Carnegie Mellon
Software Engineering Institute

news @ sei
interactive

Columns

Features

http://interactive.sei.cmu.edu

Columns

The Architect

Rethinking the Software Life Cycle

RICK KAZMAN, ROBERT L. NORD, MARK KLEIN

Several architecture-centric analysis and design methods have been created in the past 10 years, beginning with the Software Architecture Analysis Method, or SAAM. The SAAM inspired the creation of a number of other methods. The first of these methods created at the Software Engineering Institute was the Architecture Tradeoff Analysis Method, or ATAM, which has in turn inspired the Quality Attribute Workshop, or QAW, the Cost-Benefit Analysis Method, or CBAM, Active Reviews for Intermediate Designs, or ARID, and the Attribute-Driven Design, or ADD, method.

These methods share not only a common heritage, but also a common set of characteristics aside from being architecture-centric. For example, they all use scenarios to direct and focus activities in the methods; and they are all driven by operationalized quality-attribute models. The SAAM was focused on modifiability. The ATAM focuses on tradeoffs among quality attributes. The QAW attempts to elicit and accurately document quality-attribute requirements, particularly in the absence of explicit architecture documentation. The CBAM uses architecture-based information to determine return on investment of the various architectural strategies being considered. An ARID looks at the usability of a design. The ADD method shapes design decisions around quality-attribute considerations. The methods are also all focused on documenting the rationale behind the decisions made; in this way the rationale serves as a knowledge base on which to base both existing and future decisions.

In each of these methods there are activities that logically belong to different parts of the traditional software development life cycle (SDLC). These activities are, however, embedded in the methods, because the methods have been designed to be performed independently, typically by a consultant, a quality-assurance group, or a researcher from outside of the developing organization. Note that being independent is not necessarily a bad thing: many organizations realize the value of having an outside auditor investigate their internal practices, even when the auditor's activities duplicate many of the activities that already take place in the organization. The outsider brings a fresh perspective, a well-honed set of analytical skills, and is (presumably) untainted by existing "group-think" or by office politics.

The point here is that these methods have not normally been integrated with each other or integrated into an organization's SDLC. As a result of being performed independently, each method repeats existing effort and must include activities that actually belong, in concept, to requirements elicitation, or design, or maintenance phases. For example, to analyze an existing software architecture using the ATAM, one needs to understand the business needs that the system

is intended to meet, the requirements, the existing design decisions, and the anticipated changes to the system that will occur in the maintenance phase. This duplication of effort is appropriate for a method that is conducted by an outsider, but is inappropriate if the methods are seen as integral to an organization's normal development process.

A typical SDLC, as it is practiced in relatively mature software-development organizations, has (at least) the following activities:

- identification of business needs and constraints
- elicitation and collection of requirements
- architecture design
- detailed design[1]
- implementation
- testing
- deployment
- maintenance

Of course, this list is not exhaustive, and many of these activities can be broken down into sub-activities (for example, most of these activities include a documentation sub-activity and an analysis sub-activity). Also, this list is not to be taken as implying a particular development process—spiral, waterfall, agile, or any other. The idea here is simply that these are distinct activities, with their own inputs, outputs, specialists, activities, analysis techniques, and notations that need to be undertaken in the development of any substantial software-intensive project.

However, as these architecture-centric methods become more widespread, more widely adopted, and integrated into an SDLC, organizations will inevitably want to tailor them: methods that were created primarily as single-use insertions by an external organization are not necessarily appropriate as a part of a stable, ongoing development process. Consequently, organizations that wish to include elements of quality attribute-based requirements elicitation and gathering, explicit architecture design, and architecture analysis in their life cycles will be best served if they can do so "organically"—seamlessly merging appropriate portions of the methods into their SDLCs.

What this means is that the steps and artifacts of the five architecture-based methods listed above—QAW, ADD, ATAM, CBAM, and ARID—will need to be tailored, blended, and, in some

1. We are using the term "detailed design" here because it is a widely accepted term. It should in no way be taken to imply that there are no details in architecture design. The architect will definitely have to go into details in some areas, to specify, for example, the properties of components and their interactions, while detailed design typically is concerned with algorithms, data structures, and realization.

cases, removed entirely when the activities of these methods are integrated into an organization's existing life cycle.

Merging Methods and Models

In a soon-to-be published technical note,[1] we survey the activities of these methods to understand what they have in common and to propose a means of tailoring the activities so that they can fit more easily into existing SDLC models.

We can think of the typical life-cycle activities in terms of the five methods mentioned above. In particular, we want to understand where the activities in the five methods have their *major* application and impact. In Table 1, we list which artifacts are inputs to the method, outputs from the method, or both.

Table 1:Methods and Life-Cycle Stages

Life-Cycle Stage	QAW	ADD	ATAM	CBAM	ARID
Business Needs and Constraints	Input	Input	Input	Input	
Requirements	Input; Output	Input	Input; Output	Input; Output	
Architecture Design		Output	Input; Output	Input; Output	Input
Detailed Design					Input; Output
Implementation					
Testing					
Deployment					
Maintenance				Input; Output	

Not surprisingly, the methods focus on the life-cycle stages and artifacts that appear earlier in a project's lifetime. This is because these are architecture-based techniques, and an architecture is the blueprint for a system. Once a project is in implementation, testing, deployment, or maintenance, the architecture has been largely decided on, either explicitly or implicitly. There is one exception to this principle: the CBAM may apply to maintenance activities. This is because substantial changes to the system can be made in maintenance that affect the architecture. The "Input; Output" annotation under CBAM in this stage indicates this possibility.

Given the information in Table 1, we can now begin to think about placing these methods into a software development organization's own life cycle. It is unlikely that any of the methods would

1. Kazman, R.; Nord, R.; & Klein, M. *A Life-Cycle View of Architecture Analysis and Design Methods* (CMU/SEI-2003-TN-026).

be included without alteration because, as stated above, they were meant to be independent activities performed by outsiders.

Summary

Architecture-based methods can influence a wide variety of activities throughout the SDLC. These methods have traditionally taken place as independent activities. The relationships between life-cycle stages and the activities embedded in existing architecture-based methods are summarized in Table 2.

Table 2:Life-Cycle Stages and Architecture-Based Activities

Life-Cycle Stage	Architecture-Based Activity
Business Needs and Constraints	• Create a documented set of business goals— issues/ environment, opportunities, rationale, and constraints— using a business presentation template
Requirements	• Elicit and document six-part quality attribute scenarios using general scenarios, utility trees, and scenario brainstorming
Architecture Design	• Design the architecture using the ADD method steps • Document the architecture using multiple views • Analyze the architecture using some combination of the ATAM, ARID, and CBAM
Detailed Design	• Validate the usability of high-risk parts of the detailed design using an ARID review
Implementation	
Testing	
Deployment	

Maintenance	• Update documented set of business goals using a business presentation template • Collect use case, growth, and exploratory scenarios using general scenarios, utility trees, and scenario brainstorming • Design the new architecture strategies using the ADD method steps • Augment the collected scenarios with a range of response and associated utility values (creating a utility-response curve); determine the costs, expected benefits, and return on investment (ROI) of all architectural strategies using the CBAM steps • Make decisions among architecture strategies based on ROI, using the CBAM results

It might be argued that each of these steps involves additional overhead as compared with the traditional, non-architecture-aware SDLC. That is true. But the additional encumbrance is more than repaid by having an architecture that is designed, documented, analyzed, and evolved in a disciplined way. The alternative to adding these steps to the SDLC is to choose a chaotic approach to architecture in the SDLC.

For further reading on this topic, including an overview of the five methods and an analysis of how each one contributes to the SDLC, see *A Life-Cycle View of Architecture Analysis and Design Methods* (CMU/SEI-2003-TN-026), which will be published some time this fall.

References

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

2. Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops*, 3rd ed. (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

3. Clements, P. *Active Reviews for Intermediate Designs* (CMU/SEI-2000-TN-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tn009.html?si>

4. Kazman, R.; Asundi, J.; & Klein, M. “Quantifying the Costs and Benefits of Architectural Decisions,” 297-306. *Proceedings of the 23rd International Conference on Software Engineering (ICSE 23)*. Toronto, Canada, May 12-19, 2001. Los Alamitos, CA: IEEE Computer Society, 2001.

5. Kazman, R.; Barbacci, M.; Klein, M.; Carriere, S. J.; & Woods, S. G. “Experience with Performing Architecture Tradeoff Analysis,” 54-63. *Proceedings of the 21st International Conference on Software Engineering (ICSE 21)*. Los Angeles, CA, May 16-22, 1999. Los Alamitos, CA: IEEE Computer Society, 1999.

6. Kazman, R.; Abowd, G.; Bass, L.; & Clements, P. “Scenario-Based Analysis of Software Architecture.” *IEEE Software 13*, 6 (November 1996): 47-55.

About the Authors

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

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

Mark Klein is Senior Member of the Technical Staff of the Software Engineering Institute. He has over 20 years of experience in research on various facets of software engineering, dependable real-time systems and numerical methods. Klein's most recent work focuses on the analysis of software architectures, architecture tradeoff analysis, attribute-driven architectural design and scheduling theory. Klein's work in real-time systems involved the development of rate monotonic analysis (RMA), the extension of the theoretical basis for RMA, and its application to realistic systems. Klein’s earliest work involved research in high-order finite element methods for solving fluid flow equations arising in oil reservoir simulation. He is the co-author two books: *“A Practitioner’s Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems”* and *“Evaluating Software Architecture: Methods and Case Studies”*.

CMMI in Focus

Using CMMI: How Is it Progressing?

MIKE PHILLIPS

A central purpose of this column is to keep you updated on the adoption progress we are seeing as organizations upgrade to CMMI. In this issue, I'll summarize both what we have seen and what we see coming in the near future.

Adoption Indicators

One of the difficulties of characterizing the adoption of any SEI technology is to determine what measures to use. Anecdotal information is commonly used and easy to obtain. For example, one large aerospace company has banners declaring "CMMI Level 3 in '03." But such anecdotes could lead to the incorrect conclusion that only specific types of organizations are upgrading to CMMI.

The two continuing sources of quality data we have come from our transition partners—qualified DoD and industry organizations authorized by the SEI to help other organizations adopt new and improved technologies—and from SEI staff members. Both of these groups provide training and appraisal services to organizations all around the world and have seen first hand the results of CMMI use.

I find the most telling measure to be the number of people who have taken the "Introduction to CMMI" training course. As of the end of July of this year, a total of 8,837 people have taken the three-day course since its inception a year and a half earlier. This compares with just over 18,000 who have taken the "Introduction to CMM" course, which has been taught for over ten years and will have its last public offering at the SEI in December. We are pleased with this healthy start in CMMI course attendance. About 80% of CMMI instruction has been provided by our transition partners, whose ability to deliver their services at a myriad of sites has had significant impact on CMMI adoption.

The other indicator provided to the SEI is information about appraisals. The number of SCAMPI appraisals is also growing nicely, but not at the same rate as course attendance. As of the end of July, the number of reported Class A SCAMPIs was 133. Last year SCAMPIs represented 13% of the total SEI-supported benchmark appraisals reported. So far this year, the percentage has risen to 22%. I believe this may slightly understate the pace of transition, since we have de-emphasized the use of benchmark-scale Class A appraisals early in the process improvement life cycle. Anecdotal reports of various Class B and C appraisals for initial gap analysis and progress measurement suggest that the adoption curve is healthy.

We have also begun to examine the appraisal data collected to date. While the information thus far is based on hundreds rather than thousands of appraisals, the data confirms what we had hoped. First, the upgrade to CMMI is not a huge hurdle that forces organizations back to a lower maturity

level. Capability Maturity Model for Software (SW-CMM) users who have achieved a maturity level and then adopt CMMI typically are able to achieve a CMMI maturity level that matches or exceeds their SW-CMM maturity-level rating.

Second, the process improvement benefits experienced by those who adopt CMMI typically exceed, and do not fall below, benefits experienced by SW-CMM users who achieve the same maturity level.

The amount of data collected thus far is too small to make any similar assertions about the effect of using CMMI models that include more disciplines, such as integrated product and process development (IPPD) and/or supplier sourcing. We hope to see evidence that each of these expansions offers progressively better integration and improved quality.

Some of you have been interested in which of the two representations is used more, staged or continuous. Data concerning this issue is ambiguous. Those visiting the CMMI Web site select the continuous representation more often than staged. This pattern suggests that users download the continuous representation for use. However, the vast majority of appraisals reported to the SEI have used the staged representation. Most likely this is because the staged approach gives a direct measure of the familiar maturity level. (While a maturity level is also achieved by using the continuous representation with equivalent staging, few lead appraisers have reported their findings that way.) Course attendance is mixed, with a 60/40 split between staged and continuous. I suspect that the number of course offerings for each representation may cause the only differences, as attendance seems equally strong in each course offering.

Finally, we've found that the CMMI Web site has had 800,000 to 900,000 hits every month for the past six months. Those hits are originating from countries all over the world—Russia, Indonesia, Latvia, Argentina, Chile, Italy, Thailand, Israel, and many others—and from a mix of government, commercial, and academic organizations. One of the most highly accessed Web pages is the page that provides the Word version of CMMI-SW. Its popularity may result from users viewing it as the most relevant first step for those upgrading from the SW-CMM.

CMMI Product Suite Updates

When we launched the CMMI V1.1 Product Suite in early 2001, we committed to maintaining the stability of the current version for at least three years. We maintain our commitment to that approach. We receive occasional change requests and have regularly updated an errata sheet for each model that corrects errors found since the model release. There have also been requests to expand CMMI best practices to cover areas such as safety and security assurance and hardware engineering. Other requests include expanding best practices to cover more of the product life cycle, including manufacturing, operations, and disposal.

To begin considering when and which upgrades should be made to the CMMI Product Suite, we announced in September a 90-day comment period (http://www.sei.cmu.edu/cmmi /continuing-improvement.html?si), open to current users of the product suite. Information received from those using the model today will help guide the plans for corrections and improvements tomorrow. Besides the input received during the 90-day comment period, we will also consider change requests submitted since release of V1.1 and input received as part of the CMMI interpretive guidance project.

If you submitted change requests for CMMI V1.0 or V1.02 and your change requests were not incorporated, you must resubmit your comments using the public-review process if you continue to have the same concerns.

The drafting and piloting of proposed changes will take us well into 2005 at the soonest. We believe that the scope of change for the next update will likely be minor, in which case a version V1.2 of the model and appraisal method will result. Even if the scope of changes are extensive enough to result in a V2.0, we are committed to ensuring that these changes will align with the existing suite so that no repetition of training and deployment, such as those associated with V1.1 and V1.0, will be necessary.

Work in Progress

Initial Coverage for Acquisition

One of the disciplines considered for eventual inclusion in the CMMI Framework is acquisition. In December 2000, draft process areas in CMMI V1.02d were released as a way of exploring the possibility of incorporating this discipline. This draft later evolved into the supplier sourcing (SS) addition to V1.1. The treatment of supplier sourcing consisted of adding a single process area, *Integrated Supplier Management*, and informative amplifications of various practices in a few of the existing process areas. This addition of a discipline increased attention on analyzing and selecting suppliers and improving customer–supplier interactions. In spite of this step forward for outsourcing, we determined that further acquisition practices would be needed in the future to aid the government's acquisition efforts.

The CMMI Steering Group has recently asked a group to create a document that will provide broader coverage of acquisition within the CMMI Framework and meet the needs of government acquirers. This approach will provide opportunities for piloting the best practices in the document over the next year, so that a future upgrade to the CMMI Product Suite can be considered. As I've mentioned before, this approach allows us to maintain the stability of the V1.1 Product Suite but expand the communities able to benefit from CMMI.

Technical Notes

It is now easier than ever to find all CMMI-related reports and technical notes on the SEI Web site. Because of the importance of these reports and technical notes for building the understanding of

CMMI-related information, we have created a CMMI Web page for CMMI-related reports (http://www.sei.cmu.edu/cmmi/adoption/reports.html?si).

In the "coming soon" category, we have a few technical notes in various stages of preparation. One in final editing that may be available very shortly was produced by an organization that provides engineering services as a major part of its business. The challenges of developing the services, operating in a dynamic customer environment, and modifying services quickly all make the use of CMMI practices beneficial, although the existing practices often require some interpretation to more closely fit the environment of service providers.

Another team is working to characterize the best safety and security engineering practices to improve the development of safety-critical and security-critical systems. The team has proposed an approach in which there are two goals in a single process area and amplifications for these two disciplines that can be added to the informative material in various parts of existing process areas without changing the goals or practices in those process areas.

Interpretive Guidance

The interpretive guidance project was formed to collect information about how CMMI is being used by software, information technology (IT), and information systems (IS) organizations. Information was collected through a variety of channels and sources in the software and IT/IS communities. An upcoming preliminary report will reveal the data-collection methods used and the raw data collected. The report will also provide some early insight into the adoption of CMMI by the organizations that chose to participate.

Some highlights of this report include the following:

- When asked "Has your organization made a decision about adopting CMMI?" 48% of the respondents stated that adoption was in progress, 15% stated that CMMI was well institutionalized throughout their organizations, 10% stated that their organizations chose not to adopt CMMI, and 23% said the decision to adopt had not yet been made.
- When asked if, in their opinion, CMMI is adequate for guiding process improvement, over 77% of respondents agreed or strongly agreed.
- When asked if including both systems engineering and software in a single model has been a help for them, nearly 65% agreed or strongly agreed with the statement.

The report, titled *CMMI® Interpretive Guidance Project Preliminary Report*, is due to be released this fall.

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

Changing Focus

PATRICIA OBERNDORF

Over the course of the past few years, this column has highlighted a large number of topics important to the successful use of commercial off-the-shelf (COTS) software in software-intensive systems. These topics have ranged from evaluating COTS products to issues relating to modernization of legacy systems. The discourse was generated from the work of the SEI COTS-Based Systems (CBS) initiative, the focus of which has been to learn, mature, and transition principles, methods, and techniques for creating systems from COTS products.

The CBS initiative has thoroughly explored a wide range of issues. Our products have included an instrument and method for CBS risk evaluation (COTS Usage Risk Evaluation, or CURE); a process for developing and sustaining COTS-based systems (Evolutionary Process for Integrating COTS-Based Systems, or EPIC); courses for executives, managers, and practitioners; and technical reports, special reports, and technical notes.

Almost from the outset, we recognized that this tight focus on systems built from COTS products was to some extent artificial: no one builds systems solely from COTS products. Most systems today *include* COTS products, but they also contain components from other sources, such as legacy systems, a "sibling" system with some similar characteristics built by a sister organization, and a development effort that creates a functional component not available in the COTS marketplace. And all of them incorporate some "glue" code that is created to integrate all these components into a coherent system. It turns out that a good deal of what has resulted from the close examination of using COTS products applies equally to using *any* off-the-shelf (OTS) component—the "c" in "COTS" often has little to do with what is necessary to successfully integrate an OTS component into a system.

This realization, coupled with a sense that the CBS territory was largely exposed (even if many problems remain to be solved), has brought us to a crossroads. It is time to step up to the larger scope of improving the creation and sustainment of systems from any set of largely OTS (but not just commercial) constituents. This is a transition from a CBS-focused endeavor to one that considers integration more broadly. It also moves closer to addressing the interoperability and systems-of-systems issues that are so ubiquitous in the world of software engineering today and that are causing so many headaches in so many software projects.

Thus we are announcing the transformation of the CBS initiative into the Integration of Software-Intensive Systems (ISIS) initiative starting in October 2003. While the detailed plans are still taking shape as of this writing, we can outline the intended start and direction.

One part of the initial effort will be to work on terminology and basic concepts: while the essential notions of "integration" and "interoperability" are shared throughout the engineering community, we see the need for greater precision. For instance, are these terms the same? If not, what distinguishes them? (And if they really do connote the same thing, why do we all insist on using them as though they were distinct?)

The rest of the initiative and its work will be founded on several bases in addition to this definitional one. First, we are not intending to relinquish all of the intellectual currency we have gained through our work in COTS-based systems; we intend to put the wisdom we have gained to use wherever appropriate. Second, we have the great benefit of inheriting the work of several research and development projects that have been underway at the SEI for the past year. Perhaps the most significant of these is an effort in system-of-systems interoperability (SOSI); this research effort will provide a head start for our work in understanding the basic issues in interoperability across multiple systems. Another R&D effort during this past year focused on sustainability, and we intend to leverage that work particularly as we grapple with the maintenance problems that confound so many systems developers today: even if interoperability can be achieved between and among systems, and even if we can coerce two or more systems to behave the way we want them to, how do we manage to keep that interoperation going as these systems evolve? How do we cope when components—whether COTS or other OTS—are constantly being upgraded, refreshed, and retired, outside the control of the sustaining agency?

We intend to place a primary focus on the study of existing and planned systems of the U.S. Department of Defense (DoD) and other large organizations, examining them for identification of the important integration problems, for successful methods and techniques that are in use, and for lessons learned that can be shared with others. We will also explore more detailed integration and interoperability techniques, those already in use, and others that may need to be created and matured. We will explore current practice particularly in the areas of component evaluation, architectural qualities, opportunistic evolution, and the management of assumptions. We will also investigate techniques for rapid integration of systems. Another path will ensure that metrics considerations are a part of this work from the outset. And we will examine applicable DoD policies that may foster—or inhibit—the achievement of a desired state of integration or interoperability.

We will consider the proposition that full integration, especially at the system-of-systems interoperability level, depends as much on integration at the programmatic level as on detailed engineering methods and techniques. We expect to explore the use of an acquisition modeling approach to compare the acquisition strategies and plans of two programs and determine how adjusting and harmonizing those strategies and plans can improve program-to-program (and therefore system-of-systems) integration.

We are excited about this enlargement of scope and the opportunity to build on all of the CBS work to create solutions for a broader range of systems. While the SEI will continue to support and transition the results of the CBS initiative, the readers of this column can look forward to articles that explore problems and solutions regarding integration and interoperation. In keeping with this change, we will retire the “COTS Spot” name.

As for our new initiative, we wryly admit that with a name like “ISIS” we could be in for a bit of ribbing: how many of our colleagues can claim the protection of an ancient Egyptian goddess? But for the purpose of this venue, we have decided that a name that is a bit more expressive would be appropriate. So starting in the next issue, look for the new column called “Eye on Integration.” We’re looking forward to an exciting time!

About the Author

Tricia Oberndorf is Director of the Dynamic Systems Program at the Software Engineering Institute (SEI). She has been a principal of the COTS-Based Systems Initiative and concentrates on the investigation of issues in integration and acquisition of systems from commercial and other off-the-shelf components. Early her career focused on the investigation of integration and open systems questions, primarily in the context of computer-aided software engineering environments. Prior to the SEI, she was with the Navy for more than 19 years. Oberndorf has co-authored a book titled *Managing Software Acquisition: Open Systems and COTS Products*. She received an MS in Computer Science from UCSD and a BS in Computer Science from Oregon State University.

Security Matters

Use Care When Reading Email with Attachments

LAWRENCE R. ROGERS

You probably receive lots of mail each day, much of it unsolicited and containing unfamiliar but plausible return addresses. Some of this mail uses *social engineering*[1] to tell you of a contest that you may have won or the details of a product that you might like. The senders are trying to encourage you to open the letter, read its contents, and interact with them in some way that is financially beneficial—to them. Even today, many of us open letters to learn what we've won or what fantastic deal awaits us. Since there are few consequences, there's no harm in opening them.

Email-borne viruses and worms operate much the same way, except that there *are* consequences, sometimes significant ones. Malicious email often contains a return address of someone we know and often has a provocative subject line. This is social engineering at its finest—something we want to read from someone we know.

Email viruses and worms are common. If you've not received one, chances are you will. Here are steps you can use to help you decide what to do with every email message with an attachment that you receive. You should only read a message that passes all of these tests.

1. The ***K*now** test: Is the email from someone that you know?
2. The ***R*eceived** test: Have you received email from this sender before?
3. The ***E*xpect** test: Were you expecting email with an attachment from this sender?
4. The ***S*ense** test: Does email from the sender with the contents as described in the Subject line and the name of the attachment(s) make sense? For example, would you expect the sender—let's say your mother—to send you an email message with the subject line “Here you have, ;o)” that contains a message with attachment—let's say AnnaKournikova.jpg.vbs? A message like that probably doesn't make sense. In fact, it happens to be an instance of the Anna Kournikova worm, and reading it can damage your system.

1. Social engineering is the art and science of getting people to comply with your wishes. It is not a method of mind control, it will not enable you to get people to perform tasks wildly outside of their normal behavior, and it is far from foolproof. (From http://packetstormsecurity.nl/docs/social-engineering/aaatalk.html.)

5. The ***V*irus** test: Does this email contain a virus? To determine this, you need to install and use an anti-virus program. That task is described in Task 1, "Install and Use Anti-Virus Programs," of *Home Computer Security* (http://www.cert.org/homeusers/HomeComputerSecurity/#1).

You should apply these five tests—**KRESV—**to every piece of email with an attachment that you receive. If any test fails, toss that email. If they all pass, you still need to exercise care and watch for unexpected results as you read it.

Now, given the **KRESV** tests, imagine that you want to send email with an attachment to someone with whom you've never corresponded. What should you do? Here's a set of steps to follow to begin an email dialogue with someone.

1. Since the recipient doesn't already ***K*now** you, you need to send him or her an introductory email. It must not contain an attachment. Basically, you're introducing yourself and asking permission to send email with an attachment that the person may otherwise be suspicious of. Tell the recipient who you are and what you'd like to do, and ask for permission to continue.
2. This introductory email qualifies as the mail ***R*eceived** from you.
3. If the recipient responds, honor his or her wishes. If he or she chooses not to receive email with an attachment from you, don't send one. If you don't hear from the recipient, try your introductory email one more time.
4. If the recipient accepts your offer to receive email with an attachment, you are free to send it. The recipient now ***K*nows** you and has ***R*eceived** email from you before. He or she will also ***E*xpect** this email with an attachment, so you've satisfied the first three requirements of the **KRESV** tests.
5. Whatever you send should make ***S*ense** to the recipient. Don't use a provocative subject line or any other social engineering practice to encourage the person to read your email.
6. Check your attachment for ***V*iruses** before sending it. Having gained the trust of the recipient, you don't want to destroy it by inadvertently sending a contaminated attachment.

The **KRESV** tests help you focus on the most important issues when sending and receiving email with attachments. Use it every time you send email, but be aware that there is no foolproof scheme for working with email, or security in general. You still need to exercise care. While an anti-virus program alerts you to many viruses that may find their way to your computer, there will always be a lag between when a virus is discovered and when anti-virus program vendors provide the new virus signature. This means that you shouldn't rely entirely on your anti-virus programs. You must continue to exercise care when reading email.

Use the checklist (http://www.cert.org/homeusers/HomeComputerSecurity/checklists/checklist3.pdf) from *Home Computer Security* to help you make decisions about opening email attachments.

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 article is adapted from Task 3 in *Home Computer Security*, which can be found at http://www.fedcirc.gov/library/documents/homeusers/index.html and http://www.cert.org/homeusers/HomeComputerSecurity/. This work was funded by the General Services Agency of the U.S. Government.

Software Product Lines

The Man with the Plan

PAUL CLEMENTS

One of the crucial aspects of getting a software product line production capability up and running is to synchronize the production of the core assets with the production schedules of the products that need to use them. At the heart of the problem is knowing which core assets need to be produced first, and channeling all of the product-builders' needs and demands through a disciplined process that results in a prioritization that is optimal for the entire product line, as opposed to any one product. For this process to work, planning is essential.

In October of 1999 I went to visit the software core asset manager for a product line effort we were helping to get off the ground. It had been a few months since my last visit, and in the interim the product line effort had undertaken a massive and detailed planning effort. The manager wanted to show me the results. "I have to give my manager credit," he said, referring to the overall product line manager. He began unrolling a cylinder of paper over a conference table, spreading out a sheet roughly the size of a queen-size bed, covered with an enormous PERT chart. "He asked me if it would be worthwhile to hire a management consultant to help us build this project plan for the core assets and keep it up to date. I said 'Sure, why not?' Since then, the guy has worked full time on-site for a week every other week for several months."

A full-time management consultant hired to help with planning sounded expensive, if not extravagant. Had it paid off?

"I can't believe what a difference it's made," he said. "Just this week, I've had four or five product managers come to me and say they needed to have a particular task moved over here" – with a sweeping motion of his hand he indicated the left side of the huge chart, home to the earliest tasks– "and they needed it now." He smiled–rather serenely, I noticed. "I say, 'Well, there are about 2,000 tasks in the plan. Yours is one. We can move it, but it's going to affect everything else. We'll have to see if that's what management wants.'"

This resonated immediately. Almost three years earlier we had performed a risk evaluation at this organization, and one of the most critical risks to their ambitions of launching a product line was lack of planning. Project engineers intent on getting a particular product out the door as soon as possible would make heated demands on the newly-formed, embryonic, and hopelessly overworked core asset team. When they balked, the project engineer could point to a deadline, whether real or invented, whereas the core asset team could point to nothing. One of the risk evaluation participants lamented, "We never had enough information to be able to justify a 'no' answer."

Not that "no" was anyone's goal. But some well-placed "no"s will keep an organization focused on the long-term product line goals and not let them get trodden over by short-term single-project concerns. Learning how and when to say "no" is a big part of product line culture.

And that wasn't all. The manager went on to explain that his consultant would visit the product development, systems engineering, and hardware engineering groups to understand the inter-group dependencies and account for them in the core asset software plan. Groups who had poor plans of their own could not justify the arbitrary deadlines they once were able to impose on the core asset group. The message was "When you understand what you really need and why you need it, come see us and we'll see what we can do to help you. Until then, poor planning on your part does not constitute an emergency on our part." It was a message that was never delivered explicitly; it never had to be.

As we went through the plan, I noticed something I hadn't heard before. He told me he thought that the software core assets were going to be ready sometime in 2001.

"Really?" I said. "The last I heard, you were on the hook to deliver those this spring."

He nodded. "April 2000," he agreed. That was only six months hence.

"And now not until 2001?"

"Yep."

"What happened?"

With another sweep of his hand, he indicated the leftmost edge of his furniture-sized PERT chart, where there dwelled a column of blue boxes indicating dependencies on external projects such as the systems engineering group or the hardware selection group. "These won't be ready in time," he said, trying very hard to suppress a grin.

I thought a minute. "There's no way that you would have been allowed to slip your schedule almost a year without this plan, is there?"

"Nope," he said, grin no longer suppressed. "Now we're going to get to do it right."

The point, as we both knew, is not that a year delay in the product line was a thing to be smug about. But since the dependencies really did exist, the delays would have occurred anyway. But they would have occurred maybe a month at a time, each time precipitating unpleasant surprise, finger-pointing, hand-wringing, disappointment, and (more than likely) management pressure that intensified each time, to no good end. The plan had bought the core asset group the time it would have had anyway, but without planning that time would have arrived in jerky four-week chunks and not have been used to best advantage.

If the truth will set you free, then I was looking at a very liberated software core asset manager.

Besides uncovering the truth, planning had another ironic side effect. Considered a way to let the core asset group to say "no," planning now let them avoid having to. The message is one of cooperation, not confrontation:

- "We'll happily move your task up—after all, we're here to serve--but here are the ramifications. Let's see if the product line manager agrees with those effects wrought on other development groups and the product line itself."
- "We'll happily move your task up. Show me in your plan where it feeds in to a critical path so we can agree on the timing."
- "We're all ready to produce the core assets in the amount of time we promised, but we can't make progress until we get the following information from the following groups."

"That consultant makes a lot of money," the manager volunteered, rolling up the chart. I knew the manager was talking in terms of a per-hour rate, and consultants being consultants, I had no doubt that the rate was breathtaking. But if we were measuring the serenity that this information imparted to the core asset group, a serenity that I had not observed heretofore in this manager, then I got the distinct impression that the manager thought the consultant was not overpaid at all. I was inclined to agree.

About the Author

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

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

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

Watts New?

Some Programming Principles: Projects

WATTS S. HUMPHREY

This is the third in a series of columns on programming principles. The first column in March discussed some general principles of programming, with particular emphasis on the changing and ill-defined nature of software requirements. The second column in June addressed those software principles that relate to the nature of our products. These principles concern the fact that our products are intangible, can last essentially forever, and are increasingly important to our businesses and to society in general.

In this column, I discuss the principles that relate to software engineering projects. Many of these principles are common to engineering projects of all kinds, but software projects present some issues that make our work uniquely challenging and rewarding. In discussing project principles, it is important to start with the fundamental purpose or objective of most software projects. This is to develop or enhance a product to meet a business need. In fact, this defines the following important principle about almost any software project.

The principal objective of almost all software projects is to meet a business need.

This means that the schedule, cost, and quality of the work is of paramount importance. Therefore, in addressing the principles that govern software projects, I will talk about schedule, cost, and quality. Of course, by quality, I refer to the ability of the product to reliably produce the user's desired results. While there are many other important project considerations, they all relate directly or indirectly to schedule, cost, and quality performance. In closing, I will comment on the benefits of successful projects and the characteristics of project success from both a business and an engineering perspective.

Project Schedule Performance

Project schedule performance has three interesting characteristics. First, with few exceptions, if you don't meet the committed schedule or a revised schedule that everyone knows about and has previously agreed to accept, your project will not be judged fully successful. In other words, schedule performance comes first. For example, in assessing the best projects, any that are late, even by only a few days or weeks, never make it to the top of the list.

Second, and particularly for projects that last for more than a few weeks, the important stakeholders need to know where you stand and if you are likely to finish on time. The end users need to make installation and conversion plans, the testers need to schedule their resources and facilities, and management needs to know if there will be any business problems or if they will have to step in to accelerate or redirect the work.

Keeping managers and customers properly informed requires accurate and timely status reports. This is the most common area where software projects run into difficulty. Since software engineers rarely know precisely where they stand against their schedules, they cannot make a convincing report on their status or accurately forecast when they will finish. From a management perspective, this proves that they do not know how to manage their work. This leads to distrust, management meddling, and often even to project redirection or cancellation. In fact, I have seen management interference destroy projects that otherwise would have been reasonably successful, all because of poor status-tracking and reporting.

Third, schedule performance by itself is not personally rewarding. While it is essential to meet other people's criteria for project success, the satisfaction that comes from meeting a schedule is ephemeral. After a rather brief period, management's reaction will become "So what was the big deal?" You just did what you said you would do. Consider schedule performance like a down payment: it is essential to get into the game and it will improve your personal reputation with management, but, by itself, it will not produce lasting rewards or personal satisfaction.

Project Cost Performance

Most engineers feel that if they meet the schedule, any cost overruns will be small and not worth worrying about. But that is becoming less and less true. As many software projects last longer and become more expensive, both cost and schedule management are increasingly important. To see why, suppose that you were building a new house and that the builder assured you that he would finish on schedule. Then, just before the final closing, he told you that your changes had cost more than expected and that he had to pay some overtime to finish on the promised date. The bill is therefore $50,000 more than you had previously agreed. This would likely cause a serious problem. The mortgage commitment probably would not cover the added costs and you probably don't have that kind of money lying around. While meeting the schedule was nice, the cost overrun could easily be a deal breaker.

Cost is equally important for software work. However, the time to address cost problems is when you first detect them, not at the end when no one has any flexibility. If your customer wants a change, if you have technical problems, or if your original estimates were way off, you should figure out what the job is now likely to cost and get agreement before you proceed with the work. While this will involve lots of nitty debates during the project, it will avoid the big cost surprises at the end. When you are at it, also make sure that you put all of these cost negotiations and agreements in writing.

Of course, the problem that cost management presents for most software projects is that we rarely know enough about the costs of our work to estimate the impact of small changes. This again is a fairly unique problem for software, but it is a problem we must learn to address if we are ever going to effectively manage the costs of our work.

Project Quality Performance

While cost and schedule performance are important, our product must work. Also, cost and schedule performance are not, in the long run, satisfying from an engineering point of view. We like to build great products. If you finished development on schedule and within planned costs but the product was a disaster, the brief glory you got from delivering on time would quickly fade. While being known for meeting schedules is important, being known as the developer of a poor-quality product is an engineering kiss of death.

The world is changing and the importance of delivering quality products will only increase. After they have lived through a few software disasters, many software developers properly conclude that it is better to deliver good products late than to produce poor products on time. This strategy, however, will continue to have business problems. The engineers who get ahead in the future are almost certain to be those who deliver quality products on time and for their committed costs. If no one in your organization can consistently do this, there are lots of organizations all over the world that would love to take your place. Many of these organizations are already demonstrating the ability to do superior work on schedule and they are growing very quickly. So, if you want to keep your job, it would be a good idea to learn how to meet both the business *and* technical needs for your products.

Project Success Criteria

The real satisfaction we get from our work is the thrill of working on a great team, creatively using the latest technology, and producing superior products that truly meet our users' needs. As engineers, we are creators and we want our creations to be accepted, used, and appreciated. This requires quality products. In short, the truly successful and rewarding projects of the future will be those that produce quality products on schedule and for their committed costs. These are the only projects that will consistently satisfy the engineers, their managers, and the users.

The criteria that engineers and managers use for judging project success historically have been very different. As I will discuss in the next column, it is important that we learn to consistently meet both management's success criteria as well as our own. The key point to remember, however, is that management decides who to hire and how much to pay for our work. This means that we must meet management's criteria if we want to get ahead. In the next column, I will conclude this series on programming principles with a discussion of the perceptions that engineers and managers have about project success and what this means for each of us.

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, Don Firesmith, Marsha Pomeroy-Huff, Julia Mullaney, Bill Peterson, and Alan Willett.

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.

Features

e-RA Method Simplifies Decision Making for Authentication Requirements

How can the U.S. government determine how far it should go to ensure that the people using its Internet sites are who they say they are? On which sites should the government use the most sophisticated and expensive technology to authenticate users? On which sites can it employ simpler solutions?

A technique developed by an SEI team enables organizations to analyze their own authentication risks and requirements for their Internet sites, without having to call in authentication experts.

The technique, called e-RA, was developed for the General Services Administration's Office of Electronic Government by an SEI team consisting of Rich Caralli, Audrey Dorofee, Eileen Forrester, Bill Wilson, Bradford Willke, and Erin Whiteman. The term "e-RA" is short for "e-authentication risk and requirements assessment." It is a technique to elicit requirements for authentication for transaction-based systems, based on the risks to those systems and to users. The purpose of e-RA is to guide the selection of an appropriate level of authentication that will enable the system to resist threats to data, users, and organizations that could result from unauthorized system transactions.

Common technology-centric approaches—such as, "just use public key infrastructure" or "user id and password should do it"—may be either too much or too little. A solution that is too much for the risks involved could be costly and tough to implement, manage, and maintain. It could also present an unnecessary barrier for intended users. However, a solution that is too little will not provide enough protection, resulting in dire consequences for the organization, and possibly users.

The government could have approached the problem in a number of ways, the SEI's Caralli says. "You could look at the available authentication technologies and just apply one or more of them, which might require that each and every user get a $20 certificate and install it on his or her computer." Instead, the SEI team developed an approach to discover the risks of unauthorized use in a range of scenarios covering 22 electronic-government, or "e-government," initiatives comprising government-to-government, government-to-business, government-to-citizen, and internal government transactions. Examples of transactions include government travel processing, inquiries about social security benefits, and filing grant applications.

After conducting the assessment, organizations can say accurately, "this is what we need to avoid, and it takes this level of authentication to avoid it," Caralli says. "They can make decisions based on risk, then decide which technology is the most cost effective."

Or, the organization might decide to change the Web site, says Mark Liegey, a program analyst with the U.S. Department of Agriculture (USDA). Liegey was the team lead for risk assessment for the e-government e-authentication initiative, which is part of the President's Management Agenda and promotes the reuse of credentials across government. "The e-RA approach gives us the opportunity to think about whether there are other ways to reduce risk than with an expensive solution," Liegey says. "If the e-RA approach shows that a transaction exposes a user's social security number, we might ask whether we even need to ask for the social security number."

e-RA Assessments

An e-RA assessment is usually performed by using the e-RA database for data collection and analysis. Organizations perform four major activities in an e-RA assessment. They

1. record information about their e-government initiative
2. set risk tolerances for their organization
3. identify the transactions of their system or initiative
4. analyze those transactions for risks related to authentication in order to produce requirements

Risk-tolerance criteria are benchmarks or measures against which the organization can evaluate the consequences of unauthorized transactions. The same consequence could mean different things to different organizations. The organization develops its own weighting factors to describe what is important and to determine the consequences and impacts that it most wants to avoid.

Transactions are the vehicle for creating system data, inquiring on it, modifying it, or deleting it. After conducting an e-RA assessment, a system owner has a mapping of each transaction to an authentication level. This mapping can be used to develop authentication requirements and to then choose and implement technical and other operational solutions for authentication.

Each type of transaction (create, inquire, modify, and delete) usually maps to a specific type of undesired outcome if an unauthorized user executes it:

- *Create* transactions permit data to be recorded or documented. Unauthorized use can result in the creation of data that is misleading, fraudulent, or used for unintended purposes. The creation of unauthorized data can interfere with the authorized use of existing data.
- *Inquire* transactions typically provide access to view data. Unauthorized use can result in disclosure of data to users other than the owner.
- *Modify* transactions allow the modification of existing data. Unauthorized use affects the integrity of the data and the ability to use it for the purpose intended by the owner or other authorized users.
- *Delete* transactions allow data to be deleted temporarily or permanently. Unauthorized use causes the data to be unavailable to the owner and other authorized users.

Transition to Broad Use

The USDA's Liegey says the government got a better-codified and simpler solution than anyone expected. "We originally wanted the SEI to figure out a technique for doing authentication assessments and then teach us to do the assessments. So it started out being very expert-driven. But every time we did a pilot, we streamlined the process and realized it didn't have to rely on hands-on expertise from the SEI for every assessment. Instead, the SEI experts could provide us with a tool to automate the process. From a transition point of view, that's one of the most exciting things about this project. We got to something suitable for broad use in the federal government, and we got more than we expected."Liegey expects that the e-RA technique will become a recommended best practice for federal agencies.

The e-RA tool and the e-Authentication Risk and Requirements Assessments Guide are available for download from the governments e-Authentication Web site.[1]

Forrester says the team plans to study the technique to see if a similar risk-based approach can be used for requirements other than authentication. The team will be writing a technical report to describe how e-RA was developed. The e-RA team welcomes inquiries about e-RA, the technical report, or the potential for other risk-based approaches to requirements elicitation.

For more information, contact—

Bob Rosenstein

Phone
412-268-8468

Email

br@sei.cmu.edu

1. http://www.cio.gov/eauthentication/

New Software Architecture Curriculum Developed

ERIN HARPER

Based on decades of experience with software-intensive systems and supported by four widely acclaimed books in the SEI Addison-Wesley Series, the SEI has developed a software architecture curriculum.

The six courses and three certificate programs that make up the curriculum equip software professionals with state-of-the-art practices for designing, documenting, evaluating, and implementing software architectures.

Because a software architecture acts as the blueprint for a system and for the project that develops that system, getting it right is imperative. An architecture is "right" if it meets the behavioral and quality-attribute goals (such as performance or security) defined for the system, which will in turn help an organization reach its broader business goals. "If you are responsible for this critical artifact but lack the knowledge or point of view needed to develop it, the system's stakeholders will ultimately suffer in some way," says Mark Klein, a member of the SEI's technical staff who helped develop the curriculum. Much of the technology taught in the curriculum originated within the SEI. "Every instructor was instrumental in the development of the technology we're teaching, and many of them literally wrote the book on the subject," says Paul Clements, another SEI staff member who contributed to the development of the curriculum.

The Courses

The curriculum is designed so that participants can take single classes in their areas of interest or complete one or more of the three certificate programs.

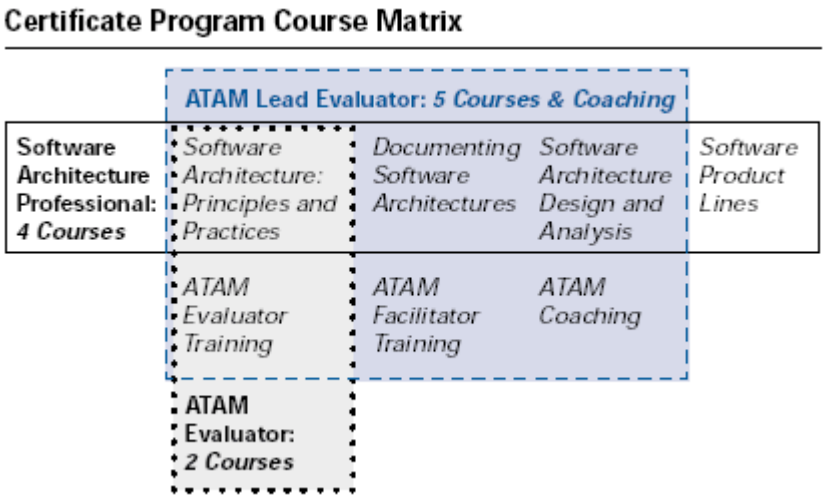
The course Software Architecture: Principles and Practices teaches participants about the concept of software architecture—defined as the structure or structures of a system, comprising software elements, the externally visible properties of those elements, and the relationships among them. Based on the book *Software Architecture in Practice, 2nd Edition*, the course emphasizes the importance of the business or mission context in which a system is designed.

The Documenting Software Architectures course provides in-depth coverage of effective software architecture documentation practices. Participants learn how to produce a comprehensive documentation package useful to many different stakeholders, including developers, managers, and system maintainers. This course is based on the book *Documenting Software Architectures: Views and Beyond*.

The Software Architecture Design and Analysis course answers the questions, "Which design decisions will lead to a software architecture that successfully addresses the desired system qualities?" and "How do you know if a given software architecture is deficient or at risk relative to

its target system qualities?" A software architecture design method called Attribute-Driven Design (ADD) is presented in the course, which also introduces a family of software architecture evaluation methods based on the Architecture Tradeoff Analysis Method[SM] (ATAM[SM]). The books *Software Architecture in Practice, 2nd Edition* and *Evaluating Software Architectures: Methods and Case Studies* serve as the foundation for this course.

The Software Product Lines course provides a comprehensive introduction to software product lines, which are sets of software-intensive systems that share a common, managed set of features satisfying a particular market or mission area, and are built from a common set of core assets in a prescribed way. Adopting a product line approach to software is both a technical and business decision. This course covers the essential technical and management practices needed to use product lines successfully. This course is based on the book *Software Product Lines: Practices and Patterns*.



The ATAM Evaluator Training course prepares software architects to participate in software architecture evaluations using the ATAM. Based on the book *Evaluating Software Architectures: Methods and Case Studies*, this course includes lectures, videotaped enactments, interactive exercises, and hands-on practice.

Building on the skills taught in ATAM Evaluator Training, the ATAM Facilitator Training course focuses on the social and leadership skills required to lead successful architecture evaluations. Class exercises allow participants to practice the tasks required of lead evaluators.

The Certificate Programs

The three certificate programs offered are the Software Architecture Professional program, the ATAM Evaluator program, and the ATAM Lead Evaluator program.

Beginning with an introduction to software architecture fundamentals, the Software Architecture Professional certificate program helps participants gain experience in architecture documentation, design, and analysis techniques. The four-course sequence also demonstrates how these techniques can be used effectively with a product line approach.

Qualified participants who complete the courses in the ATAM Evaluator certificate program are authorized by the SEI to participate in ATAM architecture evaluations. The five courses and field exercise in the ATAM Lead Evaluator certificate program provide qualified participants with the technical depth, social techniques, and experience they need to effectively lead software architecture evaluations using the ATAM. SEI-authorized lead evaluators then attend yearly ATAM update workshops to maintain their skills and status. A list of authorized lead evaluators is provided on the SEI Web site at http://www.sei.cmu.edu/ata/ale.html?si.

For more information, contact—

Tim Denmeade

Phone
412-268-8243

Email
td@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu/ata/arch_curriculum.html?si

New Acquisition Conference a Hit with Attendees

Janet Rex

The first Conference on the Acquisition of Software-Intensive Systems was held January 28-30, 2003, in Arlington, Virginia. U.S. government acquisition organization employees, their support organizations (support contractors and federally funded research and development centers), and federal government contractors met to share their experiences and insights about acquisition.

The conference was co-sponsored by the SEI and the Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics.

Why Acquisition?

While there were already conferences on software engineering, the people who actually spend billions of government dollars on software usually made do with perhaps a track at a conference aimed at engineers. This was the first conference aimed exclusively at the people in U.S. government program offices responsible for acquisition.

The conference focused on improving the acquisition of software-intensive systems throughout government. It drew from the experience and expertise of practitioners in the field to provide insights for acquisition professionals who are trying to enhance the effectiveness of their methods and techniques.

While organizers expected up to 150 attendees at this first conference, more than 200 people participated. The majority of attendees were from the Department of Defense (DoD), but there were also representatives from a wide range of U.S. governmental organizations, including the Internal Revenue Service (IRS), National Aeronautics and Space Administration, and the Federal Aviation Administration.

Conference Highlights

The keynote address was given by the Honorable Claude M. Bolton, Jr., Assistant Secretary of the Army (Acquisition, Logistics and Technology). His presentation covered the critical role of software in the U.S. Army's transformation, and focused on acquisition, specifically for future combat systems. The Army's Strategic Software Improvement Program must meet the challenge of building software systems that are flexible, expansible, sustainable, affordable, and secure, Bolton said. By bringing broad and strategic thinking to this process, the Army plans to

institutionalize improved acquisition processes, to develop enterprise initiatives, and to cultivate strategic partnerships with the SEI and other organizations.

"SA-CMM® in a Large Complex Program," presented by Lloyd Anderson of the IRS and Hugh Gray of Computer Science Corp., covered work performed by the IRS Business Systems Modernization Office. The Software Acquisition Capability Maturity Model® (SA-CMM) was selected as the acquisition management model used to develop capabilities for acquiring business solutions. The presentation covered the organizational challenges that were overcome to implement SA-CMM and the keys to a successful implementation, including establishing a process-improvement infrastructure, aligning process improvement to the organization's business strategy, and using the process to address issues important to project teams. SA-CMM has allowed the IRS to field six major functional capabilities in four years including a modernized help desk, a new application for agents computing complex business tax returns, and modernized telephony for the world's largest call center.

Several papers highlighted relevant SEI work. Ted Marz and Jim Smith shared their insights gained from evaluating several recent acquisitions in "The State of Practice in DoD Acquisitions and Some Proposed Alternatives." They noted that while Capability Maturity Model for Software (SW-CMM) Level 3 is a great start, a more sophisticated understanding of a contractor's abilities is often necessary. The presenters recommended that a Software Capability Evaluation be performed as part of source selection.

"TRL Corollaries for Practice-Based Technologies" by Caroline Graettinger, Suzanne Garcia and Jack Ferguson offered a draft set of technology readiness level (TRL) descriptions for use in assessing practice-based technologies (PBTs), because improvement of acquisition practices will require the implementation of PBTs. A study by the SEI and the U.S. Army Communications-Electronics Command in 2002 showed that current use of TRLs is not readily applied to information-assurance PBTs. These enhanced TRL descriptions are one proposal to remedy the situation.

Tricia Oberndorf and Pat Place delivered a presentation on "Acquisition Practice: Good and Bad," which focused on the acquisition of commercial off-the-shelf-based systems. Using the SA-CMM as a basis, they compared the acquisition experiences of two federal agencies that were involved in acquiring, tailoring, and deploying a financial-management package.

Feedback from attendees on these presentations and others was positive. Many commented that the conference theme and size were ideal, giving them plenty of opportunities to get together and share their experiences. The content of presentations was informative and sparked interesting

discussions. The presentations are available on the SEI Web site at *http://www.sei.cmu.edu/products/events/acquisition/2003-presentations/?si*.

The next Conference on the Acquisition of Software-Intensive Systems will be held January 26-28, 2004, in Arlington, Virginia. For more information, see the conference Web site at *http://www.sei.cmu.edu/products/events/acquisition?si*.

For more information, contact—

Jack Ferguson

Phone
412-268-5800

Email
jrf@sei.cmu.edu