Carnegie Mellon
Software Engineering Institute

news @ sei
interactive

Volume 6 | Number 1 | First Quarter 2003

In This Issue

columns

features

http://www.interactive.sei.cmu.edu

news @ sei interactive

From the Director

At this year's Software Engineering Process Group (SEPG℠) Conference, held in Boston on February 24- 27, 1,550 attendees came together to share their insights and experiences. Compared with previous SEPG conferences, SEPG 2003 included increased numbers of presentations by users and adopters of Capability Maturity Model® Integration (CMMI®) and Team Software Process℠ (TSP℠) from the United States, Europe, and Asia, evidence of the growing impact of these methodologies in the global community of software engineers.

On display in the exhibit hall at SEPG 2003 was a new book in the SEI℠ Series in Software Engineering, *CMMI: Guidelines for Process Integration and Product Improvement*. As described in the book, CMMI best practices are viewed in two different representations, staged and continuous. Those with continuous experience say this newer and less-documented path to software process improvement is transforming software businesses and yielding significant results. To learn more, read "Taking the Road Less Traveled: The CMMI Continuous Approach."

As part of their efforts to improve their development and maintenance practices, many organizations pin their hopes for improving software systems on commercial off-the-shelf (COTS) software products. Although the use of COTS products can be problematic, Lisa Brownsword and Ed Morris of the SEI provide examples of successful adoptions of COTS-based systems in "The Good News About COTS."

As network connectivity becomes an essential feature of more and more software-intensive products and systems, cyber security is an increasingly important requirement. The SEI has developed a security risk evaluation methodology for organizations, called the Operationally Critical Threat, Asset, and Vulnerability Evaluation℠ (OCTAVE℠) method. As with Capability Maturity Models and other tools, techniques, and methods, the SEI has begun to support the development of a worldwide community of the users of OCTAVE. Among the many benefits of such communities of practice is that they help to inform ongoing development and evolution. To learn more, see "OCTAVE Users' Forum: Helping to Build a Community of Practice."

Increased demands for cyber security, as well as the need to integrate COTS components into systems, require an increasingly sophisticated set of skills for the acquirers of software-intensive systems. To help acquirers of systems identify and characterize the complexity associated with acquiring today's systems, the SEI has initiated an Acquisition Support Program (ASP). You can learn more about the ASP in the article.

We hope you find these articles useful and informative, and we look forward to seeing you next year in Orlando for SEPG 2004!

Stephen E. Cross
SEI Director and CEO

The Architect

Defining the Terms Architecture, Design, and Implementation

Rick Kazman and Amnon Eden

Introduction

Over the past 10 years many practitioners and researchers have sought to define software architecture. At the SEI, we use the following definition:

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

Definitions of software architecture abound. We've been collecting definitions from visitors to our Web site (http://www.sei.cmu.edu/architecture/definitions.html) and already have received dozens. However, we are interested not only in understanding the term "software architecture" but in clarifying the difference between architecture and other related terms such as "design" and "implementation." The lack of a clear distinction among these terms is the cause of much muddy thinking, imprecise communication, and wasted, overlapping effort. For example, "architecture" is often used as a mere synonym for "design" (sometimes preceded with the adjective "high-level"). And many people use the term "architectural patterns" as a synonym for "design patterns."

Confusion also stems from the use of the same specification language for both architectural and design specifications. For example, UML is often used as an architectural description language. In fact, UML has become the industry de facto standard for describing architectures, although it was specifically designed to manifest detailed design decisions (and this is still its most common use). This merely contributes to the confusion, since a designer using UML has no way (within UML) of distinguishing architectural information from other types of information.

Confusion also exists with respect to the artifacts of design and implementation. UML class diagrams, for instance, are a prototypical artifact of the design phase. Nonetheless, class diagrams may accumulate enough detail to allow code generation of very detailed programs, an approach that is promoted by CASE tools such as Rational Rose and System Architect. Using the same specification language further blurs the distinction between artifacts of the design (class diagrams) and artifacts of the implementation

(source code). Having a unified specification language is, in many ways, a good thing. But a user of this unified language is given little help in knowing if a proposed change is "architectural" or not.

Why are we interested in such distinctions? Naturally, a well-defined language improves our understanding of the subject matter. With time, terms that are used interchangeably lose their meaning, resulting inevitably in ambiguous descriptions given by developers, and significant effort is wasted in discussions of the form "by design I mean…and by architecture I mean…"

Seeking to separate architectural design from other design activities, definers of software architecture in the past have stressed the following:

1. "*Architecture* is concerned with the selection of architectural elements, their interaction, and the constraints on those elements and their interactions…*Design* is concerned with the modularization and detailed interfaces of the design elements, their algorithms and procedures, and the data types needed to support the architecture and to satisfy the requirements."
2. Software architecture is "concerned with issues...beyond the algorithms and data structures of the computation."
3. "Architecture…is specifically not about…details of implementations (e.g., algorithms and data structures.)…Architectural design involves a richer collection of abstractions than is typically provided by OOD" (object-oriented design).

In suggesting typical "architectures" and "architectural styles," existing definitions consist of examples and offer anecdotes rather than providing clear and unambiguous notions. In practice, the terms "architecture," "design," and "implementation" appear to connote varying degrees of abstraction in the continuum between complete details ("implementation"), few details ("design"), and the highest form of abstraction ("architecture"). But the amount of detail alone is insufficient to characterize the differences, because architecture and design documents often contain detail that is not explicit in the implementation (e.g., design constraints, standards, performance goals). Thus, we would expect a distinction between these terms to be qualitative and not merely quantitative.

The ontology that we provide below can serve as a reference point for these discussions.

The Intension/Locality Thesis

To elucidate the relationship between architecture, design, and implementation, we distinguish at least two separate interpretations for abstraction in our context:

1. Intensional (vs. extensional) design specifications are "abstract" in the sense that they can be formally characterized by the use of logic variables that range over an unbounded domain. For example, a layered architectural pattern does not restrict the architect to a specific number of layers; it applies equally well to 2 layers or 12 layers.
2. Non-local (vs. local) specifications are "abstract" in the sense that they apply to *all* parts of the system (as opposed to being limited to some part thereof).

Both of these interpretations contribute to the distinction among architecture, design, and implementation, summarized as the "intension/locality thesis":

1. Architectural specifications are intensional and non-local
2. Design specifications are intensional but local
3. Implementation specifications are both extensional and local

Table 1 summarizes these distinctions.

Table 1. The Intension/Locality Thesis

Architecture	*Intensional*	*Non-local*
Design	*Intensional*	*Local*
Implementation	*Extensional*	*Local*

Implications

What are the implications of such definitions? They give us a firm basis for determining what is architectural (and hence crucial for the achievement of a system's quality attribute requirements) and what is not.

Consider the concept of a strictly layered architecture (an architecture in which each layer is allowed to use only the layer immediately below it). How do we know that the architectural style "layered" is really architectural? To answer that we need to answer whether this style is intentional and whether it is local or non-local. First of all, are there an unbounded number of implementations that qualify as layered? Clearly there are. Secondly, is the layered style local or non-local? To answer that, we need only consider a violation of the style, where a layer depends on a layer above it, or several layers below it. Since this would be a violation wherever it occurred, the notion of a layered architecture must be non-local.

What about a design pattern, such as the factory pattern? This is intensional, because there may be an unbounded number of realizations of a factory design pattern within a system. But is it local or non-local? One may use a design pattern in some corner of the system and not use it (or even violate it) in a different portion of the same system. So design patterns are local.

Similarly, it is simple to show that the term "implementation" refers only to artifacts that are extensional and local.

Conclusions

Since the inception of architecture as a distinct field of study, there has been much confusion about what the term "architecture" means. Similarly, the distinction between architecture and other forms of design artifacts has never been clear. The intension/locality thesis provides a foundation for determining the meaning of the terms architecture, design, and implementation that accords not only with intuition but also with best industrial practices. A more formal and complete treatment of this topic can be found in our paper, "Architecture, Design, Implementation." But what are the consequences of precisely knowing the differences among these terms? Is this an exercise in definition for definition's sake? We think not. Among others, these distinctions facilitatedetermining what constitutes a uniform program e.g., a collection of modules that satisfy the same architectural specificationsdetermining what information goes into architecture documents and what goes into design documentsdetermining what to examine and what not to examine in an architectural evaluation or a design walkthroughunderstanding the

distinction between local and non-local rules (i.e., between the design rules that are enforced throughout a project versus those that are of a more limited domain, because the architectural rules define the fabric of the system and how it will meet its quality attribute requirements, and the violation of architectural rules typically has more far-reaching consequences than the violation of a local rule).

Furthermore, in the industrial practice of software architecture, many statements that are said to be "architectural" are in fact local (e.g., *both tasks A and B execute on the same node*, or *task A controls B*). Instead, a truly architectural statement would be, for instance, *for each pair of tasks A,B that satisfy some property* X, *A and B will execute on the same node and the property Control(A,B) holds*.

More generally, for each specification we should be able to determine whether it is a *design* statement, describing a purely local phenomenon (and hence of secondary interest in architectural documentation, discussion, or analysis), or whether it is an instance of an underlying, more general rule. This is a powerful piece of information.

References

[Bass 98] Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice. Reading, MA: Addison Wesley Longman, Inc., 1998.

[Booch 99] Booch, G.; Jacobson, I.; & Rumbaugh, J. The Unified Modeling Language Reference Manual. Reading, MA: Addison-Wesley, 1999.

[Eden 03] Eden, A. & Kazman, R. "Architecture, Design, Implementation." Proceedings of the 25th International Conference on Software Engineering (ICSE 25), Portland, OR, May 2003.

[Garlan 93] Garlan, D. & Shaw, M. "An Introduction to Software Architecture," 1–39. Advances in Software Engineering and Knowledge Engineering, Vol. 2. Edited by V. Ambriola and G. Tortora. New Jersey: World Scientific Publishing Company, 1993.

[Kazman 99] Kazman, R. "A New Approach to Designing and Analyzing Object-Oriented Software Architecture." Invited talk, Conference On Object-Oriented Programming Systems, Languages and Applications (OOPSLA), Nov. 1–5, 1999, Denver, CO.

[Monroe 97] Monroe, R. T.; Kompanek, A.; Melton, R.; & Garlan, D. “Architectural Styles, Design Patterns, and Objects.” IEEE Software 14, 1 (January 1997): 43–52.

[Perry 92] Perry, D. E. & Wolf, A. L. “Foundation for the Study of Software Architecture.” ACM SIGSOFT Software Engineering Notes 17, 4 (1992): 40–52.

[Popkin Software 00] Popkin Software. System Architect 2001. New York, NY: McGraw-Hill, 2000.

[Quatrani 99] Quatrani, T. Visual Modelling with Rational Rose 2000 and UML, Revised. Reading, MA: Addison Wesley Longman, Inc., 1999.

[Schmidt 00] Schmidt, D. C.; Stal, M.; Rohnert, H.; & Buschmann, F. Pattern-Oriented Software Architecture, Vol. 2: Patterns for Concurrent and Networked *Objects*. New York, NY: John Wiley & Sons, Ltd., 2000.

About the Authors

Rick Kazman is a senior member of the technical staff at the SEI, where he is a technical lead in the Architecture Tradeoff Analysis Initiative. He is also an adjunct professor at the Universities of Waterloo and Toronto. His primary research interests within software engineering are software architecture, design tools, and software visualization. He is the author of more than 50 papers and co-author of several books, including a book recently published by Addison-Wesley titled *Software Architecture in Practice*. Kazman received a BA and MMath from the University of Waterloo, an MA from York University, and a PhD from Carnegie Mellon University.

Dr. Eden is a faculty member in the Department of Computer Science at the University of Essex and a research scholar at the Center for Inquiry. His research focuses on formalizing the informal narrative describing software design and architecture. He received a Ph.D. in 2000 from Tel Aviv University, and has substantial industrial experience in object-oriented analysis and design and C++ programming. He has held positions at the Tel Aviv College of Management, Technion—Israel Institute of Technology (Israel), Uppsala University (Sweden), and Concordia University (Canada).

CMMI in Focus

Cruisin' the CMMI Web Site

Mike Phillips

Several recent telephone chats about CMMI have reminded me how difficult it can be to find information when you need it. I have frequently received calls from people who have questions about CMMI and don't know that the information they seek is on the CMMI Web site[1]. This issue's column is dedicated to helping you navigate the CMMI Web pages to take advantage of the wealth of information found there.

If you want to navigate along as you read, you might want to either print the column or open the CMMI Web site in another browser window. To begin at the beginning, we'll first go to the CMMI Main page.

CMMI Main Page

As you scan the CMMI main page[1], notice that there are several elements to help you besides the links to the other CMMI Web pages. The CMMI News section highlights the most recent and significant events related to CMMI. The Working With Us section provides an opportunity for those of you who want to work with the SEI as you adopt CMMI to contact someone who can describe and arrange for SEI transition services. Finally, the CMMI Search section enables you to search the CMMI Web site for information. This facility is a quick way to find specific information without wading through the menus.

CMMI Models Page

One of the primary destinations on the CMMI Web site is the CMMI Models[2] page, where the various models and their representations can be downloaded. Each of the models is available in both PDF and Microsoft Word format. The PDF files are the official versions of CMMI models. These files always print consistently and are locked to ensure that the integrity of the model is preserved. The Word versions are provided so that you can tailor the material to support the internal process improvement efforts in your environment.

---

1 http://www.sei.cmu.edu/cmmi/

2 http://www.sei.cmu.edu/cmmi/models/models.html

Organizations often want to cut and paste elements of the model and its supporting information (the early chapters and the glossary or other appendices) for use in a particular project or functional area. This is possible using the Word format of the models. This format also allows teams to add notes or "organizational amplifications" that are helpful. I've imagined that various global replacements might be made to accommodate those who use non-American English spelling. This is not a "translation" per se but an effort to make readers more comfortable with the material.

"Errata" sheets for each model, in PDF format, are also available from this page. These sheets list the minor errors that are difficult to fully remove before publication. As the new Addison-Wesley book, *CMMI: Guidelines for Process Integration and Product Improvement*[1], was being prepared, we found a few more errata items. These errors are documented in these Web pages to ensure that the most accurate documentation is always available.

CMMI General Information and Background Pages

Other Web destinations include the CMMI General Information[2] page, which has some valuable information for those new to the concept of CMMI. The CMMI Background[3]page allows you to view the *"A" Specification*, which determined the design of CMMI, and the *Concept of Operations*, which describes how inclusion of further disciplines in CMMI models might occur under the guidance of the CMMI Steering Group. However, most of you are reading this because you are either considering or beginning the upgrade to CMMI, so the remainder of this column will concentrate on the CMMI Adoption page[4].

---

1 http://www.sei.cmu.edu/products/publications/process-improvement.htm

2 http://www.sei.cmu.edu/cmmi/general/general.html

3 http://www.sei.cmu.edu/cmmi/background/background.html

4 http://www.sei.cmu.edu/cmmi/adoption/adoption.html

CMMI Adoption Page

The information on this page is most valuable to those of you researching the reasons for adopting CMMI. The contents of this page include pointers to information developed by CMMI users, others interested in CMMI, and the SEI.

Technical Notes

In the previous issue's column, I mentioned some of the technical notes in which authors have helped us explain or interpret the CMMI material from various perspectives. One, for example, describes how a CMMI model might be effectively interpreted in an operational environment. Another elucidated the compatibility between the guiding principles for Earned Value Management and CMMI best practices. A third highlights how a product-line focused organization can apply the guiding principles for product-line practices[1] within a CMMI framework (or vice versa). There are technical notes in most of the categories on the CMMI Adoption page; look for "technical note" and a date following the report title.

Useful Contacts

The Useful Contacts[2] section links to information about events, forums, and other ways to contact those already using CMMI. This section has several links that bear mention. The first is the link to the National Defense Industrial Association, the co-sponsor of our annual CMMI Technology Conference and User Group. This link lets you check the plans for this year's conference and peruse the briefings from the November 2002 conference. If you browse through the 2002 briefings, you'll find much useful information about adopting CMMI and how different organizations approached that task. Other briefings provide practical CMMI adoption advice or focus on appraisals.

One briefing describes the long-term value of having gathered process improvement indicators, as those will help to guide future improvement. Another noted the relatively small additional effort (about 22 staff days) to provide this enhanced picture of the process improvement effort. Another noted the enhanced business case for CMMI over the SW-CMM, because applying process discipline across the rest of the development organization multiplied the reduction of test time and defects.

---

1 http://www.sei.cmu.edu/plp/plp_init.html

2 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#useful

Another important link from the Useful Contacts takes you to a page that lists the organizations that have volunteered to be listed as CMMI Early Adopters, with a point of contact for each. As the CMMI user community grows, you may find that you will want your organization to be listed as well. We often are asked where assistance can be found in addition to the SEI. The SEI Transition Partners link displays listings of instructors and appraisers for the CMMs and CMMI.

Both the Software Engineering Information Repository (SEIR) link and the Yahoo Discussion Groups link allow you to find places to ask questions of people from other active organizations about areas of concern for you. Both require a sign up, but neither have fees associated with membership. Both allow access to documents that have been provided by discussion group members.

As our focus moved forward from development to adoption and upgrade, we began accepting documents from a variety of sources that the CMMI Implementation team determined would be helpful to the leaders of organizations beginning the CMMI journey. The BSCW Shared Workspace site contains a variety of documents and presentations that have proven valuable in initial efforts. For example, one organization found it helpful to provide a Microsoft PowerPoint summary of the required and expected elements of a CMMI model. Another found it better to prepare a Microsoft Word table version that would help before and during a SCAMPI appraisal. Another company performed a study on the potential return on investment of CMMI-based process improvement. Each of these is found on this site, as well as a variety of presentations that may provide useful material for your internal briefings.

One file that we have made available on the BSCW site requires some explanation: the Draft Practice Implementation Indicator Documents (PIIDs). As we sought to move orientation of the SCAMPI appraisal method from discovery to verification, we realized that we needed some form of data capture to organize the information more effectively. The PIIDs are templates that organizations can use to record data pertinent to an appraisal. They are described in the SCAMPI Method Definition Document V1.1 (MDD), and we initially considered providing the documents themselves as part of the MDD. But we have discovered that an example provided in our documents often gets interpreted as a requirement. Therefore the PIIDs are offered separately as a tool that might help organizations gauge their readiness for any progress review such as a full SCAMPI.

Overview of CMMI

The Overview of CMMI[1] section is a good place for those relatively new to CMMI to read detailed information about the CMMI concept and learn how CMMI can benefit their organizations.

CMMI Appraisals

The entries in the CMMI Appraisals[2] section cover appraisal-related information, including a summary of appraisal results, discussion of appraisal methodologies, details of the SEI Appraiser Program, and lists of SEI transition partners.

Adoption and Transition

The Adoption and Transition[3] section contains case study information and guidelines for upgrading from the SW-CMM or EIA 731 to CMMI. A particularly valuable link in this section is the link to the results of "The Road to CMMI" workshop, containing information shared by organizations that have adopted CMMI.

Learning Resources

The Learning Resources section[4] points to training courses offered by the SEI and workshops available to increase your knowledge of CMMI.

Tools and Techniques

The entries in the Tools and Techniques[5] section provide practical tools you can use to ease your adoption of CMMI. A powerful tool that may benefit your organization is the Generic Database Model, which contains CMMI model framework components in Microsoft Access format.

---

1 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#overview

2 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#appraisal

3 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#adoptrans

4 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#learning

5 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#tools

CMMI Comparisons

The CMMI Comparisons[1] section is especially useful for those who want to compare CMMI to other standards, approaches, or improvement models. Comparisons are made between CMMI and the SW-CMM, EIA 731, Earned Value Management, product line practice, LESAT, and Balanced Scorecard.

IPPD

The IPPD[2] section introduces the concept of integrated product and process development, including a bibliography of further reading.

Process Areas

The Process Areas[3] section presents information specific to individual CMMI process areas, including Decision Analysis and Resolution, Measurement and Analysis, Configuration Management, and Risk Management.

We Want to Hear From You

As time goes on, the character of the CMMI Product Suite will build and develop based on what you, the CMMI users, find useful about it. The messages you send us will shape how CMMI can meet organizations' needs and influence the practice of software engineering throughout the world. Please let us know your needs and wants concerning CMMI. We'd like to hear from you. Send email to cmmi-comments@sei.cmu.edu.

---

1 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#comparisons

2 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#ippd

3 http://www.sei.cmu.edu/cmmi/adoption/adoption.html#process-areas

About the Author

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

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

The COTS Spot

Assumption Management

Robert C. Seacord

Introduction

Recently, I attended a tutorial on software evolution by Emeritus Professor Manny Lehman from Middlesex University. While the overall tutorial was very valuable to our work in COTS-based systems and legacy system modernization, Lehman's remark that most of the defects discovered in existing systems were probably caused by invalid or changed assumptions struck a chord. When I was a programmer at the Data Systems Division of IBM, we always included a section called "Assumptions" in our design documentation. Writing this section was always reason for reflection. What were the assumptions upon which the design was based? And what would happen if these assumptions were wrong, or were changed? These design documents were inevitably scrutinized by an inspection team, and the assumptions would be reviewed, discussed, and possibly refined. Although we took some time to consider assumptions at this point in the process, there was no further formal consideration of assumptions through the remainder of the process. This does not mean that assumptions no longer play a role in the construction and test of the software.

Of course, assumptions have a much longer history. Parnas [Parnas71] characterized interfaces as the assumptions that elements could make about each other, and most of his software engineering contributions involve that observation one way or another. Garlan's architectural mismatch paper [Garlan 95] essentially rediscovered this, but nevertheless brought assumptions into the modern vocabulary.

Why Do We Have Assumptions?

For most large information systems the operational domain is essentially unbounded. No matter how many observations or properties are identified and associated with the domain, it is always possible to add more. The software, on the other hand, having human creators, is finite. The software system is, therefore, intrinsically incomplete. The resultant gap between the system and its operational domain is bridged by *assumptions*, explicit and implicit [Lehman 00]. These assumptions fill in the gaps between the system and the documented and validated requirements of the operational domain.

Additionally, the real-world domain and the application itself are always changing. Even supposing that the initial assumption set was valid, individual assumptions will, as time

goes on, become invalid with unpredictable results or, at best, with operation that is not totally satisfactory [Parnas 94].

Requirements and Assumptions

An argument can be made that nothing in a software system should be assumed, and that everything should be stated as requirements. Even if this were the case, it would certainly be true that the *requirements* were *assumed* to be valid.

In reality, requirements are only a top-level statement of need, and the road between requirements and code is paved with assumptions. These assumptions can be validated and verified along the way, but because we must always deal with some degree of ambiguity, our confidence in the validity of the assumptions may vary considerably.

In one way or another, assumptions are reflected in the software. In my experience at IBM, assumptions were tracked, recorded during design, and reviewed by an inspection team. The inspection team could evaluate assumptions in each design to make sure that they were valid and consistent with system-level assumptions. Unfortunately, the assumption management process typically stopped at this point, because no model or infrastructure was in place to support the tracking of assumptions through implementation and test.

As a programmer of many years, I believe that any programmer would agree that assumptions are an inherent part of software implementation. Every time a decision is made—about how to design an interface, how to implement an algorithm, if and how to encapsulate an external dependency—assumptions are made concerning how the software will be used, how it will evolve, and what environments it will operate in. The unfortunate aspect of software implementation today is that these assumptions are seldom if ever recorded, although they are instrumental in determining the form the software product takes. Also, because these assumptions are not recorded, they are seldom communicated or reviewed. As a result, some assumptions may be incompatible with assumptions made elsewhere in the code, or incompatible with design- or system-level assumptions. These incompatibilities may lead to the insertion of defects or, even worse, post-deployment failures. Furthermore, as was already pointed out, it is likely that individual assumptions will become invalid as a result of changes in the operational domain over the life of the system.

When performing change analysis to determine which changes to accept and which to reject, the configuration control board has no way of knowing which assumptions are built into the software. For example, there may be a number of complex modules that assume a particular hardware configuration. A configuration control board may approve a

change, not understanding that this change invalidates these embedded assumptions. The effort to implement the change may result in a major rewrite of significant portions of the system. The least that can be said here is that a lack of assumption management certainly does not lend itself to predictable schedule and costs.

Assumption Management

Is there a feasible solution to the problem of assumption management? I believe that a solution may in fact exist, although it will require some additional infrastructure and a slight culture shift.

From an infrastructure perspective, programmers are unlikely to manage assumptions independent of source code. The failure of programmers to keep design and other documentation consistent with evolving source code is an established and well-known phenomenon. Sun Microsystems has developed a rather ingenious solution to this problem. One of the unique features of Java is that it supports embedded documentation comments, which are used to generate the API documentation. While parsing the source code to create class files (object files), the compiler converts the declarations and doc comments into HTML documentation [Friendly 95]. This is a user-friendly mechanism for programmers to update documentation by updating the structured comments within their source code. This process can be easily performed in manner that is not disruptive to the coding process.

Perhaps even more closely related to assumption management is the programmatic use of assertions. At its most basic form, an assertion is simply a procedure that takes a boolean parameter and reports to the programmer if the boolean is false [Lewis 97]. Assertions are a form of assumption management, where the assumption can be checked at runtime. Exceptions reflect another kind of assumption, which are used in many modern programming languages, including C++, Java, and Eiffel.

Assumption management is a little bit like assertions on steroids. Beyond simple boolean conditions that can be evaluated by a compiler at runtime, assumption management allows programmers to record a vast range of assumptions in structured English. These assumptions can be easily recorded as part of the implementation, and then extracted from the source code using a pre-processor or *aspect-oriented* programming language [Elrad 01]. Recording assumptions in source code alone might prove invaluable, but extracting them into a searchable repository should allow system architects and lead designers to more easily review the assumptions of individual programmers to determine if they are consistent with design and system assumptions. These databases can later be

used by configuration control boards in change analysis to more accurately determine the impact of a proposed change.

Summary

Assumption management entails a slight shift in software development culture in that the assumptions that are being made as part of the development process must also be recorded in source code and other software artifacts. However, the potential benefit resulting from this practice may be tremendous in the earlier identification and elimination of defects, and in improved change analysis for more predictable and cost-effective software evolution.

References

[Lehman 00] Lehman, M. M. & Ramil, J. F. "Software Evolution in the Age of Component Based Software Engineering," 249–255. *IEE Proceedings Software, 2000*, Vol. 147, No. 6, December 2000.

[Elrad 01] Elrad, Tzilla; Filman, Robert E.; & Bader, Atef. "Aspect-Oriented Programming." *Communications of the ACM 44*, 10 (October 2001).

[Friendly 95] Friendly, Lisa. "The Design of Distributed Hyperlinked Programming Documentation." *Proceedings of the International Workshop on Hypermedia Design '95 (IWHD '95).* <ftp://ftp.java.sun.com/docs/javadoc-paper/iwhd.pdf>.

[Garlan 95] Garlan, David; Allen, Robert; Ockerbloom, John, Architectural Mismatch: or Why It's Hard to Build Systems Out of Existing Parts, Proceedings of the International Conference on Software Engineering, Seattle, 1995

[Parnas 71] Parnas, D. "Information Distribution Aspects of Design Methodology." Proceedings 1971 IFIP Congress, North Holland Publishing Company.

[Parnas 94] Parnas, D.L., "Software Aging" in Proceedings of the 16th International Conference on Software Engineering", Sorento Italy, IEEE Press, 279-287, May 16-21/94.

[Seacord 02] Seacord, Robert; Plakosh, Daniel; & Lewis, Grace. *Modernizing Legacy Systems: Software Technologies, Engineering Processes and Business Practices*. New York, NY: Addison-Wesley, 2003.

[Lewis 97] Lewis, Peter N. "Using Assert()." *MacTech Magazine 13*, 12 (1997). <http://www.mactech.com/articles/mactech/Vol.13/13.12/UsingAssert()/>.

[Wallnau 01] Wallnau, Kurt; Hissam, Scott; & Seacord, Robert. *Building Systems from Commercial Components*. New York, NY: Addison-Wesley, 2001.

[Lehman 98] Lehman, M. M. & Belady, L.A. *Program Evolution: Processes of Software Change*. London: Academic Press, 1985.

About the Author

Robert C. Seacord is a senior member of the technical staff at the SEI and currently leads a team researching software sustainment. He is coauthor of two Addison-Wesley SEI Series in Software Engineering books, *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices* and *Building Systems from Commercial Components* as well as more than 40 papers on component-based software engineering, Web-based system design, legacy system modernization, component repositories and search engines, security, and user interface design and development. He has more than 20 years of development experience and was previously a technical staff member at the X Consortium and IBM.

Security Matters

Can You Prove It?

Larry Rogers

Have you ever received a CD-ROM in the mail, either at home, to be used on your home computer, or at the office, to be used on one of your company's systems? How do you know it came from the company shown on the CD and not from someone else? Also, how can you even determine that the CD is from where it claims to be from and that its contents are what was intended?

The answer is that in most cases, you can't. The CD by itself is just a collection of bits whose origin and ordering cannot be easily determined. Given that CD burners and blank CDs are inexpensive and creating CD artwork is straightforward, it seems that someday, somewhere, somebody will use this technique to send a faked CD to someone as part of an attempt to break into their computers and even their network. Maybe it's already happened.

Let's take this scenario a little further. Now that you know that creating fake CDs is possible, what could you do to verify the authenticity of a CD the next time you receive one?

You could visit the Web site of the organization that created the CD to see if the files are also available there as a download. If they are, you could do a bit-for-bit comparison between the files from the web site and the files on the CD. You'd know that the CD is from that organization and its contents are valid if the comparisons match.

Unfortunately, the Web site for most organizations doesn't help you verify the information stored there or on the CD. You are now faced with the dilemma of not installing either the CD or Web-based version and running the risk of being hacked, or installing one of them and hoping that the contents are valid. This is a hard choice to make.

The issues you face here are *authentication* and *integrity*. You need to authenticate the CD's producer and verify the integrity of the CD's contents. Authentication and integrity are two of the fundamental tenetsof computer security.

First, let's consider authentication. How does one entity prove itself to another? With most computer-based applications, a login and password pair provides proof of identity. You've used these often to log in to your computer or to access a network email or merchant account.

We know from experience that the traditional login and password scheme is weak because it can be easily compromised. Even with strong passwords (i.e., passwords created using letters, numbers, and punctuation) electronic eavesdropping—called sniffing—can capture them.

This type of password is called reusable because you use it over and over. The personal identification number that you use at an automated teller machine is another type of reusable password.

An alternative to reusable passwords is one-time passwords. One-time passwords expire after the first time they're used. Even if they're captured, they're no good.

One-time passwords virtually eliminate the eavesdropping problem. However, they add complexity that has a cost. For example, how do you know which one-time password you should use next? Perhaps you carry a list with you and cross off the ones you've used. That's not very safe because a list could be stolen. Also, what happens when you're out of the office and you've just used your last one-time password? How do you get more?

Another alternative is biometrics. According to Webopedia (an online dictionary found at http://www.webopedia.com), biometrics is an authentication technique that relies "on measurable physical characteristics that can be automatically checked. Examples include computer analysis of fingerprints or speech." Though biometrics is in its more formative stages, usage is expected to grow significantly as more products come to the marketplace.

Imagine then that your computer has a fingerprint scanner. You'd place your index finger into the scanner and the computer would then analyze it to determine who you are. Based on your identity, you'd then be authorized to do your work.

Unfortunately, even biometric techniques aren't foolproof. To use a fictional example from Hollywood, in the James Bond film "Never Say Never Again," SPECTRE defeated the retinal scan used to secure the nuclear missiles by implanting a different eye into one of their agents. Someday this fiction may become fact!

What's the answer? The solution you choose means accepting a tradeoff between threats and the level of complexity and cost that you're willing to accept. For some computers, the login and password scheme is appropriate, but for others, even the most sophisticated biometric techniques may not be enough.

Consider this: what would you be willing to accept as the authentication scheme so that you could use your home computer to vote for a United States presidential candidate in the next election? In light of what happened in the 2000 elections, many have been thinking about this problem and have proposed solutions. Would a login and password be good enough? How about using a credit card-sized device and a reader with your computer? When you swipe this card through the reader and provide a passphrase (a sentence, for example), only the correct combination would authenticate your identity and thereby allow you to vote. Are you comfortable with this scheme? What happens when someone loses their card or forgets their passphrase? Is this method any better than simply signing the voting register as most of us do now? These are all good questions without easy answers.

Authentication is a significant issue when it comes to security, computer and otherwise. There's no practical, inexpensive, and hard-to-foil scheme just yet, but research continues and new products are being developed and used. There's certainly a need, and need usually leads to a technological breakthrough. For now, logins that use one-time passwords and biometrics are the best practices.

Next, let's think about integrity. How do you determine that a collection of information - a file for example - is what was intended by its author? The answer here is called a checksum. As defined again in Webopedia, a checksum is "a simple error-detection scheme in which each transmitted message is accompanied by a numerical value based on the number of set bits in the message. The receiving station then applies the same formula to the message and checks to make sure the accompanying numerical value is the same. If not, the receiver can assume that the message has been garbled."

To create a checksum, you'd look at all of the bits in a message and do some mathematics on them. For example, you could add them together. Unfortunately, a scheme that simple isn't very effective in determining if something has changed, because many different combinations of bits can give the same sum when added together.

The point is that not just any old checksum scheme will do. The mathematics used in the checksum computation must be sufficiently complex so that it is virtually impossible to find two files that have the same checksum value. Only with such a scheme can you determine if a file has been changed.

Integrity is another significant security issue, again both computer-related and otherwise. Knowing that information has changed is essential to e-commerce and other businesses that use the Internet to exchange information. Fortunately, there are checksum products available in the marketplace that can detect changes as small as a single bit no matter the size of the information being checked.

So let's go back to the original problem of trusting the CD we got in the mail. One candidate solution to this problem is called digital signatures. Digital signatures contain authentication information so that you know who created the content and integrity information (checksums) so that you can verify that the content hasn't changed. Had the organization that made the CD digitally signed every file on that CD, you would have been able to determine that the CD came from them and that the files contained what they intended. You would have been able to install its contents and sleep well that night!

There are many products available that create and use digital signatures. Some are free and some are commercial. Examples are Pretty Good Privacy (PGP) and Secure/Multipurpose Internet Mail Extensions (S/MIME). While their specifics differ, both use digital signatures to authenticate and verify information.

Digital signatures go a long way toward answering the question, Can you prove it? when verifying the authenticity and integrity of information. With today's technology, the security of tasks such as verifying electronic mail and making purchases through the Internet can be significantly improved. When the technology matures so that additional biometric attributes convey an even higher degree of confidence for authentication purposes, you can expect these technologies to be widely used.

There may come a day when you no longer need a credit card or driver's license to make a purchase or even identify yourself at the airport ticket counter. Your fingerprint and a glance into a retinal scanner may be all it takes to prove who you are!

About the Author

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

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

This and other columns by Larry Rogers, along with extensive information about computer and network security, can be found at <http://www.cert.org>.

Software Product Lines

Components As Products

Linda Northrop and John McGregor

Products in a software product line are software products for internal use or for sale. Common components for the software product line are part of the core asset base. But what if you are in the software component business? Then each product you build *is* a software component. If you have a family of components with many common features but that vary in some distinct and predictable ways, could you have a product line of components? And is this a silly idea? Not at all silly, just confusing. In fact, today a product line of components makes a lot of sense (and for many, a lot of money).

But before we explain, let's first define what we mean by component. Szyperski gives this definition [Szyperski 98]:

> *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.*

A component is therefore opaque, and it is this opaqueness that allows one company to use proprietary information and algorithms from another company without detailed access that would compromise the intellectual property rights of the providing company. The using company must be able to easily compose the component with other components without special environments or tools. Standards for specific purposes, such as the Enterprise Java Beans and CORBA models, provide market strata and architectural constraints that guide and facilitate the component selection process.

Providing individual software components as products is a rapidly growing business, but not a new business. Software libraries for domains such as numerical analysis or graphics have been available for over thirty years. But a number of changes have occurred in the last few years that have reshaped the component marketplace.

An increased focus on defining interfaces and component models that are defined via interfaces has made it simpler to identify candidates for a particular use. These candidate components "almost" fit and the amount of rework required to use them has dramatically decreased. When you couple this opportunity to eliminate some of the in-house development work with the ever-growing demand for software—more complex and more challenging software—using some prefabricated components makes sense. And many have reached that conclusion; the market demand for components is strong. Moreover, the granularity of commercial components has grown from small, low-payoff components such as stacks and queues to components that provide more comprehensive functionality, such as a complete voice recognition engine. These new

larger, more sophisticated components command a healthy price that results in an attractive return on development investment—an incentive for the component developer. In response, electronic marketplaces such as flashline.com have made it possible to offer small individual components for sale at affordable but profitable prices.

Consider a hypothetical (but close to real) example from the domain of telecommunication protocols. Our hypothetical company, Protocols-R-Us, is a company founded by a small group of managers and developers who all worked for the same multinational telecommunications corporation. The company has begun production of "protocol stack" software components, which are used to parse signals coming into a communications device, to do error detection and retry, and to deliver the signal to the platform software.

As a small company, Protocols-R-Us moved carefully to define their market. Their original business plan was narrowly scoped to address only their first product. The company began by supplying an implementation of a communications protocol to both their former employer and one other company. As the first product was nearing release, the managers began to consider other products. Given the depth of technical knowledge of the company founders, they explored products that could be leveraged from the work done so far. The team determined that their market consisted of companies that are developing the broad array of communication devices and add-ons that are being introduced into the consumer market. Each of these devices is capable of receiving a variety of signals, including voice conversations and data transmissions. Each device would actually need to "understand" numerous protocols. Which protocols are useful is information that is changing rapidly, as is the specification of many of the newer protocols.

The company team saw the rapid increase in the creation and release of new protocol standards as a business opportunity. They could capitalize on this business opportunity only if they could design and produce implementations of several protocols that could coexist in a single device and could do so in a way that kept pace with the release of standards updates as protocols were revised and expanded. The company would also have to be able to keep costs low because the "shelf life" of a given implementation is extremely short. Client companies would want each new version of their device to have the latest implementation in order to provide the widest array of services to their clients. Protocols-R-Us set a goal to meet the protocol processing needs of customers by providing high-quality implementations of telecommunications protocols and to do so as quickly and efficiently as possible.

The company's small staff would need to be enlarged, but given current job market conditions, the new hires would not have a depth of experience in either software development or telecommunications. There would have to be a significant increase in productivity among the experienced personnel or the company would miss most of the benefits of the current telecommunications explosion.

How could they succeed? They could take a software product line approach. And their products would be? A *product line of components*.

As the team began planning for the product line, they had to identify the products that would constitute the product line (the product line scope). The initial product was an implementation of the HyperText Transfer Protocol (HTTP) and a HyperText Markup Language (HTML) translator. If they were to build on their existing client base, it made sense to include protocols that are related either to the Transfer Control Protocol/Internet Protocol (TCP/IP) or HTTP. However, these are well-known protocols for which many implementations exist. A new, but somewhat related, area was the Wireless Application Protocol (WAP) used to provide Internet access from a mobile phone. This is an emerging area in which it should be possible to establish a presence. As an initial scope the team identified the following pieces that would be needed for a complete protocol stack from signal reception to screen presentation:

- Wireless Application Environment (WAE)
- Wireless Session Layer (WSP)
- Wireless Transaction Protocol (WTP)
- Wireless Transport Layer Security (WTLS)
- Wireless Datagram Protocol (WDP)

To offer a "complete solution" the company would also have to provide an implementation of the Wireless Markup Language (WML). The protocol stack accompanied by the language would provide support for accessing HTML-based content on the mobile phone.

The team recognized that their most important asset would be the basic architecture from which all of the products are derived. Since the products will be components, they actually had to consider two architectures: the architecture of the systems into which the components are intended to fit, and the architecture for the protocol components—their product line architecture. They knew that the product line architecture for the components must provide a link between the standard organization for communication protocols and the individual components to permit clients, who are familiar with the network model, to easily understand how the Protocols-R-Us components will fit into their architecture.

The architecture would have to be updated to anticipate the evolution of those components with which it must interoperate. The company formed strategic alliances with a number of companies that produced related products. Through these partnerships, Protocols-R-Us hopes to keep up with the latest design changes and receive advance releases. It will test its products with these new releases to identify any unintended interactions. The architecture must also be maintained to remain current with the latest versions of the applicable standards.

And how will the final products be packaged? Each component will be packaged with a test component that contains the test cases used to test the protocol component. The component will also contain technical documentation that includes specification of both the **provides** and **requires** interfaces for the component. Finally, the shrink-wrapped package will also contain sample designs for integrating sets of the components into different configurations that provide support for different types of products.

Protocols-R-Us shows that the basic principles of product line development apply regardless of the nature of the products—even if the products are components. In the case of our component product line, the main changes in practices are the result of the size of the products and the need to manage the requirements with limited effort. The major impact on the architecture development is the need to pay more attention to the architecture of the environment into which the components will be integrated. There is always some integration between an application and its operating environment; however, the interface between an individual component and an application architecture is not as standardized as the OS interface. On the positive side, the interfaces implemented by a single component are smaller and can be more completely specified than the interface of an entire application.

And so you can have a software product line where the components are products. In fact, a product line approach is a promising development paradigm for the component industry for the same reasons it made sense for Protocols-R-Us.

References

[Szyperski 98] Szyperski, C. *Component Software: Beyond Object-Oriented Programming* Harlow, England; Reading, MA: Addison-Wesley, 1998.

About the Authors

Linda Northrop has over 30 years of experience in the software development field as practitioner, manager, consultant, and educator. She is currently director of the Product Line Systems Program http://www.sei.cmu.edu/programs/pls/pl_program.html at the Software Engineering Institute (SEI). Her current publications are in the areas of software product lines, software architecture, and object technology. She is co-author of the book, *Software Product Lines: Practices and Patterns*.

Northrop chaired the first Software Product Line Conference (SPLC1 <http://www.sei.cmu.edu/plp/conf/SPLC.html>) in 2000 and SPLC2 <http://www.sei.cmu.edu/SPLC2/> in 2002. Linda is the OOPSLA Steering Committee Chair and was the OOPSLA 2001 Conference Chair <http://oopsla.acm.org/oopsla2001/>. She is also on the Executive Committee of ACM

SIGPLAN, and is a member of ACM, the IEEE Computer Society, the Computer Sciences Accreditation Commission, and the ACM/IEEE Joint Committee on Software Engineering.

Dr. John D. McGregor is an associate professor of computer science at Clemson University, a Visiting Scientist in the Product Line Systems program at the SEI, and a partner in Luminary Software, a software development consulting firm. Dr. McGregor has conducted research for organizations such as the National Science Foundation, DARPA, IBM and AT&T. Dr. McGregor is co-author of *Object-oriented Software Development: Engineering Software for Reus,* published by Van Nostrand Reinhold. Dr. McGregor is also co-author of *A Practical Guide to Testing Object-Oriented Software,* published by Addison-Wesley. He has published numerous articles on software development focusing on design and quality issues. Dr. McGregor's research interests include software engineering, specifically in the areas of product development, design quality, testing and measurement.

Watts New

Some Programming Principles - Requirements

Watts S. Humphrey

In this and the next several columns, I discuss some principles for programming work. These are principles that, when followed, will consistently produce superior software that meets the needs of customers and businesses. This column concentrates on the principles that are inherent in software work because of the nature of software products and their requirements. These principles also concern the characteristics of the people who use these products. These programming principles are as follows:

- When we program, we transform a poorly understood problem into a precise set of instructions that can be executed by a computer.
- When we think we understand a program's requirements, we are invariably wrong.
- When we do not completely understand a problem, we must research it until we know that we understand it.
- Only when we truly understand a problem can we develop a superior product to address that problem.
- What the users think they want will change as soon as they see what we develop.

The Programming Job

As any experienced programmer knows, it is hard for people to be absolutely and completely precise about anything. However, to produce a usable program, we must specify exactly what the program is to do under every possible circumstance. The difficulty of being precise is brought home to me whenever someone gives me directions.

"Go three blocks, turn left at the gas station, then take the third street on the right."
"But," I interrupt, "precisely where do I start and in what direction should I face?"

For some reason, the questions that I must ask to get precise directions always seem to annoy people. Why should this annoy them? Don't they know that, without precise and complete information, I might get lost and waste a great deal of time?

The answer is, no they do not. In fact, even when they know exactly what they want done, most people are unable to tell you precisely what to do. What is worse, they will even get annoyed when you press them for the required details. This attitude complicates the programmer's job. To work properly, computing systems must be given absolutely precise instructions. Therefore, to write any program, the programmer must reduce that problem to a precise form before it can be executed by a computer. This means that we must somehow persuade one or more knowledgeable users to explain all of the problem's details. In summary, there are four reasons why this is hard.

- First, the users will not know how to be precise about their needs.
- Second, they will get annoyed when we press them to be precise.
- Third, we will often think that we understand the problem before we really do.
- Fourth, the users will often misunderstand or not even know what they need.

Programming is and always will be a precise intellectual activity that must be performed by people. While machines can help us in this work, they can never figure out what we need, so they can never replace us. The problem is that people are error prone and programs are extraordinarily sensitive to errors. The programming challenge we face is to devise processes and methods that will help us to produce precise intellectual products that are essentially defect-free.

Understanding the Problem

The difference between thinking that you understand a problem and truly understanding it is like night and day. It is amazing how often I think that I know something but then find that I really do not. My mother used to explain that being positive was "being wrong at the top of your voice." When it matters, like when writing a program, preparing a talk, or drafting a paper for publication, I try to prove that what I think is true really is true. When I look at data, consult a reference, or talk to an expert, I often find that my initial opinion was either wrong or too simplistic. The real story is invariably much more interesting and often more complex than I had initially realized.

It is easy to settle for half-baked answers. Some years ago, a programming manager told me about a conversation that she overheard between two of her programmers. They were developing a new version of IBM's COBOL compiler and were debating a particular feature. One of them felt that the users would prefer one approach and the other felt that a different format would be more convenient. This manager told me that, even though

neither of them really knew which approach would be best, they ended up agreeing on some middle ground that they both thought would be OK.

On any large product, there are hundreds to thousands of these minor decisions and, when programmers don't really know the proper or most convenient or most efficient answer, they usually guess. Even if the odds of being right are 50-50, they will make hundreds to thousands of incorrect decisions. Then, the odds of their producing a convenient and highly-usable product will be essentially zero.

Researching Problems

There are lots of ways to research problems. For example, I often write about how people develop programs and the costs and benefits of various programming practices. Ever since I started my research work on personal and team programming methods over 13 years ago, I have gathered data about my own and other people's work. I now have a database of over 11,000 programs that I regularly use to verify some opinion or to answer some question. Depending on the question or problem, data can be very helpful.

The second approach is to ask someone who knows the answer. The problem here is that, like the COBOL programmers, you may not have an expert on hand and will often settle for just getting a second opinion. While it might make you feel better to have someone agree with your guess, it is still a guess. You can be pretty sure that it will either be wrong or a simplistic approximation of the true situation.

Often, we cannot find a convenient expert and the time required to get expert help seems likely to delay the project. However, this is only an excuse. Once you build all of your guesses into a program's design, the costs of fixing it later will be vastly greater than any conceivable delay. So bite the bullet—take the time to get the facts needed to do a truly superior job. It will invariably pay off.

Getting Informed Input

Some years ago, one of the projects in my organization was developing a very large programming product. I was convinced that usability was a key need and kept pushing the project managers to have a panel of experts review their designs. The managers kept telling me that it was too early to hold such a review. They argued that they had not yet completed enough of the design work. Then, one day, they told me that it was too late. It

would now be too expensive to make any changes. I insisted, however, and we held a two-day review.

I sat in on the meetings with a panel of a dozen user experts. The programming managers could not answer several of the more detailed operational questions and had to call their programmers for help. At the end of two days, these managers had agreed to make half a dozen changes before they shipped the first release. They now agreed that, without these changes, the product would have been unusable. After I moved on to a different position, this management team never held another expert review. Since these were very smart people, I concluded that they must not have appreciated the enormous importance of getting informed user input.

There are many ways to research requirements questions. Sometimes you will need experts but often a simple prototype and one trial use will tell you the answer. You can also get associates or even non-programmer support people to try to use a prototype. By observing their reactions, you can resolve many usability issues. In other cases, you can accumulate several questions and build a prototype to try out the most promising alternatives. Then you can often get some experts to test the prototypes and to give you their views. In any event, keep track of your assumptions and guesses, and verify them as soon as you can.

Building Superior Products

During my time as IBM's Director of Programming, I received a constant stream of requests to incorporate one or another field-developed program into the IBM product line. These programs were developed by IBM's customer support personnel to help their accounts solve critical problems. We did accept a few of these programs and, in a few years, these few field-developed programs were consistently at the top of our customer satisfaction ratings. Their ratings were well above those for the standard products that had been developed in our laboratories.

In contrast to the company's "standard" products, these field-developed programs had been designed by people who were intimately familiar with the users' environment. These developers intuitively understood precisely how the product should work and didn't have to guess what would be more convenient. They knew. If you want to develop a superior product, either become intimately familiar with the user's environment or have someone with that familiarity readily available to answer your questions. You will need to consult them almost every day.

Changing Problems and Products

If expert advice were all that was needed, requirements would not be such a serious problem. However, there are three issues that complicate the story. First, the users will not be able to tell you what they want. Generally, these are not development people and they cannot visualize solutions to their problems. They could tell you how they work right now but they will have no idea how they would work with a new product, even if its designed were based on exactly how they work today. The second problem is that few users understand the essence of the jobs they are doing. By that, I mean that they do not understand the fundamentals of their jobs, so they cannot help you to devise a product solution that addresses the real operational needs. All they can usually do is help you define how to mechanize the often manual system that they are currently using. The third problem is a kind of uncertainty principle. By introducing a new product, you actually change that user's environment. Often, this will completely change the way the job should be done and it could completely obsolete the original requirements.

What these three problems tell us is that your initial definition of the requirements will almost certainly be wrong. While you must get as close as possible to a usable solution, you must recognize that the requirements are a moving target that you must approach incrementally. To produce a truly superior and highly-usable product for a realistic cost and on a reasonable schedule, you must follow a development strategy and process that assumes that the requirements will change. This requires that you encourage early changes and that you develop and test the program in the smallest practical increments. Also, if possible, get the users to participate in the early testing. Then, when you find what you did wrong or what your customers could not explain or what the users did not know, the amount of rework will be small enough to do in a reasonable time. Then you can quickly take another user checkpoint to make sure you are still on the right track.

These few principles are fundamental to just about every programming job. What is most interesting is that experienced programmers will generally recognize these principles as correct, even while they work in an environment that does not apply them. The challenge that we programmers face is to convince ourselves, our managers, our executives, and even our customers to devise processes, to plan projects, and to manage our work in a way that is consistent with these principles. While it might seem simple and easy to convince people to do something so obvious, it is not. There are many reasons for this, but the most basic reason is that welcoming early requirements changes exposes our projects to unlimited job growth, which has historically led to serious schedule overruns. These are problems that I will address in the next few columns where I discuss the principles for designing products, guiding projects, leading teams, and training and coaching the people who do programming work.

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, Sholom Cohen, Luke Dzmura, Jim Over, Bill Peterson, and Marsha Pomeroy-Huff.

In closing, an invitation to readers

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

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

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.

The views expressed in these articles are the author's only and do not represent directly or imply any official position or view of the Software Engineering Institute or Carnegie Mellon University. This article is intended to stimulate further discussion about this topic.

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.

The Good News About COTS

Lisa Brownsword and Ed Morris

You've heard the stories. Organizations pin their hopes for improving software systems on commercial off-the-shelf (COTS) products only to be burned by poor quality, lack of service, and failures to deliver on time. The propulsion system failure aboard the U.S. Navy's guided missile cruiser USS *Yorktown* is just one example. That failure, which led to the Yorktown being towed back to port, was attributed to a COTS product that failed due to a "divide by zero" problem. But it isn't just the military that has these problems. In 1999, Hershey Food Corporation's revenue fell 12%, a drop that was largely attributed to an inability to get products to market in time for Halloween and Christmas. Corporate insiders blamed the delay on a new installation of a COTS enterprise resource planning (ERP) system.[1]

The reaction to these well-publicized failures—particularly in the defense community—is not unexpected. Programs are hesitant to use COTS products. They don't want to take the chance of becoming another sad story.

The truth is, however, that there are a lot of success stories out there. Consider NASA's new Control Center System (CCS)—the ground-based command and control system for the Hubble Space Telescope.[2] The CCS unifies the functions for telescope command, engineering data processing, data archiving, data analysis, spacecraft and ground system monitoring, and simulation. This replacement system for the original Hubble capability integrates 30 COTS and GOTS (government off-the-shelf) components with one million lines of legacy code and one half million lines of custom code.

Because of the ready availability of the COTS and GOTS components, a prototype for the new system was built in three months. This prototype was invaluable, since it provided a rapid demonstration of the concepts for the system, the end-to-end flow of information, and the new user interface. The first production release of the new system was delivered one year after proof of concept, with many more releases delivered in the ensuing four years. This represented greater productivity than previous systems. The new system provides capabilities that the old, primarily custom-coded system could not, while reducing the number of custom lines of code by more than 50 percent.

---

1 Hayes, Mary. "Hershey's Biggest Treat: No Tricks." *InformationWeek.* October 29, 2002.

2 Pfarr, T. & Reis, J. E. "The Integration of COTS/GOTS Within NASA's HST Command and Control System," 209-221. *Proceedings of the First International Conference on COTS-Based Software Systems (ICCBSS 2002)*. Orlando, FL, February 4-6, 2002. In *Lecture Notes in Computer Science 2255.* Berlin: Springer-Verlag, 2002.

NASA has shared a number of factors it considered to be key to its success:

1. NASA established a clear design philosophy favoring the use of COTS products. NASA adopted an "80-20 rule" that stated that if a COTS or GOTS product met 80 percent of the functional requirements, it would be adopted pending final approval for deferring the remaining 20 percent of requirements.

2. NASA employed a soft-ware architecture that supported upgrading or replacing components without destroying the integrity of the system. NASA accomplished this by encapsulating COTS and GOTS products behind abstract interfaces that minimize the effect on other system components when a product or the way it is used changes.

3. NASA developed clear component-selection criteria that allowed it to make COTS and GOTS selections efficiently, but emphasized hands-on use as the final arbiter for product selection. Products that passed the initial selection but failed the hands-on evaluation were quickly dropped.

4. NASA hired experts who were knowledgeable about the new products to train personnel in design and implementation.

5. NASA reevaluates COTS and GOTS products at each product release. The project has found that some products were successful in the CCS from their initial selection, some were failures and were replaced, and many fell in between. NASA performs periodic reassessments and reevaluations of COTS and GOTS products to replace under-performing products with better ones to improve system capability and long-term viability.

Successful use of COTS products is not limited to the federal sector, either. The Boeing Company has put together a system to manage its commercial airplane configurations and parts.[1] The system is large and complex, with many interrelationships with other Boeing business processes and systems. The Boeing system includes several large COTS applications, COTS infrastructure, plus custom-built integration software and vendor-supported COTS customizations.

Like NASA, Boeing found that using COTS products changes the way systems should be built. For Boeing, as for NASA, a strong architecture was a key. Boeing found that it needed to carefully consider the current and future architectures and direction of COTS products in making product selections. Boeing also started with a loosely defined, flexible set of critical requirements—not the carefully detailed set that is often central to the development of a custom system. The lack of initial rigidly specified requirements allowed Boeing to use COTS products as is, even when that meant refining requirements and changing Boeing business processes. Thus requirements were formed through

---

[1] Baker, T. "Lessons Learned Integrating COTS Into Systems," 21-30. *Proceedings of the First International Conference on COTS-Based Software Systems (ICCBSS 2002)*, Orlando, FL, February 4-6, 2002. In *Lecture Notes in Computer Science 2255*. Berlin: Springer-Verlag, 2002.

negotiation of stakeholder needs and knowledge of the capabilities and embedded business processes of available COTS products. What also mattered was Boeing's focus on identifying rock-solid vendors who offered scalable products that supported Boeing's integration standards and were committed to the success of the Boeing system.

NASA and Boeing successfully built complex systems based on COTS products because they didn't practice business as usual. They changed their business and engineering processes to make the best use of the available COTS products. And they recognized the need for sustained engineering and management effort, such as evaluating new product releases to determine system impact and actively monitoring emerging technology, to keep the products and the system current.

These are only two of a growing number of success stories. In fact, even the Hershey story has a happy ending. A subsequent upgrade to the ERP system has gone smoothly, with the enhanced capabilities reducing costs and processing times.[1] We are always looking for more success stories, and would appreciate hearing about them from anyone who is willing to share them with us. If you have a good news story or want to find out more about the factors that lead to successfully using COTS products, please contact Lisa Brownsword.

For more information, contact—

Lisa Brownsword

Phone
703-908-8203

Email
llb@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu/cbs/?ns

[1] Weiss, T. R. "Hershey Upgrades R/3 ERP System Without Hitches." *Computerworld*, September 9, 2002.

The Acquisition Support Program

By Laura Bentrem

In this era of newly streamlined regulations, everyone involved in U.S. government acquisition hopes to benefit from a process that is more agile and efficient. Yet the government's acquisition, development, and maintenance of software and software-intensive systems continues to be risky. While some acquisition projects may use disciplined processes that lead to success, inconsistent software acquisition practices continue to result in projects that are past due and over budget.

Since its inception, the SEI[SM] has been helping U.S. government acquisition programs in their efforts to improve their processes and minimize risks. Recently, the SEI formalized this ongoing support by creating the Acquisition Support Program (ASP), a group devoted to addressing the unique demands and challenges of acquisition.

The ASP functions as an "experience factory"—a group researching the current state of acquisition, distilling expertise, documenting best practices, and disseminating results. Recording the know-how of government acquisition experts is especially crucial today because many senior government employees will soon be retiring, taking with them knowledge accumulated through years of experience.

By working directly with key acquisition programs to help them achieve their objectives, ASP staff members will refine their understanding of the acquisition environment and improve their ability to characterize that environment for others through best practices and lessons learned. "Working in this area is not new for the SEI," says ASP Program Director Brian Gallagher. "It's just new to approach it in this way."

Central to the operation of the ASP are the Chief Engineers, one for the Army, one for the Navy, one for the Air Force, and one for the Coast Guard. Each Chief Engineer is responsible for quarterbacking the SEI's acquisition efforts for their respective service branch. Equally important, they also are responsible for coordinating across services, ensuring identification of Department of Defense (DoD)-wide acquisition trends, and applying common solutions where appropriate.

A second key component of the ASP is the Acquisition Improvement Team (AIT). AIT is the backbone of ASP, providing the skills and knowledge to execute SEI engagements with individual programs. Team members leverage their individual specialization and expertise on cross-service assignments and meet periodically to exchange ideas.

Rounding out the ASP organization is the Knowledge, Integration, and Transfer team, whose job is to facilitate the accumulation and dissemination of knowledge through case studies, lessons learned, courses, and the like.

The ASP's first task is to establish strategic impact programs (SIPs) with the Army, Navy, Air Force, and Coast Guard. A SIP is a multi-year program of work, part of the strategy of a senior acquisition official who is committed to improvement and change within a particular acquisition community and industry base. The services are eager to begin work with the ASP. In a November 2002 memorandum, Claude Bolton, currently assistant secretary of the Army for acquisition, logistics, and technology, announced that he "chartered the Software Engineering Institute…to be 'at point'" in a joint effort to "promote a dramatic improvement in the acquisition of software-intensive systems." Bolton was also the keynote speaker at the first annual Acquisition of Software-Intensive Systems Conference, an SEI-sponsored conference held in Washington, D.C. this past January.

While the ASP helps acquirers make incremental improvements in the acquisition of software-intensive systems, it will also benefit SEI technical programs by providing opportunities to implement and improve new technologies through acquisition pilots. It has already established such pilots with the Army, Navy, and Air Force. The ASP will pursue acquisition pilots in cases where the match between an SEI technology and a government acquisition program presents an opportunity for acquisition-community learning. Pilots will allow the program to experiment with maturing SEI products and services in real-world acquirer contexts. Analyzing results and documenting lessons learned is part of the normal maturation process for SEI technical work, but the ASP also plans to disseminate these results and lessons to the acquisition community through case studies, course modules, workshops, publications, technical literature, and conferences.

Some of the ASP's work will likely be done in conjunction with other organizations that study acquisition. ASP staff members see opportunities for collaboration with the Defense Acquisition University (DAU), federally funded research and development centers, and other organizations working to improve acquisition. Technical staff from the ASP met with representatives of the DAU early in February 2003 to discuss developing a community of practice.

The ASP's broad challenges are to improve the software engineering skills of acquisition program managers and the acquisition workforce, encourage the use of best practices by collecting and disseminating lessons learned, and improve the government's ability to acquire software-intensive systems. ASP engineers are currently working on engagements that involve the full spectrum of acquisition activities, from refining the language in acquisition contracts to continuously identifying and mitigating risk and

complexity during system development and operational fielding. First and foremost, the ASP is interested in providing guiding principles for acquisition.

"We're coming out of an era of process improvement where we've gotten quite disciplined about processes," said Gallagher. "The challenge is to move in the direction of reform and embrace these ideas of agility and efficiency, but not throw away the very discipline that has been so hard won."

For more information, contact—

Brian Gallagher

Phone
412-268-7157

Email
bg@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu/programs/acquisition-support/?si

OCTAVE[SM] Users' Forum: Helping to Build a Community of Practice

By Pamela Curtis

Communities of practice—groups of people who share a concern, a set of problems, or a passion about a topic, and who deepen their knowledge and expertise in that area by interacting on an ongoing basis[1] —are sometimes instrumental in the successful adoption of a technology. Early adopters may learn as much from sharing lessons learned, implementation ideas, and other information with one another as they do from the technology developer. SEI[SM] developers of the Operationally Critical Threat, Asset, and Vulnerability Evaluation[SM] (OCTAVE[SM]) method—a method for assembling a comprehensive picture of an organization's information security needs—expect this to be true for its users, particularly because they designed OCTAVE to be highly flexible, and they encourage users to alter it to meet their needs.

To facilitate interaction among OCTAVE users, the SEI held the first OCTAVE Users' Forum on September 19–20, 2002, at the SEI offices in Arlington, Virginia. The forum featured a variety of user presentations highlighting OCTAVE field experience, as well as SEI presentations on new method artifacts and new and future directions in managing information security risk.

Thirty-seven representatives from the U.S. Department of Defense (DoD), federal civilian agencies, academia, and private industry attended this first meeting of the OCTAVE user community. Attendees included OCTAVE researchers and developers, people who have implemented OCTAVE in their organizations, OCTAVE transition partners (organizations that are licensed to provide OCTAVE training and services), and people who had expressed interest in learning more about OCTAVE. Their organizations included

- Advanced Technology Institute
- Clark County, Nevada
- Department of Commerce
- Department of Transportation
- General Services Administration
- Library of Congress

---

[1] Wenger, E.; McDermott, R.; & Snyder, W. M. *Cultivating Communities of Practice: A Guide to Managing Knowledge.* Boston: Harvard Business School Press, 2002.

- National Center for Manufacturing Sciences
- National Institute of Justice
- Office of the Comptroller of the Currency
- Secure Communications Solutions, Inc.
- Software Engineering Institute
- Sytel, Inc.
- Telemedicine & Advanced Technology Research Center
- U.S. Nuclear Regulatory Commission
- Xceed Consulting

The forum was funded, in part, by the General Services Administration Federal Computer Incident Response Capability (GSA FedCIRC).

By participating in the forum, attendees met fellow OCTAVE users, heard about the role of OCTAVE in various sectors, and exchanged ideas about how to tailor the method to optimize its effectiveness in various organizational contexts. SEI representatives benefited by obtaining user feedback on OCTAVE.

The forum included moderated sessions on several topics and 10 presentations. Chris Alberts, lead developer of the OCTAVE method, described the shortcomings that the OCTAVE team saw in other approaches to security evaluation as it began initial development of OCTAVE:

- They tend to focus on technology and vulnerability, not on operational risk.
- They don’t make the link among threats, assets, and vulnerabilities and the organization’s business.
- They don’t provide a single implementation that addresses all operational environments.

These, along with the fundamental problem of information assets being at risk due to insecure networks and poor organizational practices, became the critical drivers in the development of OCTAVE.

OCTAVE’s applicability in multiple environments was demonstrated by a panel that described the deployment of OCTAVE by the Defense Health Information Assurance Program. According to Jeff Collmann, an associate professor of radiology at Georgetown

University who is providing project oversight, the goal of the project is to enhance health information-assurance readiness at all U.S. military treatment facilities. A major element of the project is DoD compliance with Health Insurance Portability and Accountability Act (HIPAA) regulations related to the security and privacy of health data. To effect this extensive OCTAVE deployment, 171 teams from all services and regions have been trained in the OCTAVE method and have begun performing evaluations at their own installations. Captain G. Iris Obrams, an M.D. with the U.S. Public Health Service at Coast Guard Headquarters, described how the Coast Guard's interdisciplinary teams are preparing to conduct OCTAVE evaluations at the service's 32 clinics and 70 afloat and 44 shore-based sick bays. Lieutenant Colonel Ray Green, who leads the DoD HIPAA data-security effort and is responsible for ensuring that the DoD meets HIPAA regulations, spoke about his support and sponsorship of OCTAVE as an integral part of the DoD's efforts to comply with HIPAA regulations.

Frank Stasa, chief information officer (CIO) for the Pittsburgh Technology Council and Catalyst Connection, gave a presentation about an OCTAVE-S[1] pilot recently completed by the council. His remarks exemplified how OCTAVE reveals the potential impact of vulnerabilities on business. The pilot involved the CIO and the chief financial officer (CFO), as well as key IT staff members. "Including the CIO and the CFO on the team helped to elevate the importance of information security and make senior management aware of the critical issues facing us," said Stasa. Stasa and his team had not been getting budget increases that they felt were crucial for protecting the council's information assets. The OCTAVE program served to demonstrate how compromise of critical systems would affect the council's business in areas such as productivity, costs, and reputation. "As a result, the CFO readily approved the acquisition of critical hardware that we identified during our workshops," said Stasa. The needed budget increases were also approved soon after the pilot.

These and other presentations from the forum are available in PDF format at *http://www.cert.org/octave/forum/agenda.html.*

The OCTAVE method's developers plan to continue to build the community of practice for OCTAVE. They will be holding the second OCTAVE Users' Forum within the next year (details will be posted on the OCTAVE Web site; see URL below) and are investigating other means of helping OCTAVE users share information. If you have information you would like to share, please send email to *octave-info@sei.cmu.edu* and include the phrase "information sharing" in the subject line.

---

[1] OCTAVE-S is a derivative of OCTAVE that is tailored for small organizations.

For more information, contact—

Bob Rosenstein

Phone
412-268-8468

Email
br@sei.cmu.edu

World Wide Web
http://www.cert.org/octave/

Taking the Road Less Traveled: The CMMI® Continuous Approach

By Lauren Heinz

A greater degree of granularity in organizational performance measurement, a more revealing look at the trouble spots in organizational practices, and a greater focus on organizational strengths—just a few of the reasons software practitioners give for how a continuous approach to process improvement is benefiting their organizations.

"It's like going to a dressmaker and having a dress fitted," says Mary Anne Herndon, a process improvement manager for the Aerospace Intelligence and Information Sector (AIIS) of the Science Applications International Corporation (SAIC). "The continuous representation tailors easily to an organization's current state. It helps you figure out where you are now and where you need to go."

SAIC's AIIS represents a growing number of organizations that are selecting the continuous representation over the staged representation when implementing a Capability Maturity Model Integration (CMMI) model or performing a Standard CMMI Appraisal Method for Process Improvement (SCAMPI℠) appraisal. Since the release of CMMI V1.1 in December 2001, approximately one-third of the Introduction to CMMI course attendees have selected the continuous version of the course and nearly one-third of all SCAMPI appraisal results reported to the Software Engineering Institute (SEI℠) have been from organizations using the continuous representation.

Although the staged and continuous representations are simply two ways of viewing CMMI best practices, and the staged approach remains a proven method for achieving organizational maturity, those with experience using the continuous approach say this newer and less-documented path to software process improvement is transforming software businesses and yielding significant results.

Staged Versus Continuous

Generally, organizations new to process improvement prefer a staged approach, which predefines the process areas required to attain each maturity level (1-5) and thereby provides a roadmap for institutionalizing best practices. Achievement of a maturity level is based on achievement of the practices of a set of related process areas. Organizations that are upgrading from the Capability Maturity Model® (CMM®) for Software, a staged model, are more likely to prefer staged.

In the continuous representation, process areas are organized into four process area categories: Process Management, Project Management, Engineering, and Support. Based on its business objectives, an organization selects the process areas in which it wants to improve and to what degree. Instead of maturity levels, capability levels (0-5) are used to measure improvement. Achievement of a capability level is based on achieving the practices of a single process area. This also enables an organization to implement process improvement in different process areas at different rates. For example, an organization can reach capability level 2 for one process area and capability level 3 for another.

"In moving away from the quest for an overall maturity level," Herndon says, "the continuous approach helps an organization focus on its capabilities and meet business objectives. Concentrating on strengths is a great way to motivate people. Staged tends to be a one-size-fits-all approach; it's pass or fail. That doesn't help you succeed like knowing what you do well and figuring out how to do it better."

A Targeted Look

Terry Rout, a manager with the Software Quality Institute at Griffith University in Australia and chairman of the Australian Committee for Software Engineering Standards, has been using continuous models for more than eight years. Whether through the CMMI Product Suite or ISO 15504, Rout says a continuous approach offers the most enlightening glimpse into how an organization is performing its processes.

"A high degree of granularity occurs naturally from using the continuous framework," says Rout, noting that continuous is widely used in Australia and is the preferred standard for its Department of Defence. "You have to work fairly hard to get those kinds of results from staged."

Like most appraisers, Rout presents his findings in an achievement profile—a series of charts showing the capability levels an organization has achieved in its targeted process areas. "A well-crafted profile can speak volumes about an organization's capabilities. A profile might show that an organization is getting close to capability level 2 in the Project Management process areas, but is at level 0 in some of the Engineering ones. That's a pretty serious warning signal that something needs to change. Otherwise you're just establishing great management of rather poor engineering practices," Rout says. "With staged, you might very well miss how the weaknesses in your process areas are reflected across the board."

Both Rout and Herndon find value using achievement profiles for discussing problem areas with upper management. “It makes it very easy to decide where the investments need to be made, based on the business case,” says Herndon, who is also an authorized SCAMPI Lead Appraiser[SM].

A Case for Using Both

Despite his personal preference for using a continuous approach, Rout says that an organization may find value in both representations at some point in its improvement history. “An organization should use the richness of the continuous view, but they should also consider the priorities of staged,” he said. “In the end, both roads go in the same direction. It’s just a matter of knowing when to skip back and forth.”

Craig Hollenbach, a technical fellow for the Defense Enterprise Solutions (DES) sector of Northrop Grumman, sees the reward of applying both principles. “The staged approach gives you a maturity level, which is easily communicated in the business world,” says Hollenbach, whose sector was appraised at maturity level 5 at its December 2002 assessment and achieved capability level 5 in nearly all CMMI process areas. “But we have a lot of projects that perform only a certain part of the life cycle…we have to pare down the number of process areas we look at and to what extent. This is where continuous is most appropriate.”

Extending the Continuous Approach

Now that Northrop Grumman DES has reached such a high level of maturity and capability, Hollenbach is faced with how to keep the momentum going. He anticipates extending continuous practices to areas outside of the technical arena, such as business services. “We will keep broadening the scope of improvement,” he says. “In 2000, we were wondering what to do next. Now that we’re at level 5, we just have to keep using the tools and processes that got us here to continue to make a difference.”

At SAIC, AIIS is also starting to use the continuous approach as a way to make connections across the organization, from the technical areas to service units such as finance and contracts. “Even though the products are different, by applying the same practices, we are integrating the stovepipes,” Herndon says. “This integration really promotes communication. Now we have a language that everyone speaks.”

Share your Experiences

The SEI is interested in hearing from other organizations that have benefited from using the continuous (or staged) approach to CMMI. Please contact the SEI to tell us about your experiences and what you have learned from them.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu/cmmi/?si