations," a *continuous* representation or a *staged* representation. For example, the Electronic Industries Association's Interim Standard 731, Systems Engineering Capability Model (SECM) is a model with a continuous representation. The Software Engineering Institute's Capability Maturity Model for Software (SW-CMM®) is a model with a staged representation.

To illustrate why an organization might choose one representation over the other, imagine two companies, Foo Toys and Widget Toys. Both companies manufacture software-intensive toys and, until now, have not pursued process improvement.

The management of Foo Toys wants to improve how the company handles risks and integrates product components. Management is happy with how the company's other processes are operating and so decides to focus on those two process areas only. Foo Toys' management chooses the continuous representation. Using that representation, Foo Toys will concentrate on only those process areas that relate to risk management and the integration of components. When Foo Toys achieves both the specific goals for a process area and the general goals associated with all levels equal to or less than a particular capability level, it achieves the capability level for that process area. (Goal achievement is determined by a review of the practices associated with the goal.) So, if Foo Toys successfully achieves both the specific goals for product integration and all the capability level 2 and 3 goals, it could be said that, "Foo Toys is level 3 in product integration."

The management of Widget Toys, on the other hand, wants to improve the company's overall development capability and sees many areas requiring attention. Recognizing the many interdependencies across process areas, Widget Toys' management chooses the staged representation. Using that representation, Widget Toys will concentrate on the process areas at maturity level 2, thus establishing its project management processes. When Widget Toys performs the practices in these process areas successfully, it also achieves the corresponding goals. When Widget Toys achieves all of the goals of a process area, the process area is satisfied. For Widget Toys to successfully achieve a

maturity level, it must satisfy all of the process areas through that level. If Widget Toys satisfies all of the process areas through maturity level 2, it could be said that "Widget Toys is maturity level 2."

By design, the granular information contained in the two CMMI model representations is virtually identical. However, each of the representations provides benefits that will be valued differently by organizations.

In CMMI models, process areas describe key aspects of such processes as configuration management, requirements management, product verification, systems integration, and many others. Let's examine the two representations in more detail.

Continuous Representation

In the continuous representation of a CMMI model, the summary components are process areas. Within each process area there are specific goals that are implemented by specific practices. Also contained in the continuous representation of a CMMI model are generic goals that are implemented by generic practices.

Specific goals and practices are unique to individual process areas, whereas generic goals and practices apply to multiple process areas. Each practice belongs to only one capability level. To satisfy capability level 2 for a process area, Foo Toys must satisfy the specific goals and level-2 practices for that process area as well as the level-2 generic goals for that same process area.

Staged Representation

In the staged representation, the summary components are *maturity levels*. Within each maturity level there are process areas, which contain goals, common features, and practices. For Widget Toys, the practices serve as guides on what to implement to achieve the goals of the process area.

In a staged representation of a CMMI model, practices are categorized into common features:

1. *Commitment to perform* includes practices that ensure that the process is established and will endure. It typically involves establishing organizational policies and leadership.
2. *Ability to perform* includes practices that establish the necessary conditions for implementing the process completely. It typically involves plans, resources, organizational structures, and training.

3. *Activities performed* includes practices that directly implement a process. These practices distinguish a process area from others.
4. *Directing implementation* includes practices that monitor and control the performance of the process. These typically involve placing designated work products of the process under configuration management, monitoring and controlling the performance of the process against the plan, and taking corrective action.
5. *Verifying implementation* includes practices that ensure compliance with the requirements of the process area. These typically involve reviews and audits.

Capability Levels Versus Maturity Levels

The continuous representation consists of *capability levels*, while the staged representation consists of *maturity levels*. The main difference between these two types of levels is the representation they belong to and how they are applied:

- *Capability levels*, which belong to a continuous representation, apply to an organization's process-improvement achievement in individual process areas. There are six capability levels, numbered 0 through 5.
- *Maturity levels*, which belong to a staged representation, apply to an organization's overall process-improvement achievement using the model. There are five maturity levels, numbered 1 through 5. Each maturity level comprises a set of goals that, when satisfied, improve processes. Maturity levels are measured by the achievement of the goals that apply to a set of process areas.

Table 1: Capability Levels and Maturity Levels

Level	Continuous Representation: Capability Levels	Staged Representation: Maturity Levels
Level 0	Not Performed	N/A
Level 1	Performed	Performed
Level 2	Managed	Managed
Level 3	Defined	Defined
Level 4	Quantitatively Managed	Quantitatively Managed

Level 5	Optimizing	Optimizing

When Widget Toys uses the staged representation, it will evaluate its progress using the same basis as all other organizations that use the same model with the staged representation. Although Widget Toys can pursue process improvement at any pace it wishes, the basis for evaluating its progress will be exactly the same.

Using the staged representation, Widget Toys can identify the *maturity levels* through which it can evolve to establish a culture of engineering excellence. Each maturity level forms a necessary foundation on which to build the next level.

Using the continuous representation, Foo Toys can produce a *capability level profile* (i.e., a list of process areas and their corresponding capability levels). Types of capability level profiles include the following:

- An *achievement profile* represents the current achieved capability level in selected process areas at Foo Toys.
- A *target profile* represents the capability levels that Foo Toys wishes to achieve.

Maintaining capability level profiles throughout the process-improvement life cycle enables the engineering process group at Foo Toys to demonstrate its progress to management as well as guide its process-improvement activities.

A target profile can reflect the unique needs of the organization (called *target staging*) or it can reflect the levels used by the staged representation (called *equivalent staging*). Equivalent staging permits benchmarking of progress among projects, organizations, and other enterprises.

Selecting a Representation

When making the decision about which architectural representation to use for process improvement, Foo Toys and Widget Toys would consider the comparative advantages of each approach as represented in Table 2:

Table 2: Advantages of Using Each Model Representation

Continuous Representation	Staged Representation
Grants explicit freedom to select the order of improvement that best meets the organization's business	Enables organizations to have a predefined and proven path.

objectives. Enables increased visibility into the capability achieved within each individual process area. Supports a focus on risks specific to individual process areas. Affords a more direct comparison of process improvement to ISO 15504 because the organization of process areas is derived from 15504. Allows the generic practices from higher capability levels to be more evenly and completely applied to all of the process areas.	Builds on a relatively long history of use. Case studies and data exist that show return on investment. Permits comparisons across and among organizations. Introduces a sequence of improvements, beginning with basic management practices and progressing through successive levels, each serving as a foundation for the next. Summarizes process-improvement results in a simple form—a single maturity-level number.

Foo Toys decided to choose the continuous representation because it wanted to focus improvement efforts in two predefined areas.

Widget Toys decided to choose the staged representation because it wanted a clear path to process improvement that provides an easy comparison to competitors that use the same model.

About the Author

Sandy Shrum is a member of the CMMI product-development team and has been a senior writer/editor at the SEI since 1995. Before joining the SEI, she spent eight years with Legent Corp., where she was a senior information developer, a member of a software-development team, and a member of Legent's IT organization. She has an MA in professional writing from Carnegie Mellon University and a BS in business administration and marketing from Gannon University.

Roundtable Interview on CMMI

Moderated by Sandy Shrum

In this article, SEI staff and members of the CMMI Steering Group engage in a wide-ranging discussion about the CMMI project. The participants are:

Mike Phillips, director of the SEI Transition Enabling Program, program manager for the CMMI product-suite development and transition, and custodian of that suite of tools for enterprise process improvement.

Bob Rassa, director of System Supportability at Raytheon Electronic Systems, deputy chairman of the CMMI Steering Group, chairman of the NDIA Systems Engineering Committee, the Industry sponsor of CMMI.

Hal Wilson, vice president for e-business with Litton PRC and a member of the CMMI Steering Group.

Dennis Ahern, advisory engineer, Software and Processing Systems Department, for Northrop Grumman Corp., Electronic Sensors and Systems Sector (ESSS), and deputy program manager for the CMMI Product Development Team.

Joan Weszka, manager, process and program performance, for Lockheed Martin Mission Systems, Systems & Software Resource Center, and a member of the CMMI Steering Group.

Jack Ferguson, director, software-intensive systems, Office of the Deputy Under Secretary of Defense (Science, Technology, and Logistics). Prior to this, he was the CMMI project manager.

Sandy Shrum (moderator), member of the CMMI Product Development Team and a senior writer/editor in the SEI Technical Communication group. She is guest editor of this issue's Feature section on CMMI.

The topics of discussion include

- CMMI models versus other improvement models
- Transitioning systems engineers to CMMI models
- CMMI models and international standards
- Effect on senior management
- How organizations should and should not use CMMI models

CMMI Models Versus Other Improvement Models

Sandy Shrum: What would you say are the advantages of using a CMMI model over other process-improvement models?

Mike Phillips: The key letter in CMMI is the "I." The whole reason we are on this journey is to create a better improvement tool that *integrates* a number of important disciplines and ways of doing business across the enterprise. We think there is a particular advantage to bringing complementary disciplines into one fold and making sure that process improvement benefits the entire enterprise. Specifically, in the initial phase, we are bringing the engineering disciplines together.

Bob Rassa**:** The CMMI process-improvement methodology will help eliminate stovepipe environments and redundancy in a company's process-improvement environment. When you're dealing with separate stovepipe models, you tend to build processes around those. That tends to cause redundancy because you get a systems engineering group that will build their systems engineering processes and a software group that builds their software processes. There is also the issue of a sense of methodology. The CMMI product suite will have a common assessment methodology that will help to simplify the assessment process.

Hal Wilson: The integration of software and systems engineering in our organization is the key to the value of the CMMI models. Having started the process of continuing the improvement of software as well as beginning the process of systems engineering process improvement, it became very clear to us that there were, as Bob indicated, some elements of stovepiping in that process and a lot of redundancy. We see the big value in the fact that the CMMI models bring the two together in a cohesive fashion and, in fact, are more cohesive than we expected they could be. That is a big advantage to us. Another advantage is the existence of a more complete software product engineering model to match up with the elements of systems engineering. That is a tremendous advantage and improvement over the previous software-engineering-only model.

Dennis Ahern: I support everything my colleagues have said. Where I work [at Northrop Grumman], we began to develop an integrated engineering process several years ago. Actually we call it "Integrated Enterprise Process," and it includes software and systems engineering plus a number of other engineering disciplines. We recognized this need long ago, and I think we were not alone. In talking with colleagues at other companies, I know that other organizations have started down this path. Indeed, the Federal Aviation Administration recognized a similar need years ago. I think that integrated process improvement clearly represents something in which industry has demonstrated an interest and wishes to see occur.

Joan Weszka: I'd like to think that the use of the CMMI model has some of the same advantages as using the Integrated Product and Process Development (IPPD) method. Specifically, we're promoting timely collaboration of all the disciplines throughout the life cycle—not just the product development life cycle, but also the process-improvement life cycle, as defined by the SEI IDEAL model. As a result of discipline collaboration, we'd expect that there would be a yield of both quality improvement and cost savings. Like Northrop Grumman, Lockheed Martin has also integrated its processes across the corporation. We have integrated them from both a software and systems engineering perspective and from an IPPD perspective. What we would expect to see as an advantage to using the CMMI model is the ability to adapt an industry standard to our integration paradigm.

Jack Ferguson: CMMI allows a much broader process improvement rather than being stovepiped into little communities. Also, especially for the software people, it provides more emphasis on software engineering versus the management emphasis in the software model. As we talk to software engineers about the CMMI approach, we discuss the relatively modest expansion of software product engineering to the other process areas that we inherited from systems engineers. You see a lot of people's heads nod when we say that we've turned many of these activities that were of fairly limited scope in the old software model into process areas and fleshed them out with more detail. The technical people on the software side said they appreciate that, and that's where we should be headed. We talked to the process-improvement people who are active in Europe; and they said that they've been telling their customers for a long time that they need to worry about not only the management of process improvement, but the technical aspects of improving software development processes.

Shrum: Let's compare the CMMI model to the existing models. If I'm with a company that is already using the Software CMM Version 1.1, how would you persuade me that I should be using the new integrated model?

Weszka: I think the biggest advantage is that the CMMI models provide a growth path for enterprise-wide process improvement. For example, if today you're using the Software CMM for process improvement, in the future you might want to expand your scope—into systems engineering, into IPPD, or into other disciplines that will be included under CMMI models in the future. As these additional disciplines are added, the organization can expand its process-improvement efforts to maximize the benefit and the return on investment.

Ahern: Like anything, a Capability Maturity Model can be improved. Version 1.1 of the Software CMM has been used for some time. Before the CMMI effort began, it was already recognized that there were areas where the Software CMM could be improved. In fact, there were drafts of another release that we on the CMMI team used as one of our sources. We think that the CMMI model that we released for public review incorporates a

lot of the improvements that the community using Version 1.1 had already recognized and wanted to see in the next release. We're hoping that as the CMMI model gets reviewed, we will learn that we were successful in putting those improvements in, because I think we have been successful in large part.

Wilson: I would agree with both of those statements. I think they're probably the key ones. As I mentioned before, we believe that the extension of the software product engineering activities within the software-only model to be more specific in the areas of validation and verification, and other areas have really made an improvement in the model. That's another reason to say that, even if you're just doing software engineering and just using Version 1.1, it would be worth moving to the CMMI software-only version.

Rassa: Recognize that Software CMM Version 2.0, which was not released, was also going to raise the bar for software process improvement. The new CMMI model used as its baseline document SW-CMM Version 2.0, so the new model raises the bar and that may deter some organizations from going that way, from adopting the CMMI model. They're going to say, "Well, you know, I have a lot of extra work to do." That is a shortsighted view because the purpose of Capability Maturity Models and CMMI models is continuous process improvement. We feel that the CMMI model offers numerous enhancements over Software CMM Version 1.1. It should be advisable for all activities to at least have a look at it. One obvious enhancement is that you now can assess yourself in a continuous environment, as opposed to merely a staged environment. That may benefit a number of companies who don't need to add all attributes of Level 2, but they want to do some of Level 3 or Level 4. The continuous environment allows them to better tailor the CMM to their application. (For more on the staged and continuous representations of CMMI, see the Spotlight article in this issue.) Of course, with increased government emphasis on process improvement, that may provide additional incentive for adoption.

Phillips: I like Joan's comment about the growth path and the notion that what is now in the CMMI model is in fact an excellent growth from what was in 1.1. The areas that aid the engineering discipline are now fleshed out in better ways, and I think that capitalizes on Hal's point. Here are things that the software engineering community was doing, perhaps without adequate recognition of the quality and the ability to address these broad engineering requirements—for example, validation and verification.

Weszka: I'd like to add one other comment, and that concerns the impact to the bottom line. If an organization is in fact using the Software CMM today and another part of that organization is using another model—for example, it might be using EIA/IS 731—adopting the integrated model, we would expect, would result in improved efficiency in their process-improvement efforts. So there may be strong business reasons for an organization to move from its existing use of the Software CMM to the CMMI model, if there are other stovepiped efforts underway.

Ahern: Historically there has been an issue in a lot of places about the differences between software engineering organizations and systems engineering organizations. Even though it is essential for these two groups to cooperate—and in many instances they do, quite successfully—still they have been in different worlds. I think the CMMI model offers an opportunity for those who want to bring these organizations more closely together. It provides them with more common terminology and allows them to communicate better in an integrated environment.

Weszka: One of the biggest differences between the software engineering and systems engineering communities regarding process improvement has been the fact that because the Software CMM is a staged model and EIA/IS 731 is a continuous model, the two engineering groups have different process-improvement paradigms. By adopting one of the CMMI "architectures," either the staged or the continuous, and using the integrated software and systems engineering model, combined engineering groups can bridge some of the differences that have existed in the past and establish a more common culture for process improvement.

Rassa: A simple change like changing from the *software* engineering process group to the engineering process group causes a paradigm shift that may enable some of that cross-organizational building of the community of practice for broad, enterprise process improvement to be more effective.

Wilson: Most organizations do both software and systems engineering, in the sense that there are elements of systems engineering in most big software programs, even when they're considered software only. One of the things you find when you break down the barrier and ask what's in common is a tremendous benefit return. In the case of the integrated SE/SW CMMI model, or even just the software version, using terminology that is normally used in systems engineering raises the understanding of the software engineers to the importance of doing some of those activities. And while they're mentioned and covered in the software product engineering part of Version 1.1, they become, I think, more defined and clear in the CMMI versions. They are effectively the same between the software and systems engineering group requirements, regardless of whether you are using the continuous or staged model in the CMMI. I think that's its real power from a practical point of view.

Transitioning Systems Engineers to CMMI Models

Shrum: We've touched on systems engineering. Is there anything in particular that you would use to persuade systems engineers to transition to CMMI?

Wilson: Our organization adopted 731 as our systems engineering standard when it became available. One of the issues was that it's a slightly different model in the sense

that it's a continuous model. What we use in Version 1.1 of the Software CMM is a staged model. So we were faced with some transition issues. From the point of view of an organization that was just going to use 731, almost every major system activity that you perform today includes some software. Because of the way we built things to meet our cost objectives—and without having a combined CMMI—it would be very difficult for a pure systems engineering view to be as effective in our business areas. There may be some organizations that indeed do *just* systems engineering and don't consider software, but I think there are probably very few of them. Having used both models, I would encourage an organization to look toward a combined CMMI model as the way to move, particularly since the new version of CMMI really makes it simple to provide both because they use almost a common view.

Rassa: Raytheon also has begun to adopt 731 because it was essentially *de rigeur*, and we've done four assessments. But we find that what's lacking is our integration of software and systems engineering processes, which is what CMMI does for us. I agree with Hal's point that most of the government or aerospace industry really does systems engineering and software engineering, among other disciplines, and the integration activity that CMMI is expected to bring will have great efficiency benefits to us and to other companies. That was the whole notion behind CMMI to start with.

Phillips: The points that I would like to insert here build on a comment that Bob made earlier about the advantages—the various pros and cons between the staged methodology and the continuous. I've just dealt with an organization that has a strong systems engineering community within it, but is trying to decide how to approach the problem. They're looking at the desirability of being able to move, if you will, vertically, which calls for the continuous notion because there are some things that they really want to be able to accelerate above others to their management, to meet their management business goals. But they also have this difficulty of making sure that they get the right ones first, which is more of the staged paradigm. So this notion of being able to look at a model that gives you both representations and use the guidance for the equivalence, and pursue the ones that help you the most to meet your business goals, gives it a flexibility that no other approach that I've seen has.

Ferguson: I agree, especially about allowing enterprise-wide improvement versus individual stovepipes in the various disciplines. The current software and systems engineering models foster improvement in each discipline, but CMMI allows both the systems engineers and the software engineers to communicate horizontally, and not just vertically. It also allows the process-improvement efforts to occur in a much more integrated fashion across the organization rather than vertically.

Rassa: Let me throw in another possibility here. It strikes me that one of the advantages of moving forward in the model is that the 731 instantiation focused, for good reasons, on those things that were considered normative and did not give more information on how

one might proceed against the various practices being called out. That was a very different paradigm from what existed on the software engineering side. Now if someone says, "Give me an example, tell me a little bit more," it is available in the Volume 2 portion. So the informative Volume 2 includes the normative material and also provides the ability for folks who need more guidance on how to proceed to get it from that side, from the overall model.

Weszka: Some people would consider that a downside to the CMMI model specifically. They could say, and I have heard it said, that 731 is written in a very concise, clear way; the practices are well articulated; and there isn't a need for a lot of explanation.

Rassa: We have the two volumes, I think, for a very good reason.

CMMI Models and International Standards

Shrum: How do you see the CMMI model fitting into the international standards environment?

Rassa: Interesting question. The CMMI Steering Group has established the general philosophy that we will look at issuing CMMI first as a national standard and then secondly as an international standard. The reason for progressing to an international standard is to give it some durability and credibility, particularly in an international marketplace, when U.S. companies are doing business abroad or partnering with companies abroad. Having CMMI as an international standard will, we believe, facilitate that process. However, I should also point out that we are just beginning the process of making CMMI an international standard, and it will be quite a number of years before that will have been consummated. A lot of arrangements will have to be made and a lot of planning with the standards organization that would do the issuance. We have initiated discussions with the GEIA concerning the national standard, the U.S. standard, and that's as far as we have gone so far—just discussions initiated. We believe that having an international standard is advantageous; however, we have yet to assess all of the pros and cons. Before we finalize the move to an international standard, I believe we would need a lot more input and have some serious discussion about it. We have an indication that to create an international standard is a very tedious process and might disturb some of the CMMI plans that we have in place, in terms of our custodianship and how we authorize assessors and trainers.

Wilson: One of the things that the steering group and the PDT (product development team) recognized early in the process of defining the requirements for CMMI was the need to recognize the input from ISO 15504. The effort that the PDT put into mapping and adjusting the CMMI so that it would have a relatively smooth means of mapping into 15504 is important to recognize. That's its primary means of mapping into international

standards today, prior to any activity to actually make the CMMI itself an international standard.

Ahern: Indeed, we did have the requirement and we did keep our eye on that standard, and we have the goal of making CMMI consistent with it. However, it is a moving target in that the standards, not only ISO 15504 but also other relevant ISO standards, are on the move and changing as we speak. So it's going to be an ongoing issue to keep CMMI well positioned and consistent with relevant international standards. Obviously we wish to avoid creating a situation in which U.S. companies that conduct business internationally are at a disadvantage because they work to a CMMI model. A CMMI model should not put up a barrier and should not make it more difficult for a company to be consistent with international standards. So I think it's an important goal and one that needs to be constantly reevaluated as we proceed in the next few years.

Phillips: Let me toss in a couple of thoughts. Since I joined the SEI four years ago and began attending the annual Software Engineering Process Group meetings, I was seeing increasingly international participation, to the point that it seemed to me that the Software CMM was becoming a de facto international standard. But at the same time, things like 15504 were requiring some things that the existing model didn't accommodate, at least not easily. So to me one of the values of our approach to CMMI has been to recognize the desirability of producing something that will meet those needs more effectively, to accommodate the 15504 intention and therefore make it a logical move from a de facto standard around this set of ideas to something that can then be embraced first as a national standard, as Bob says, and over sufficient time, potentially as an international standard.

Weszka: I agree that it is a critical goal to move the CMMI models into the standards arena. Of course, harmonization is really important, and we've already discussed the amount of work that has been done to date to ensure that we are harmonizing with other standards as they evolve. The SEI, as the custodian of the model, will be a key player in taking the CMMI forward into the standards arena.

Effect on Senior Management

Shrum: What effects will senior executives see as a result of using a CMMI model?

Phillips: Let me start with the one that I found to be so persuasive in coming to the SEI in the first place, having been an executive in the Air Force before. I was talking with a member of an engineering group who said they had adopted the CMM as an approach and had reached, in the staged model, Level 3. The effect was that the work force now understands far more clearly how the job needs to be done and there's a stability in doing things in an understood and effective way that simply didn't exist in the chaos that

existed before adopting the models. So to me the senior executives who have not been in a process-improvement domain associated with this kind of building capabilities and maturities will see great impact on the effectiveness of the work force.

Weszka: I would agree that senior executives should observe additional business value and return on investment for process improvement as their process-improvement efforts are integrated using the CMMI models. But in addition to the tangible results, they should also see the intangible benefits accruing over an expanded segment of their business. For example, improved employee morale, reduced attrition, better communications—all those additional advantages from process improvement should be visible across a larger business base. Senior executives who have already sponsored process improvement typically would have organizational cultures that support measurement already. So they can measure the results of using the CMMI model, including improvements in quality, productivity, cycle time, and customer satisfaction. In addition, they can use channels such as employee satisfaction surveys and the number of suggestions for improvements to measure the intangible results. And, of course, they can continue to use the measurement programs that they have in place to measure the quantitative tangible improvements to their business.

Wilson: In addition, I think in an organization that has traditionally been a Software CMM supporting organization moving to a systems engineering supporting organization, one of the big-value elements that senior executives will see in the CMMI is that both software and systems engineering elements of the organization will begin to speak with a common lexicon within the process areas. Rather than saying, "Well, we're different because...," the integrated CMM really brings together process definitions in a way that all can speak the same language. While the individual processes may vary slightly, internal to their suborganizations, the basic senior executive view will be the same, which should make it much easier to operate in that type of organization.

Rassa: I'd like to give a different view. Most senior executives don't tend to focus on a number of the things that have been mentioned. What they tend to focus on is the bottom line, and we hope that senior executives will see an improvement in their win ratio of business pursuits due to the adoption of CMMI, which is caused by several things that some of my colleagues have mentioned. They also should note improved profit or performance because of the process-improvement aspects of CMMI that are given to each program that is won by an organization so that they can have a greater assessment of risk. And that risk should be lower because of the process-improvement methodology that we've put in place. But the senior executives also need to recognize that there may be a slight investment required for implementing CMMI and that this should be clearly offset dramatically by the improved win ratio and the improved profitability.

Weszka: In addition to the things that Bob mentioned, there is the improved predictability, which of course is a fundamental concept underlying all of the Capability Maturity

Models. That's another thing that executives should expect to see: improved predictability in program performance across the enterprise as process improvement is focused on a larger segment of the business.

Ahern: If you look at some of the differences between software organizations and systems organizations within a given company, in a sense the software organization is the new kid on the block. The other engineering disciplines are—and view themselves as—more well established, whereas software engineering is a new and increasingly important discipline that perhaps is still trying to discover ways to be fully accepted as a real engineering area. I think that one of the things that software engineering has to gain from adopting a CMMI approach is to become better integrated into the larger engineering context. At the same time, it seems that software engineering has taken the lead relative to other engineering disciplines; software engineers have recognized—as they have in the manufacturing area—that getting processes under control and especially under quantitative control is very valuable. Applying that, as Joan said, across the entire enterprise can potentially help a great deal in terms of improving competitive position.

Ferguson: I think that senior executives will start to see enterprise-wide process improvement expand beyond software engineering and systems engineering. The premise of CMMI is that it allows CMMI process-improvement techniques to apply to any domain. We've just focused on the two, software and systems engineering. I think senior executives will see that this common way of improving processes can apply to everything—not just technical development. It can apply to finance; it can apply to project management. So its use could explode dramatically from the current relatively narrow technical communities to areas across the entire organization.

How Organizations Should and Should Not Use CMMI Models

Shrum: How do you see CMMI models being used by organizations, and how would you warn organizations *not* to use CMMI models?

Wilson: From the perspective of where you start and where you might go, obviously the concept of the Capability Maturity Models, and the CMMI models in particular, is to focus on the initial definition of the processes that make up a mature organization. Once that is established, the focus moves to improving and tuning those processes over time with the appropriate feedback. One of the things I would recommend to an organization starting out is to focus on the internal process value of the business processes that they perform and not use CMMI structure to usurp them. Because CMMI models are really intended—as I see it and as we've used them—as guidance documents. They help to ensure that your processes include all of the critical elements that a capable, mature organization needs, and help ensure that you don't leave any out. The CMMI approach isn't intended to force an organization to put some emphasis where no emphasis is

demanded in your internal process. In the future, as things change and organizations adapt, you'll find that the core processes remain. You can keep them and map them to the variations that occur. Having gone through several versions of systems engineering process models, and also going from the SW-CMM Version 1.1 orientation and moving both toward a CMMI model, we've found that our processes remain stable because they were good processes to start with. They were tuned over time in the Version 1.1 model and in the other systems engineering models. But we're now mapping our internal processes to the CMMI model. It becomes a lot easier because those processes still remain our key business processes. If you look at using a CMMI model that way, you focus on your real values and the return you get from your processes. That is where the real value is achieved.

Ahern: The CMMI product development team had the requirement to create a model that would be good for process improvement across multiple disciplines, with systems engineering and software engineering as starting points. An organization needs to think about the different ways in which a model might help with process improvement. On the one hand, you could be focused on benchmarking: determining where you stand relative to either where you were two years ago or where you want to be three years hence, or vis-à-vis your competitors or perhaps across a larger industry. So one use of the model for process improvements is to try to set benchmarks for where you are and where you want to go. On the other hand, there are uses of the model that allow greater attention to detail. You might not look at the whole model but instead just use the parts of it for more narrowly focused efforts at process improvement. I think it's important for managers to realize that they have this second option. Indeed the more narrowly focused look might have a larger payback because it allows you to take a particular area of concern, examine it more thoroughly, and do more problem identification than you would be able to do if you took the whole model, which is very broad and all-encompassing. If you only are involved in benchmarking efforts and don't do the other, I believe you're missing an opportunity to identify process problems and their root causes.

Rassa: In summary, I think a CMMI model is designed to be used to guide a company's process improvement, either at an organizational level or a programmatic level. A company should recognize that at a programmatic level we do encourage tailoring to suit the specific program. You shouldn't try to look at CMMI as the gospel, that you absolutely have to follow it to the letter. You're encouraged to tailor it as it applies to your organizational needs. It's supposed to guide your company's internal processes. Most companies have internal processes, software operating instructions, or integrated product development processes, et cetera. And a CMMI model is the map to where your processes should lead. But you should still build your processes around your organizational needs.

Weszka: I agree with Bob's points. The CMMI models should be used for internal process improvement. That includes process definition, improving the processes once they're

established, and then assessing them. I would suggest that although the CMMI models can be used strictly for benchmarking purposes, a better approach is to use them both for improvement as well as for benchmarking. On the other hand, they should not be interpreted prescriptively. That, to me, is the biggest danger—that organizations could interpret the practices as mandatory requirements. Instead, the model should be tailored to the organization's business objectives; that must be a consideration whenever the models are used.

Phillips: I'd just like to reinforce that. We often find, here at the SEI, a desire from various organizations to have us tell them what processes to use and help them choose exactly the processes that will meet a particular process area within the existing CMMs. We seek to dissuade that because it's so important to determine the business value, to determine the processes within the company that meet their needs and then to see how those interrelate as a cross-check against things that have been found across other industries. But we would wish to dissuade anyone from saying, "Let me just find those things that meet some particular process model and only do those particular things for improvement."

Ferguson: I'd like to go back to something that Bob Rassa said earlier and emphasize that process improvement using these models requires up-front effort and resources. Just taking one of these models and laying it on the back of the existing work force and saying, "Here, go implement this," is a surefire means for disaster. Resources have to be provided to define the processes, define the process improvement steps, and implement those. We find that over and over again in the government that adequate resources are not provided up front. We look at the anticipated savings, but don't provide the seed money to gain those anticipated savings.

Rassa: CMMI models should not be used solely to achieve a rating to satisfy a perceived customer requirement.

Phillips: I think we'd all agree with that. That's just a clear statement of the potential misuse. The value of the CMMI effort is to improve the processes within the organization. Benchmarking is desirable, but it should not drive the result.

Wilson: I think if you look at the long-term value of following a Capability Maturity Model or a CMMI model, the organization changes with the beginnings of process improvement. As Jack pointed out, if you initially invest correctly and focus on the value received, the improvement you make, and how that improvement actually adds value to your organization, you will make the right start. Then the question of measurement of internal progress, adapted based on which model you choose and which representation you choose, does have value internally because you're always comparing against a common element: your internal view. But that's not the real reason you do it. You measure because you watch the results and see the value returned, not the number assigned to a level. The number or level is really an indicator of how well you're

progressing internally. Are you achieving the milestones and time stages you've set in terms of your overall improvement plans? In that sense it's a valuable tool. I believe that misuse occurs when you start to compare unlike elements. Even within a corporation, where multiple organizational entities are independently assessed, it's very hard to compare results from one entity to another. They may be using a common organizational process, but tailoring processes to specific organizations makes comparison very difficult, even within a single corporation.

Links

CMMI[SM] Resources on the Web

The links below, both internal and external to the SEI, provide information about the CMMI[SM] Project.

Overview

Capability Maturity Model® Integration (CMMI)
http://www.sei.cmu.edu/cmm/cmms/cmms.integration.html

CMMI FAQ Version 1.0
http://www.sei.cmu.edu/cmm/cmmi/comm/FAQ.html

CMMI-SE/SW Version 0.2
http://www.sei.cmu.edu/cmm/cmmi/public-review/public-review.html.

Articles

CMMI Supports Enterprise-Wide Process Improvement
http://www.stsc.hill.af.mil/crosstalk/1998/jul/publisher.asp

Mappings of the Capability Maturity Model
http://www.stsc.hill.af.mil/cmmi/index.asp

Software Productivity Consortium on CMMI
http://software.software.org/quagmire/descriptions/cmmi.asp

Capability Maturity Model Process Improvement
http://www.stsc.hill.af.mil/CrossTalk/1998/may/cmmi.html

News

Software Process Newsletter
http://softeng.cs.mcgill.ca/process/spn.html

Stakeholder/Review Team
http://www.sei.cmu.edu/cmm/cmmi/sr_list.html

Announcements
http://www.processimprovement.com/CMMv2.htm

Nominations
http://www.sei.cmu.edu/cmm/cmmi/comm/call.nomin.html

Specification

CMMI: A Specification Version 1.3
http://www.sei.cmu.edu/cmm/cmmi/specs/aspec1.4.html

Steering Group

Charter
http://www.sei.cmu.edu/cmm/cmmi/sg.charter.html

The Architect

Software Architecture Evaluation in the DoD Systems Acquisition Context

Lawrence G. Jones, Rick Kazman

Many modern defense systems rely heavily on software to achieve system functionality. Because software architecture is a major determinant of software quality, it follows that software architecture is critical to the quality of any software-intensive system. For a DoD acquisition organization, the ability to evaluate software architectures before these are realized as finished systems can substantially reduce the risk that the delivered systems will not meet their quality goals. This column presents the basic principles of applying a software architecture evaluation in the DoD system acquisition context.

What Is Software Architecture?

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.* [Bass 98]

It is important to understand that there is no such thing as the architecture of a system—that is, there is no single artifact that one can definitively point to as the architecture. There are, however, many relevant and important views of an architecture depending on the stakeholders and the system properties that are of interest. If we consider the analogy of the architecture of a building, various stakeholders such as the construction engineer, the plumber, and the electrician all have an interest in how the building is to be constructed. Although they are interested in different components and different relationships, each of their views is equally valid and is necessary to ensure they will function properly together. Thus, all views are necessary to fully represent and to fully analyze the architecture of the building. The analogy holds for a software architecture, but in this case the stakeholders might include the development organization, the end user, the system maintainer, the operator, and the acquisition organization. Each of these stakeholders has an important interest in different system properties. We will elaborate on the importance of software architecture to the delivered system and these various stakeholders next.

Why Is Software Architecture Important?

The point is often made that the DoD buys systems not software, so why should the DoD concern itself with software architectures? Simply stated, almost all modern systems, including modern defense systems, rely heavily on software to achieve critical functionality. Thus, many important system quality goals—security, availability, modifiability, performance, and so forth—are achieved through software. The software architecture is a major determinant of software quality and thus of system quality. So, even though we are buying a *system*, the software and in particular the software architecture are of paramount importance in determining whether we get the level of system qualities required. These interrelations, hips are depicted in Figure 1 (adapted from [Fisher 98]).

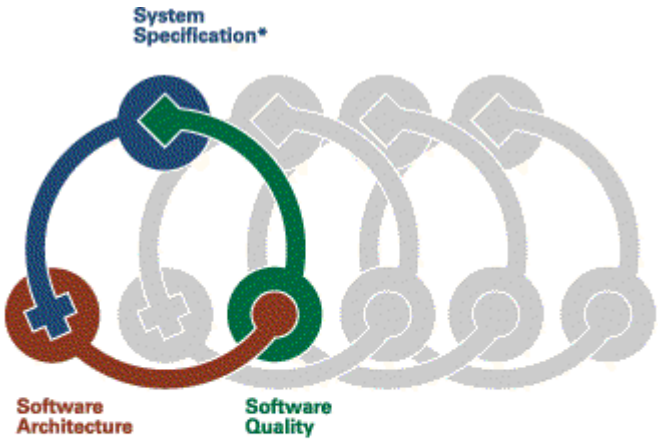

Figure 1: The Relationships Among System Quality Requirements and Software Architectures

It is also important to understand that architectures allow or preclude nearly all of the quality attributes of large, complex systems. For example, if your system has stringent performance requirements, then you must pay attention to things such as component interactions, communication mechanisms, scheduling policies, and component deadlines. If you have modifiability goals for your system, then you need to pay attention to encapsulation properties of your components. If reliability is important, then the architecture must provide schemes for redundancy, restart, fail-over. The list of such

system-wide architectural concerns and strategies goes on and on. All of these approaches to achieving system quality are architectural in nature, having to do with the decomposition of the total system into parts and the ways in which those parts communicate and cooperate with each other. While a "good" architecture cannot guarantee a successful implementation (i.e., an implementation that meets its quality goals), a "bad" architecture can certainly preclude one, as shown in the many case studies in *Software Architecture in Practice* [Bass 98].

Additionally, architectural decisions are among the earliest design decisions made. If an inappropriate architectural choice is made, the consequences are profound. Studies show that the cost to fix an error found during requirements or early design phases are orders of magnitude less than the same error found during deployment or maintenance [Boehm 81]. Thus, it makes economic sense to take steps to ensure the quality of a software architecture. Next we will describe an approach that has proven successful in improving the quality of a software architecture.

Software Architecture Evaluation

The SEI has been developing the Architectural Tradeoff Analysis Method^{SM} (ATAM^{SM}) for the past two years [Kazman 99]. This method not only permits evaluation of specific architecture quality attributes but also allows engineering tradeoffs to be made among possibly conflicting quality goals. The ATAM draws its inspiration and techniques from three areas: the notion of architectural styles, the quality attribute analysis communities, and the Software Architecture Analysis Method (SAAM), which was the predecessor to the ATAM [Kazman 96]. The ATAM is intended to analyze an architecture with respect to its quality attributes, not its functional correctness.

The ATAM involves a wide group of stakeholders including managers, developers, maintainers, testers, reusers, end users, and customers. It is meant to be a risk-mitigation method, a means of detecting areas of potential risk within the architecture of a complex software-intensive system. This focus has several implications, including

- The ATAM can be done early in the software development life cycle.
- It can be done inexpensively and quickly (because it is assessing architectural design artifacts).
- It need not produce detailed analyses of any measurable quality attribute of a system (such as latency or mean time to failure) to be successful, but instead identifies trends where some architectural parameter is correlated with a measurable quality attribute of interest.

What we aim to do in the ATAM, in addition to raising architectural awareness and improving the level of architectural documentation, is to record any risks, sensitivity points, and tradeoff points that we find when analyzing the architecture. Risks are architecturally important decisions that haven't been made (for example, the architecture team hasn't decided what scheduling discipline to use or whether to use a relational or object-oriented database), or decisions that *have* been made but whose consequences are not fully understood (for example, the architecture team has decided to include an operating system portability layer, but is not sure what functions should go into this layer). Sensitivity points are parameters in the architecture to which some measurable quality attribute is highly correlated. For example, it might be determined that overall throughput in the system is highly correlated to the throughput of one particular communication channel, and availability in the system is highly correlated to the reliability of that same communication channel. Finally, a tradeoff point is found in the architecture when a parameter of an architectural construct is host to more than one sensitivity point where the measurable quality attributes are affected differently by changing that parameter. For example, if increasing the speed of the communication channel mentioned above improves throughput but reduces its reliability, then the speed of that channel is a tradeoff point.

To use ATAM effectively in the DoD, it is necessary to understand the special characteristics of the DoD acquisition environment.

The DoD Acquisition Management Process Context

DoD 5000.2R prescribes a high-level acquisition process known as the *DoD Acquisition Management Process*. It serves as the overall roadmap for program execution and includes mandatory acquisition procedures and specific guidance for acquisition programs [Bergey 99]. Although the DoD management process is primarily directed toward major system-acquisition programs, it is intended to serve as a general model for all DoD acquisition programs.

In particular, the contractual process that must be followed is prescribed in the Defense Federal Acquisition Regulation Supplement[1] [DFARS 98]. This contractual process includes three important phases: the pre-award phase, the award phase, and the post-award phase. During the pre-award phase, the acquisition organization prepares and issues a request for proposal (RFP) and interested bidders may respond. During the award phase, source selection occurs. During the source-selection process, the acquisition organization evaluates proposals, obtains best and final offers from bidders, and selects a winning bidder. During the post-award phase, the government administers the contract,

1. The Defense Federal Acquisition Regulation Supplement (DFARS) is the DoD implementation and supplementation of the Federal Acquisition Regulations (FAR).

monitoring the technical progress and performance of the winning bidder. Depending on the scope of the contract, the winning bidder may or may not be responsible for support of the developed system following delivery and deployment.

We will use these phases as a basis for describing different points at which the ATAM might be effectively applied in a DoD or government acquisition.

Applying Architecture Evaluation within the DoD Acquisition-Management Process

Pre-Award and Award Phases for a System-Development Contract

Two major activities that take place in the contractual pre-award and award phases are generation of an RFP and source selection respectively. Release of the RFP defines the official beginning of the solicitation period. After the solicitation formally closes, source selection commences with proposal evaluation and ends with a contract award. Specifying an ATAM-based architecture evaluation can be an effective means of evaluating the technical risks associated with a proposed software architecture. The results can be used as part of the technical evaluation criteria for source selection. The requirement to perform an ATAM-based architecture evaluation must be appropriately integrated into the RFP and the source-selection plan.

Post-Award Contract Administration and Performance Phase for a System-Development Contract

After contract award, an ATAM might be used for (at least) four purposes:

1. Select an architecture among several candidate architectures.
2. Assist in architecture refinement once an architecture has been chosen.
3. Ensure that the chosen architecture is properly documented and communicated.
4. Assist in early evaluation of architectural designs to reduce program risks.

Other ATAM Contractual Applications

The ATAM could also be applied in an acquisition for upgrading an existing system after it has been operationally deployed and is in its post-development/support life-cycle phase. Additionally, an ATAM could also be applied to a legacy system to evaluate and improve how well the architecture would support a set of proposed upgrades. Many

legacy systems have only implicit software architectures: they were either never properly documented or the documentation has not been updated in concert with the system. In such cases, an architecture extraction and reconstruction activity would have to precede the ATAM to redocument the architecture [Kazman 98].

Conclusions

While the ATAM is still in the developmental phase, it has already proven its ability to significantly improve software architectures in several pilot projects in a software development environment. The next challenge is to codify the application of ATAM principles in an acquisition environment. ATAM principles have been effectively applied to a limited extent in source selection, and the initial results are promising. The SEI is collaborating with several acquisition organizations on the use of the ATAM, to help them transition the process into their own organizations and to help them include appropriate language in an RFP to make architectural evaluation an integral part of evaluating proposals. As experience is gained in this area we will continue to share our lessons learned.

About the Authors

Lawrence G. Jones is a senior member of the technical staff in the Product Line Systems Program of the Software Engineering Institute (SEI) of Carnegie Mellon University. In addition to his product line duties, he is also a member of the Capability Maturity Model Integration (CMMI) team. Before joining the SEI, he served in the United States Air Force in a variety of software development, management, and education positions. He served as principal scientist at the Supreme Headquarters for the Allied Powers in Europe (SHAPE) Technical Centre in The Hague, Netherlands. He is also the former chair of the Computer Science Department at the U.S. Air Force Academy. He is active in the computing profession and has membership on the Association for Computing Machinery (ACM) Accreditation Committee, the Institute of Electrical and Electronics Engineers (IEEE), and the Executive Committee of the Computing Sciences Accreditation Commission. He holds a PhD in computer science from Vanderbilt University and master's and bachelor's degrees in industrial engineering from the University of Arkansas.

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 entitled *Software Architecture in Practice*. Kazman received a bachelor's and master's degree from the University of Waterloo, a master's degree from York University, and a PhD from Carnegie Mellon University.

References

[Bass 98] Bass, L., Clements, P., Kazman, R. *Software Architecture in Practice.* Addison Wesley, Reading, Massachusetts, 1998.

[Bergey 99] Bergey, J., Fisher, M., Jones, L. *The DoD Acquisition Environment and Software Product Lines*. CMU/SEI-99-TN-004, May 1999, Pittsburgh, PA.

[Boehm 81] Boehm, B. *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NF, 1981.

[DFARS 98] Defense Federal Acquisition Regulations Supplement, 1998 Edition [online]. August 17, 1998. http://farsite.hill.af.mil/vfdfara.htm.

[Fisher 98] Fisher, M. *Software Architecture Awareness and Training for Software Practitioners; U.S. Army CECOM Course*. June 1998, Pittsburgh, PA.

[Kazman 96] Kazman, R., Abowd, G., Bass, L., Clements, P., "Scenario-Based Analysis of Software Architecture", *IEEE Software*, 13:6, Nov. 1996, 47-55.

[Kazman 98] Kazman, R., Carriere, S. J., "Playing Detective: Reconstructing Software Architecture from Available Evidence", *Automated Software Engineering,* 6:2, April 1999, 107-138.

[Kazman 99] Kazman, R., Barbacci, M., Klein, M., Carriere, S. J., Woods, S. G. "Experience with Performing Architecture Tradeoff Analysis," *Proceedings of the 21st International Conference on Software Engineering (ICSE 21)*, (Los Angeles, CA), May 1999, 54-63.

COTS Spot:

The Elusive Search for Categories

David Carney

One nagging issue about commercial off-the-shelf (COTS) products that surfaces with alarming regularity is category: As people debate whether to choose some software product, they often ask, "Is this product really COTS?" The assumption behind the question is that there exists some collection of attributes that define "real" COTS products, permitting us to categorize any particular commercial product. Implicitly, the goal is to segregate true COTS products from the false (and, presumably, less worthy) ones, and thereby better follow the spate of recent policy directives that affect us all.

If only we could. With few exceptions, however, it is remarkably difficult to determine, with any basis in objective reality, the hard boundary between what is "really a COTS product" and what is not. Even when we can make such a test, it seldom gives us the help we genuinely need.

Toward a Definition of COTS

There are a number of reasons for this unfortunate situation. One reason is that we can't even find a consistently agreed-upon definition for either the "commercial" or the "off-the-shelf" parts of the acronym. So defining both together is a remarkably difficult task. For instance, a standard dictionary definition of "commercial" includes such characteristics as:

> Of or pertaining to commerce ... involved in work intended for the mass market ... designating goods produced in large quantities for use by industry ... having profit as a chief aim …[1]

This is probably sufficient for informal use. But as a guide for making distinctions in government acquisitions, this definition is full of holes. Is it truly sufficient to merely intend that your software product is aimed at a mass market? Is your software thus automatically "commercial"? Or if you produce some software "in large quantities"—surely the easiest of actions with computer software—is this enough to officially "be commercial"?

1. American Heritage Dictionary, 2nd College edition.

If you work for the Department of Defense, you don't get much help from the Federal Acquisition Regulation (FAR), which offers its own definition of "commercial item." The definition is shrouded in Washingtonese, but the key points are similar:

> Property ... sold, leased, or licensed, or offered for sale, lease, or license to the general public ... [or that] will be available [or for which] ... modifications are customarily available in the commercial marketplace or minor modifications are made to meet Federal Government requirements ... [or is] ... a nondevelopmental item developed exclusively at private expense and sold competitively to multiple state and local governments.

Again, the seeker of well-defined categories finds little to go on. "Minor modifications" could include a whole galaxy of changes, and once again there are no specifics about distinguishing the intent to sell something from actually having sold it.

As with "commercial," the same problem appears when we try to pin down what "off-the-shelf" means. Probably no one (no one who regularly reads this column, at least) would contend that, in the context of serious software systems, "off-the-shelf" should connote the way that we use simple computer games (i.e., take off the plastic wrap, pop the disk in, and off you go). But if it's not that simple, users presumably have to take some minor individuating steps to make the software run in their own systems. And if the user has to do at least some diddling, where do we logically stop? As we move down the slippery slope from simple installation through more complex parameterization, through tailoring, through customization, to whatever comes next—when do we look at Toto and realize that we're not in Kansas anymore, and that our software is no longer "off-the-shelf"?

The SEI, in developing a series of COTS-related courses, has provided a definition that seeks to fill some of these holes. It describes a COTS product as one that is:

- sold, leased, or licensed to the general public
- offered by a vendor trying to profit from it
- supported and evolved by the vendor, who retains the intellectual property rights
- available in multiple, identical copies
- used without modification of the internals

This is a distinct improvement, and this definition has proven valuable on many occasions. But it is also true that in many real-world cases, the products that people care about don't quite fit all of these criteria; a lot of things that people are considering for their systems still fall into the grey areas, and do not clearly fit this definition.

So I've come to the conclusion that the attempt to pin down whether a product fits the perfect definition of "COTS" is the wrong question: Even if you can determine absolutely

that a given package is or isn't COTS, that won't tell you what you really need to know. In other words, something that's "really COTS" isn't always better, or cheaper, or faster; there's simply too much context that's needed.

How to Proceed?

I agree that we need some help in this cold, cruel world of commercial software. Like any marketplace, it is filled with pitfalls, and many struggling managers would welcome some guidance in navigating them. But rather than the simplistic acronym approach ("Is it or isn't it COTS?"), I think that we all would be better served by making the effort to find some truly useful characterizations of commercial software. These could aid us both in making choices between competing alternatives, as well as in making the more basic "build vs. buy" decision.

Let's start by turning things upside-down. Let's see if we can characterize some commercial software not by what we'll save from using it, but rather by what it will cost us.

One thing that immediately comes to mind is that commercial software implies process change: It is generally understood that "going commercial" means some kind of business-process reengineering. Conversely, if you absolutely must have an exact match for an existing process, then the commercial route is not for you, and you'd better go custom. Thus, there is some sort of correlation between how much you're willing to change your business process and how much you can accept a commercial software product as-is.

This suggests that we can place the notion of business process—more precisely, the degree to which a business process must change—on a spectrum whose scale is bounded by a custom development at one end and a totally as-is solution on the other. The former end would imply no change at all to my processes, which is true (at least in theory) with custom code written entirely to specification. At the other end, where we take code from some other source, and with no changes, my business processes would undergo a lot of change, maybe even massive process reengineering. This "change-to-process" spectrum would look something like this:

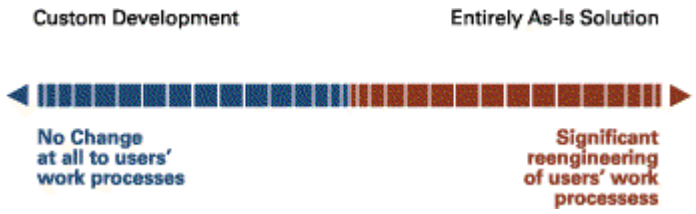

Figure 1: Change-to-Process Spectrum

Clearly, we're being a little simplistic here. On the left side, I'm asserting that a custom development would mean "no change at all," and on the right, I'm sidestepping what "significant reengineering" would really entail. Either of these assertions is arguable. Still, this overall notion—that the amount of process change is related to the amount of custom vs. commercial software—seems reasonable. Rather than worrying about whether some solution is or isn't really a COTS product, it may be of more use to map its potential value (and its potential savings) against the cost that is implied by how much reengineering it will demand. That's a better deciding factor than pedigree.

Let's shift to an orthogonal view of the same question. It's clearly beneficial to know, before we commit to a commercial product, how much we're going to have to tinker with it. So I now posit a spectrum of the changes that we make to a product (either must make or choose to make); its scale is bounded by the degree of change that is possible. Thus, borrowing from my earlier example, we could move from the triviality of a computer game (pop in the disk with no change) through more and more complex code changes: installation, parameterization, tailoring, customization, finally arriving at some sort of gross internal modification. This "change-to-product" spectrum might look like this:

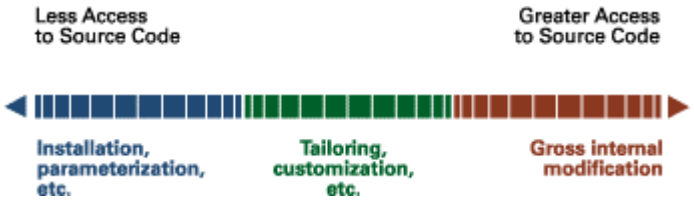

Figure 2: Change-to-Product Spectrum

This spectrum is a bit more ambiguous than the other one. For instance, up to a point, vendors of some software packages expect and encourage changes: Whether you're installing a simple UNIX package or implementing Oracle Financials, there is a certain amount of work that the vendor expects you to do. It's equally true that there are certain things the vendor doesn't want you to do. Locating exactly where that distinction lies is sometimes difficult (though it's a sure thing that when you violate it, you're on the Dark Side). Complicating this is that the scale I suggested ("degree of change that is possible") is based on the visibility of the code and how easily we can access it. But "code" means a lot of different things these days: not only third-generation languages, but fourth-generation languages, schemas, graphical user interface (GUI) languages, and so forth.

Whatever else is true, there are a lot of products commonly considered "COTS" that appear all along this "change-to-product" spectrum. Knowing they're "real COTS" is of

trivial importance. Knowing how much work you have to do to use them is much more valuable.

These spectrums are only two simple possibilities, and I'm not arguing that they are sufficient to make an important decision about choosing some commercial software. I am arguing, though, that a commercial software decision is massively context dependent, and simply giving some product a label, even if one can, doesn't address the problem. What the decision-maker needs is greater insight into the details of his or her particular context and more information about how candidate products fit in that context.

An Example

To realize just how context dependent a commercial decision is, we only have to populate the "change-to-process" spectrum with some real software. Take word processors: Where would we put Microsoft Word? Well, that depends. If our users are switching to automatic document generation for the first time (if, that is, typewriters still existed!), then it would be a huge process change, similar to an earthquake, and far over on the right side. Next, suppose our users currently do word processing, but only with standard generalized markup language (SGML). This change would still be an earthquake, but lower on the Richter scale. Suppose our users are currently using FrameMaker. In that case, while there are many details that change, the essential process is very similar. So even in deciding about Microsoft Word, which is unquestionably a "real COTS solution," we still have different degrees of process reengineering, depending on the current state of the business process.

It might be interesting, at some point in the future, to superimpose these two axes into an X-Y graph, and populate it with other actual examples. But I've probably gone on long enough, so I'll leave that as an exercise for the reader.

Finally, I don't think that there's anything revolutionary in what I've argued for. Everyone knows, deep down, that labels are at best guideposts—"What's in a name?" says the adage. Adages aside, however, and whatever we all may know deep down, we're all still suckers for simple categories, pedigrees, and so forth. It's easier. It's less effort. My argument is only that, with commercial software at least, the adage is true; and fitting a product into the category of **real** COTS seldom does us much good.

So I'll sign off for this issue. Next time, I'll discuss the topic that I promised for this issue—namely, some relationships between commercial software and risk management.

About the Author

David Carney is a member of the technical staff in the Dynamic Systems Program at the SEI. Before coming to the SEI, he was on the staff of the Institute for Defense Analysis in Alexandria, Va., where he worked with the Software Technology for Adaptable, Reliable Systems program and with the NATO Special Working Group on Ada Programming Support Environment. Before that, he was employed at Intermetrics, Inc., where he worked on the Ada Integrated Environment project.

Net Effects

NCC IN Y2K

Scott Tilley

The past two years have seen interest in net-centric computing wax, wane, and then wax again. At the moment, there appears to be a resurgence of interest in the area, driven in part by the proliferation of non-traditional computing devices, by dramatic changes in networking capabilities, and by renewed interest in centralized system administration. In this last column of *Net Effects* for 1999, I take a look at what may be in store for net-centric computing in 2000.

When this column first debuted over 18 months ago, I remarked that it's important to make a distinction between network computers (NC) and net-centric computing (NCC). This remains even truer now than it was then. In part, this is because of the limited adoption of NCs in the marketplace. But NCC itself hasn't gone away; it's just changed to better reflect the current trends in technology.

In the book *Saving Big Blue: Leadership Lessons and Turnaround Tactics of IBM's Lou Gerstner* by Robert Slater (New York, NY: McGraw-Hill, 1999), the topic of IBM's continuing interest in network computing is covered at length. For IBM, the combination of NCC and the growth of the Internet jointly contribute toward what they see as their future: e-business. The emerging e-business economy plays to IBM's traditional strengths of sophisticated computer technology, systems-integration capabilities, and large storage systems. Although sales of NCs by IBM have not been as high as IBM had hoped, the overall vision of a net-centric computing future in 2000 and beyond appears to remain undiminished.

As an aside, Y2K purists know that 2001, not 2000, is the first year of the new millennium. Equating 2000 with the new millennium has become so common that it no longer seems important to point out this discrepancy. Besides, the confusion gives us pundits two opportunities to predict the future, in case we get it wrong the first time. ☺

Information Appliances

Will there come a time when information appliances attached to the Internet will outnumber the typical personal computer (PC)? Many people, including proponents of NCC such as Oracle's Larry Ellison, seem to think so. The past year has seen a proliferation of non-traditional computing devices that are network-aware. Witness the popularity of Palm Computing's Palm VII, which provides wireless Internet access (albeit in selected cities) for mobile professionals. If Sun Microsystem's Jini architecture

for connecting information appliances to a network in a seamless and simple manner is a success, we may indeed see a glut of new non-traditional computing devices in 2000 and beyond.

There seems to be a change in the perception of what a computing device actually should be. Currently most PCs are general-purpose machines that are designed to do many things, although some would argue that they don't do any one of these things very well. For example, my notebook computer comes with fax software—software that I rarely use. I have found it to be difficult to use and buggy, causing odd interactions with my email software. Consequently, I rely on a dedicated "old-fashioned" fax machine. It may not be quite as convenient, but at least it works all the time, and in a predictable manner.

For the average user, the complexity of today's PCs often outweighs their potential advantages. Indeed, it has reached the point where even the venerable *Wall Street Journal* is running articles on the difficulty in using a mainstream PC. Columnist Walter Mossberg predicts that 2000 will signal the beginning of a new era of computing, one in which information appliances will begin to dominate the computing landscape. Rather than being general-purpose machines, information appliances are special-purpose machines that are designed to do one thing, but to do it well. Upgradability by the consumer is limited, but the tradeoff is enhanced usability and stability.

Such information appliances are in fact already here, but we don't always consider them "computers" in the traditional sense. Nor should we. But they are available now and are already in fairly widespread use. For example, Sega's new Dreamcast gaming system is an example of an information appliance that comes with special-purpose hardware and items typically seen only in a PC, such as a modem for dialup Internet connections. It is no accident that WebTV now has over one million users, or that AOL is fast approaching twenty million users. Consumers in general opt for electronics that work, even if they are more restrictive than the typical PC favored by geeks who don't mind tinkering with their machines on a daily basis to keep them running.

Manufacturers of traditional PCs are reacting to the move to simpler computers, and information appliances in general. AOL is readying a TV-based interface to its popular Web service. Oracle is rumored to be working on a Linux-based NC. Even Compaq and Hewlett-Packard are reported to be developing a new "e-PC" that will abandon some of the legacy interfaces found in today's PC (such as the ISA bus or the parallel port) with more limited, but hopefully more reliable, capabilities. Manufacturers are not doing this for purely altruistic reasons of course; they see the adoption of information appliances as a new source of revenue. This is due in part to the need to upgrade (read replace) the devices more quickly than traditional PCs, since the information appliances cannot easily be upgraded.

Ubiquitous Networking

The role of the network in NCC is obvious. One of the more inevitable developments has been the increase in the number of homes with more than one personal computer (or information appliance, as described above). This has caused a new market to be quickly created, replete with products that provide inexpensive and (relatively) easy-to-manage home networks.

The networks leverage the existing infrastructure to provide connectivity throughout the home. There are essentially three types of home networks: telephone line, power line, and wireless. Telephone-line networks require only a standard telephone jack; plugging a cord into the jack and connecting the other end to the computer creates a simple yet acceptable network (when used with the accompanying software). Power-line networking operates similarly, using small adapters that plug into standard wall outlets and then into computers. Although the speed provided by such home networks is not (yet) spectacular, it is sufficient for sharing printers among several PCs, for sharing Internet connections, and for occasional file transfer between devices.

Perhaps the most promising type of maturing technology is wireless networking. If you've ever used the infrared port on your PC to communicate with another similarly enabled device (say, another PC or a printer), you already appreciate the convenience that wireless connectivity provides. By having networks unrestricted by physical connections, computing devices such as NCs can be used in a location-independent manner. Indeed, wireless networks can be used to facilitate the roaming of information appliances from one location to another, in a manner similar to cellular phone calls as they are passed from cell to cell. This type of wireless access is already being made available to users of digital PCS (Personal Communications Services) phones from several vendors, enabling them to surf the Web from their telephones at any time and from anywhere.

Ubiquitous computing has been the focus of research efforts at Xerox PARC and other institutes for some time. Central to the success of ubiquitous computing is the availability of inexpensive and high-speed wireless networking. The emerging IEEE standard 802.11, which directly addresses wireless networking, will surely contribute to the widespread adoption of ubiquitous networking.

Centralized Administration

One of the biggest advantages of personal computers is their "personalization" capability, which permits end users to add their own hardware or software and generally change system configurations without the support or knowledge of the organization's information technology (IT) group. However, this advantage is also viewed as one of the PC's greatest shortcomings as well.

From a support point of view, decentralized computing comes at a very high cost. Many IT veterans yearn for a return to a centralized administration setting, where applications can be managed and deployed in a more cost-effective and timely manner. Until recently, most users were reluctant to adopt this model of computing, since it often came at the cost of removing the computing environment they were familiar with. This is no longer the case, and the move to centralized administration in a net-centric setting is only expected to increase as network bandwidth and powerful servers become more abundant and more economical. Lucent Technologies' recent announcements of an all-optical terabit router is a glimpse at the future of broadband networking, a future where bandwidth would appear to be sufficient to support the most likely usage scenarios of centralized administration of interconnected information appliances.

Given the pervasiveness of the Microsoft Windows operating system, a solution to managing common office software is a prerequisite for the success of centralized administration. Fortunately, both Microsoft and other companies (notably Citrix Systems) have addressed the problem through extensions to NT 4.0 Server (called Terminal Server Edition, or TSE); similar capabilities are built into the forthcoming Windows 2000 product line. Citrix's WinFrame and MetaFrame software imparts relatively low-power information applications with the ability provide users with a complete Windows NT session, but without the NT operating system and the associated application software packages installed on the thin client. This is done by migrating most of the computing to a few centralized hosts, and communicating to the host from the client using a simple, communication-and-display protocol.

WinFrame provides these capabilities for devices running Windows, while MetaFrame provides similar capabilities for non-Windows devices. Given the current popularity of the freely available Linux operating system, the allure of a MetaFrame-like solution is obvious. Since Linux has far less resource requirements (in terms of CPU processing speed, memory, and disk) than a typical Windows installation, existing hardware that would otherwise be made obsolete can be used as thin clients to access servers running Windows NT TSE, in effect providing users with the same Windows experience that they are accustomed to, but at a much lower cost and with a much reduced administrative overhead. It is expected that the complexity of the Windows 2000 product will only add to the adoption of Linux as a thin-client solution for many organizations.

About the Author

Scott Tilley is a visiting scientist with the Software Engineering Institute at Carnegie Mellon University, an assistant professor in the Department of Computer Science at the University of California, Riverside, and principal of S.R. Tilley & Associates, an information-technology consulting boutique. He can be reached at stilley@sei.cmu.edu.

Security Matters

Protecting Critical Systems in Unbounded Networks

Robert J. Ellison, David A. Fisher, Richard C. Linger, Howard F. Lipson, Thomas A. Longstaff, Nancy R. Mead

Society is growing increasingly dependent on large-scale, highly distributed systems that operate in unbounded network environments. Unbounded networks, such as the Internet, have no central administrative control and no unified security policy. Furthermore, the number and nature of the nodes connected to such networks cannot be fully known. Despite the best efforts of security practitioners, no amount of hardening can assure that a system that is connected to an unbounded network will be invulnerable to attack. The discipline of survivability can help ensure that such systems can deliver essential services and maintain essential properties such as integrity, confidentiality, and performance, despite the presence of intrusions.

The New Network Paradigm: Organizational Integration

From their modest beginnings some 20 years ago, computer networks have become a critical element of modern society. These networks not only have global reach; they also affect virtually every aspect of human endeavor. Networked systems are principal enabling agents in business, industry, government, and defense. Major economic sectors, including defense, energy, transportation, telecommunications, manufacturing, financial services, health care, and education, all depend on a vast array of networks operating on local, national, and global scales. This pervasive societal dependence on networks magnifies the consequences of intrusions, accidents, and failures, and amplifies the critical importance of ensuring network survivability.

A new network paradigm is emerging. Networks are being used to achieve radical new levels of organizational integration. This integration obliterates traditional organizational boundaries and integrates local operations into components of comprehensive, network-based business processes. For example, commercial organizations are integrating operations with business units, suppliers, and customers through large-scale networks that enhance communication and services. These networks combine previously fragmented operations into coherent processes open to many organizational participants. This new paradigm represents a shift from bounded networks with central control to unbounded networks.

Unbounded networks are characterized by distributed administrative control without central authority, limited visibility beyond the boundaries of local administration, and a lack of complete information about the entire network. At the same time, organizations' dependence on networks is increasing, and the risks and consequences of intrusions and compromises are amplified.

The Internet is an example of an unbounded environment with many client-server network applications. A public Web server and its clients may exist within many different administrative domains on the Internet. Many business-to-business Web-based e-commerce applications depend on conventions within a specific industry segment for interoperability. Within the Internet, there is little distinction between insiders and outsiders. Everyone who chooses to connect to the Internet is an insider, whether or not they are known to a particular subsystem. This characteristic is the result of the desire, and modern necessity, for connectivity. A company cannot survive in a highly competitive industry without easy and rapid access to its customers, suppliers, and partners. More and more, a company's partners on one project are its competitors on the next, so trust definition and maintenance becomes an extremely complex concept. Trust relationships are continually changing, and in traditional terms may be highly ambiguous. Trust is especially difficult to establish in the presence of unknown users from unknown sources outside one's own administrative control. Legitimate users and attackers are peers in the environment and there is no method to isolate legitimate users from the attackers. In other words, there is no way to bound the environment to legitimate users using only a common administrative policy.

Expanding the Traditional View of Security

The natural escalation of offensive threats versus defensive countermeasures has demonstrated time and again that no practical systems can be built that are invulnerable to attack. Despite the industry's best efforts, there can be no assurance that systems will not be breached. Thus, the traditional view of information-systems security must be expanded to encompass the specification and design of survivability behavior that helps systems survive in spite of attacks. Only then can systems be created that are robust in the presence of attack and are able to survive attacks that cannot be completely repelled.

In short, the nature of contemporary system development dictates that even hardened systems can and will be broken. Survivability solutions should be incorporated into both new and existing systems to help them avoid the potentially devastating effects of compromise and failure as a result of attack.

The Definition of Survivability

We define s*urvivability* as the capability of a system to fulfill its mission, in a timely manner, in the presence of attacks, failures, or accidents. The term *system* is used in the broadest possible sense, to include networks and large-scale systems of systems. In particular, the focus of survivability is on unbounded networked systems where traditional security precautions are inadequate.

The term *mission* refers to a set of very high-level requirements or goals. Missions are not limited to military settings; any successful organization or project must have a vision of its objectives, whether they are expressed implicitly or as a formal mission statement. Judgments as to whether or not a mission has been fulfilled are typically made in the context of external conditions that may affect the achievement of that mission's goals. For example, assume that a financial system shuts down for 12 hours during a period of widespread power outages caused by a hurricane. If the system preserves the integrity and confidentiality of its data and resumes its essential services after the period of environmental stress is over, the system can reasonably be judged to have fulfilled its mission. However, if the same system shuts down unexpectedly for 12 hours under normal conditions (or under relatively minor environmental stress) and deprives its users of essential financial services, the system can reasonably be judged to have failed its mission, even if data integrity and confidentiality are preserved.

It is important to recognize that it is the mission fulfillment that must survive, not any particular subsystem or system component. Central to the notion of survivability is the capability of a system to fulfill its mission, even if significant portions of the system are damaged or destroyed. S*urvivable system* is often used as a shorthand term for a system with the capability to fulfill a specified mission in the face of attacks, failures, or accidents. Again, it is the mission, not a particular portion of a system, that must survive.

Characteristics of Survivable Systems

As noted, e*ssential services* are defined as the functions of a system that must be maintained when the environment is hostile, or when failures or accidents occur that threaten the system.

Central to the delivery of essential services is the capability of a system to maintain essential properties (i.e., specified levels of integrity, confidentiality, performance, and other quality attributes). Thus, it is important to define minimum levels of quality attributes that must be associated with essential services. For example, a launch of a missile by a defensive system cannot be effective if the system's performance is slowed to the point that the target is out of range before the system can launch.

The capability to deliver essential services (and maintain the associated essential properties) must be sustained even if a significant portion of the system is incapacitated. Furthermore, this capability should not be dependent on the survival of a specific information resource, computation, or communication link. In a military setting, *essential services* might be those required to maintain an overwhelming technical superiority, and *essential properties* may include integrity, confidentiality, and a level of performance sufficient to deliver results in less than one decision cycle of the enemy. In the public sector, a survivable financial system is one that maintains the integrity, confidentiality, and availability of essential information and financial services, even if particular nodes or communication links are incapacitated because of an intrusion or accident, and that recovers compromised information and services in a timely manner. The financial system's survivability might be judged by using a composite measure of the disruption of stock trades or bank transactions (i.e., a measure of the disruption of essential services).

Key to the concept of survivability, then, is the identification of essential services, and the essential properties that support them, within an operational system. There are typically many services that can be temporarily suspended while a system deals with an attack or other extraordinary environmental condition. Such a suspension can help isolate areas that have been affected by an intrusion and can free system resources to deal with the intrusion's effects. The overall function of a system should adapt to preserve essential services.

The capability of a survivable system to fulfill its mission in a timely manner is thus linked to its ability to deliver essential services in the presence of an attack, accident, or failure. Ultimately, mission fulfillment must survive, not any portion or component of the system. If an *essential service* is lost, it could in some cases be replaced by another service that supports mission fulfillment in a different but equivalent way. However, we still believe that the identification and protection of essential services is an important part of a practical approach to building and analyzing survivable systems. As a result, we define *essential services* to include alternate sets of essential services (perhaps mutually exclusive) that need not be simultaneously available. For example, a set of essential services to support power delivery may include both the distribution of electricity and the operation of a natural gas pipeline.

Developing Survivability Solutions

Survivability solutions are best understood as risk-management strategies that first depend on an intimate knowledge of the mission being protected. The mission focus expands survivability solutions beyond purely independent ("one size fits all") technical solutions, even if those technical solutions are broad-based and extend beyond traditional computer security to include fault tolerance, reliability, usability, and so forth. Risk-mitigation strategies first and foremost must be created in the context of a mission's

requirements (prioritized sets of normal and stress requirements), and must be based on "what-if" analyses of survival scenarios. Only then can we look toward generic software engineering solutions based on computer security, software quality attribute analyses, or other strictly technical approaches to support the risk-mitigation strategies.

Hence, survivability depends not only on the selective use of traditional computer-security solutions, but also on the development of effective risk-mitigation strategies that are based on scenario-driven "what-if" analyses and contingency planning. "Survival scenarios" positing a wide range of cyber-attacks, accidents, and failures aid in the analyses and contingency planning. However, to reduce the combinatorics inherent in creating representative sets of survival scenarios, these scenarios focus on adverse effects rather than causes. Effects are also of more immediate situational importance than causes, because an organization will likely have to deal with (and survive!) an adverse effect long before a determination is made as to whether the cause was an attack, an accident, or a failure. Awaiting the outcome of a detailed post-mortem to determine the cause, before acting to mitigate the effect, is out of the question when an organization is dealing with the survival of most modern, mission-critical applications.

Developments at the CERT Coordination Center®

The CERT Coordination Center® (CERT/CC) is developing a survivable network analysis (SNA) method to evaluate the survivability of systems in the context of attack scenarios. Also under development is a survivable systems simulator that will support analysis, testing, and evaluation of survivability solutions in unbounded networks.

The SNA method permits assessment of survivability strategies at the architecture level. Steps in the SNA method include

1. system mission and architecture definition
2. identification of essential services and corresponding essential architecture components
3. generation of intrusion scenarios and corresponding compromisable architecture components
4. survivability analysis of architectural softspots that are both essential and compromisable

Intrusion scenarios play a key role in the method. SNA results are summarized in a survivability map that links recommended survivability strategies for resistance, recognition, and recovery to the system architecture and requirements. Results of applying the SNA method to a subsystem of a large-scale, distributed healthcare system

have been summarized. Future studies will involve the application of the SNA method to proposed and existing distributed systems for government, defense, and commercial organizations.

The survivable systems simulator is based on a new methodology called "emergent algorithms." Emergent algorithms produce global effects through cooperative local actions distributed throughout a system. These global effects (which "emerge" from local actions) can support system survivability by allowing a system to fulfill its mission, even though the individual nodes of the system are not survivable. Emergent algorithms can provide solutions to survivability problems that cannot be achieved by conventional means. The survivable systems simulator will allow stakeholders to visualize the effects of specific cyber-attacks, accidents, and failures on a given system or infrastructure. The goal is to enable "what-if" analyses and contingency planning based on simulated walkthroughs of survivability scenarios.

For Additional Information

This column is based on the following publications, which contain additional information about this topic.

R. J. Ellison, D. A. Fisher, R.C. Linger, H. F. Lipson, T. A. Longstaff, N. R. Mead, "Survivability: Protecting Your Critical Systems," *IEEE Internet Computing*, November/December 1999.

H. F. Lipson and D. A. Fisher, "Survivability—A New Technical and Business Perspective on Security," *Proceedings of the 1999 New Security Paradigms Workshop*, September 21-24, Association for Computing Machinery, 1999.

About the Authors

Robert J. Ellison is a member of the technical staff in the Networked Systems Survivability Program at the SEI. He is currently involved in the study of survivable systems architectures. He has previously lead SEI efforts in software development environments and CASE tools. He has a PhD in Mathematics from Purdue University.

David A. Fisher is currently leading a research effort in new approaches for survivability and security in information-based infrastructures at the SEI's CERT® Coordination Center. From 1973-75, Fisher served as program manager in the Advanced Technology Program (ATP) at the National Institute of Science and Technology (NIST), where he developed and managed a major initiative in component-based software and began an

initiative in learning technology. Fisher has more than 60 publications in the areas of information survivability, algorithms, component-based software, programming languages, compiler construction, and entrepreneurial development in the software industry. He earned a PhD in computer science at Carnegie Mellon University, an MSE from Moore School of Electrical Engineering at the University of Pennsylvania, and a BS in mathematics from Carnegie Institute of Technology.

Richard C. Linger is a senior member of the technical staff in the Networked Systems Survivability Program at the SEI, where he is developing methods for analysis and design of survivability for large-scale infrastructure systems. Before joining the SEI, he was a senior technical staff member in IBM, where he co-developed cleanroom software engineering technology for development of ultra-reliable software systems. He has extensive experience in software project management; software specification, design, verification, and certification; and process improvement and technology transfer. His research interests include methods for large-scale system specification and verification, and development and certification of high-reliability software. He has published 3 software engineering textbooks and more than 60 papers, and is an adjunct professor at the Carnegie Mellon Heinz School of Public Policy and Management and the Carnegie Mellon School of Computer Science.

Howard F. Lipson has been a computer security researcher at the SEI's CERT Coordination Center for more than seven years. He has played a major role in extending security research at the SEI into the new realm of survivability. His research interests include the design and analysis of survivable systems, survivable systems simulation, and critical infrastructure protection. Lipson has been a chair of two IEEE-sponsored workshops on survivability. Prior to joining the SEI, he was a research consultant in systems design, helping to manage the complexity and improve the usability of leading-edge software systems. Earlier, Lipson was a computer scientist at AT&T Bell Labs, where he did exploratory development work on programming environments, executive information systems, and integrated network management tools. He holds a PhD in computer science from Columbia University.

Thomas A. Longstaff is currently leading research in network security at the SEI. As a member of the CERT® Coordination Center, he has been investigating topics related to information survivability and critical national infrastructure protection. Publication areas include an overview of information survivability, survivability requirements, survivability in tradeoff analysis, and coming attractions in information survivability. Before coming to the SEI, he was the technical director at the Computer Incident Advisory Capability (CIAC) at Lawrence Livermore National Laboratory in Livermore, California. He completed a PhD at the University of California, Davis in software environments. He received a BA in physics and mathematics from Boston University, and an MS in computer science from the University of California, Davis.

Nancy R. Mead is currently leading a research effort in Survivable Network Architectures at the SEI, and is an adjunct professor in the Master of Software Engineering, Carnegie Mellon University. She is involved in the study of survivable systems requirements and architectures and the development of professional infrastructure for software engineers. Her research interests are in the areas of software requirements engineering, software architectures, software metrics, and real-time systems. Before joining the SEI, Mead was a senior technical staff member at IBM Federal Systems, where she spent most of her career in development and management of large real-time systems. She has a BA and MS in Mathematics from New York University and a PhD in Mathematics from Polytechnic Institute of New York.

Watts New?

Making the Strategic Case for Process Improvement

Watts S. Humphrey

In my previous column, I wrote about management support for process improvement and how your approach should change depending on the manager you are dealing with. The questions left open were how to make the strategic case for improvement, how to make the tactical case for improvement, and how to move from a tactically based to a strategically based improvement program.

In this issue, I describe how to make the strategic case for process improvement. I start on the assumption that you can get the ear of a senior manager. You may work directly for this manager, or you may have obtained an appointment to make a presentation on the subject. In any event, you now have an appointment to see a senior manager. How do your prepare and what do you say?

The General Improvement Case

The approach to follow for almost any type of improvement effort would be much the same:

- Clearly define what you propose.
- Understand today's business environment.
- Identify the executive's current hot buttons.
- Make an initial sanity check.
- Start the plan with two or three prototypes.
- Estimate the one-time introduction costs.
- Determine the likely continuing costs.
- Document the available experience data.
- Estimate the expected savings.
- Decide how to measure the actual benefits.
- Determine the improvement's likely impact on the executive's current key concerns.
- Identify any other ways that the proposed improvement could benefit the business.

- Produce a presentation to give this story clearly and concisely.

Defining the Proposal

Before you do anything, define exactly what you want the executive to do. The best guide that I have found is to actually prepare an implementation letter for the executive's signature. Then in the meeting, if he or she says, "Great, let's do it," pull out the letter and hand it over as a proposed implementation instruction. While this reaction is not likely, the exercise will help you to produce a clear statement of what you intend to propose. Also, if you are several management levels removed from this executive, you should describe the letter as a proposed draft instruction that you have not yet reviewed with your immediate management. Better yet, show the draft letter to your manager first and get his or her suggestions on improving it.

Understand Today's Business Environment

In preparing for the presentation, remember that there is no magic formula for convincing senior managers. Every case is different. The approach must vary depending on the situation and the executive's current priorities. If, for example, this executive has just cut resources to meet a profit goal or the organization has just lost a major contract, this might not be a good time to propose an additional expense. So, plan your improvement strategy with a clear appreciation of what is happening right now in the business.

Identify the Hot Buttons

Next, find out what this executive is most concerned about. Since most executives give lots of talks and issue many statements and announcements, this is generally fairly easy to do. With few exceptions, executives use every available occasion to plug the topics they feel are highest priority. So get copies of some of this executive's recent announcements and presentations, and look at the common themes. You will usually see a fairly consistent message. The manager may frequently mention profitability, or market growth, or development cycle time. Because executives are concerned with many things, he or she will almost certainly make many points. But if there is an overriding concern, much like a television commercial, this topic will pop up every time there is an opportunity. Once you know the executive's current hot button, figure out how the process improvement you propose would address that concern, then make sure the improvement justification addresses this topic.

Make an Improvement Sanity Check

In preparing an executive proposal, the first step is to gather the known facts about the costs and benefits of the proposed improvement program. As soon as you have the data, make an initial sanity check: Does the proposed process improvement directly address the executive's key concerns? If not, are the cost savings significant enough to justify the executive's listening to the proposal? If the improvement directly addresses something the executive has been pushing for, then cost will not be a key concern. If cost savings are important, however, are the proposed savings large enough to be convincing?

Most executives know that improvements are rarely as effective as first proposed and that there are always hidden costs. A good rule of thumb is that improvements with savings of 2 or more times are usually impressive while numbers below 25% are likely to be ignored or subjected to very close scrutiny. If cost is important and you are not proposing a significant cost saving, consider putting off the presentation until you can make a stronger case.

Prototype Introduction

If the proposed improvement passes this sanity check, the next step is to analyze the costs of introduction. It is almost always a good idea to start an improvement program with one or more prototype tests. This not only reduces the initial introduction costs, it also maximizes your chances of success. Just about any change will affect both engineer and management behavior, and these changes are rarely natural or easy. Thus, many people will likely have initial problems following the new methods. To be successful, you must identify and resolve these problems at the very beginning. The longer it takes people to properly use the new methods, the more the introduction will cost and the longer it will take to show benefits. The principal advantages of starting with a prototype program are that the initial costs are lower and it is easier to watch a few limited pilot programs to make sure they are getting the needed support and assistance.

One major risk in any improvement program is that the prototype project could be cancelled or redirected. To protect your project from this risk, try to get two or three trial projects underway. That way, if one is killed or redirected, you will still have the others to fall back on.

Introduction Costs

While you will almost certainly follow a gradual introduction strategy, it is a good idea to show both the prototype and the total introduction costs. The reason is that the introduction strategy will probably change several times before you are done and you

don’t want to keep changing the cost–benefit story. Emphasize that you are presenting the total introduction costs for the entire organization, but that the initial costs for the prototype program will be much lower.

In any significant improvement, there will be initial introduction costs as well as continuing costs for sustaining the improvement. Since any process-improvement introduction will require some executive and management time, you need to make an appropriate allowance. Generally, however, the major costs will be the time to train and support the engineers. Even the introduction of a new tool takes training and support, so don’t gloss over the introduction costs; they can amount to very big bucks.

For example, with a new programming language, a minimum of two weeks of intensive training is usually required, often followed by a period of close consultation during initial use. Similarly, a new tool will require an initial training session of several days plus guided practice sessions and continuing professional support for at least a few weeks.

In estimating these costs, remember one key guideline: Your story will be judged by its weakest point. If someone finds an error or a serious underestimate anywhere in the story, the assumption will be that similar errors infect the entire story. So be careful about making low estimates or assuming that some costs are insignificant. If you don’t know the facts, find someone who does. Above all, don’t make unsupported assumptions; your entire presentation could be discredited.

In addition to executive, manager, and engineering time, trainers and expert assistance will almost invariably be needed. This can add a significant cost, particularly if you plan to use outside assistance. On the other hand, the costs of building internal experts and trainers can be very large, and few executives will want to make such a significant commitment, at least until the proposed improvement has been proven with early tests.

The Continuing Costs

After the improvement has been introduced, there will be ongoing support costs. You may need continuing training to cover engineering turnover or staff growth. Expert assistance and support may also be needed. These costs can be substantial, so it is important to identify them. Describe them clearly up front and then justify them. If you don’t give a complete cost story, management will sense that there are hidden costs and likely assume that these costs are much greater than they actually are.

The Process-Improvement Benefits

Next, we turn to the benefits. Here, you must address two points: first, how long will it take for the improvement program to recover the introduction costs, and second, how will the improvement address the executive's principal concerns? If you can show that the improvement will pay for itself, then the other benefits would be pure gravy. So start by making the cost case.

The way to make the cost case is to first gather the available facts on improvement benefits. Here, you are at a disadvantage. Costs are always easier to prove than savings. Executives know this, however, probably better than you do. After all, they spend much of their time justifying changes. So don't worry about proving an ironclad case; executives will rarely demand it. But they will want a logical story that hangs together and looks complete and realistic.

Improvement Experience

So, first, what are the available facts? Unfortunately, there are few statistically sound improvement studies, even for accepted process-improvement methods. While there may be some available analyses, you will probably have to rely on anecdotal evidence. This may not be as precise as a comprehensive statistical study, but such evidence can be even more convincing. The best case would be one in which someone in your industry has implemented the same improvement and described its benefits in a talk or a paper. If you can find a suitable example, summarize the general findings in the executive presentation, but then emphasize the results reported by your competition.

Calculating the Savings

There are many ways to save money. In the final analysis, however, most software cost savings result from personnel reductions. For example

- By introducing a design inspection program, you can eliminate defects early in the process and save considerable rework.
- A measured quality program can reduce the numbers of defects found in test and shorten testing time.
- A configuration-management system can save development time by ensuring that correct program versions are always available.
- A change-control system can reduce the number of uncontrolled changes and save development time.

While these savings are all real, they all have the disadvantage of being very hard to prove, either in advance or after the fact. As a result, the most convincing argument is generally that the XYZ Corporation cut their test time by x%, or that the ABC Company reduced customer-reported defects by y%. Starting from these numbers, you can then generally show the amount of money you would save if your organization had similar results.

Measuring the Benefits

In concluding the presentation, discuss how the prototypes will be designed to measure the improvement benefits. For example, if the proposed improvement is intended to reduce development cycle time, discuss how to demonstrate that it does. A common problem, however, is that few organizations have data on their current operations. Thus, even if you conduct a highly successful prototype experiment, you may have no way to show that it was successful. That is, you will have lots of "after" data but no "before" data with which to compare it. As part of the proposal, raise this issue and suggest ways to handle it.

Even when organizations have little or no data on their current operations, there are usually a few things that you can measure. For example, data are often available on the length of time by which projects have missed their planned delivery dates. There are also often records of the numbers of defects found in system test or reported by customers. Similarly, data can generally be found to calculate the percentage of the development schedule that is spent in integration and system test. Another good measure is the total development hours divided by the total lines of delivered source code. While no single measure can characterize the quality of an organization's processes, there are many possible measures that can be obtained from most accounting and project-reporting systems.

Because you need to apply these measures to the existing projects, it is important to start looking around for available data even before you make the proposal. Then you can use these data in justifying the proposed improvement. Also, you can be reasonably sure that there will be a way to measure the benefits when you are done.

Other Benefits

While cost savings are important, not all improvements can or should be cost justified. For example, if you can show that the change will improve schedule accuracy and predictability, reduce cycle time, or make your organization more competitive, management will often approve the proposal, even if it does not clearly save money. The

key is to convince management that the improvement is good for the business and then, if possible, show that it will also pay for itself. If you cannot prove the savings story, however, don't give up. If the other benefits are compelling, management may be willing to proceed anyway.

Stay Tuned

In the next issue, I will use an example to show how to structure and give an executive presentation on process improvement. Following that, subsequent columns will deal with how to make the tactical case for improvements and then how to move from a tactically to a strategically based improvement program.

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 Dan Burton, Marsha Pomeroy-Huff, Jim McHale, Mark Paulk, and Bill Peterson.

In Closing, an Invitation to Readers

In these columns, I write about software issues and the impact of quality and process on engineers and their organizations. I am, however, most interested in addressing 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 I plan future columns.

Thanks for your attention, and please stay tuned.

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 six books. His most recent books are *Managing the Software Process* (1989), *A Discipline for Software Engineering* (1995), *Managing Technical People* (1996), and *Introduction to the Personal Software Process*^{SM}(1997). 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.