Carnegie Mellon
Software Engineering Institute

news @ sei interactive

Volume 3 | Issue 3 | Summer 2000

In This Issue

http://interactive.sei.cmu.edu

news @ sei interactive

From the Director

Stephen E. Cross

In this issue of news@sei, I am pleased to announce the SEI's contract with the U.S. government has been renewed for the next five years, through September 2005. As you may know, the SEI was established in 1984 at Carnegie Mellon University as a federally funded research and development center (FFRDC) dedicated to advancing the practice of software engineering and improving the quality of systems that depend on software. The SEI's new five-year contract with our sponsor, the Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics [OUSD(AT&L)], represents the third time that the government has endorsed our mission by renewing the SEI contract; previous contract renewals were in 1991 and 1996.

The process for establishing and renewing an FFRDC contract is detailed in the government's Federal Acquisition Regulations. Contract renewal is a thorough and difficult undertaking. Beginning in the summer of 1999, the US government conducted an extensive review of the SEI, culminating in a unanimous recommendation by government reviewers to renew the SEI contract. The recommendation stated that:

> *The SEI has demonstrated, and its various sponsors have attested to, its high technical quality and its responsiveness to their needs. The work of the SEI, in supporting its sponsors, is critical; no other organization exists that is capable of delivering these services at the same high quality, technical level and with objectivity and independence. The Panel commends the SEI in its accomplishments to date and urges that the SEI further raise the software engineering discipline to the next level of refinement with as broad a scope as possible in the next five years. The SEI is encouraged to "go out on a limb" with respect to their vision, mission, and goals to remain at the leading edge. It is the unanimous recommendation of the Comprehensive Technical Review Panel that the SEI continue its role as an FFRDC.*

As we look ahead to the next five years, I believe that our mission is more critical than ever .In a recent article in Information Week, columnist Leon Kappelman writes about the escalating consequences of poor-quality software and cites the SEI as a beacon of hope. So, although we are proud of our accomplishments, we are also challenged by the work remains to be done. As this issue of news@sei interactive makes clear, we continue to aggressively pursue our mission of improving the practice of software engineering.

Thanks for reading, and please keep letting us know how well we are meeting your needs by sending your comments and suggestions to news-editor@sei.cmu.edu.

Stephen E. Cross
Director, Software Engineering Institute

Attribute-Based Architectural Styles

Rick Kazman, Mark Klein

Attribute-based architectural styles (ABASs) are architecture patterns that can be used as primitives for designing and analyzing software architectures. ABASs build on architectural styles by explicitly associating a reasoning framework (qualitative or quantitative) with a style. ABASs allow an architect to reuse the collected wisdom of the architecture design community in much the same way that object-oriented design patterns have given novice designers access to a vast array of experience in the object-oriented design community.

Software architecture is a key artifact in the development of complex software-intensive systems. If you are reading this column, presumably you already agree with this statement. To realize the benefits of architectures, they must be designed so that the resulting systems behave predictably. But there is no real discipline of architectural design today—it is a "black art," left to a select number of gurus. This notion of design as art is anathema to the creation of an engineering discipline for software. To try to address this gaping hole in the practice of architectural design, we have been creating attribute-based architectural styles (ABASs).

ABASs are architecture patterns that can be used as building blocks for designing and analyzing software architectures. ABASs build upon the notion of an architectural style. An architectural style is a description of components, connectors (relations among components), topology (arrangement of components and connectors), and some constraints on their interaction.

The main addition of ABASs to architectural styles is that they add a foundation for precise and efficient reasoning about specific quality-attribute goals. They do this by explicitly associating a reasoning framework with an architectural style. This reasoning framework may be quantitatively grounded (such as Rate Monotonic Analysis or queuing theory or metrics) or it may be qualitative in nature (such as checklists, questionnaires, or scenario-based analysis).

The reasoning framework shows how to reason about the design decisions comprised by the style. These reasoning frameworks are based on models that are quality-attribute specific (such as performance, reliability, and modifiability models), which exist in the various quality-attribute communities. In some cases these models are mathematical in nature and in other cases they simply capture a designer's experience in the form of standard questions or design heuristics. For example, adding reasoning based on http://www.sei.cmu.edu/publications/documents/91.reports/91.tr.006.html. Rate Monotonic Analysis to the pipe-and-filter style allowed us to create the Performance Concurrent Pipelines ABAS, which supports the designer in quantitatively reasoning about worst-case latency. Similarly, adding scenario-based

reasoning using the Software Architecture Analysis Method (http://www.sei.cmu.edu/publications/articles/saam-metho-propert-sas.html) allowed us to create the Modifiability Layering ABAS, which supports the designer in reasoning about the effects of changes on the modifiability and maintainability of the system. (For more on Attribute-Based Architectural Styles, and these particular ABASs, please see the SEI's ABAS Web site (http://www.sei.cmu.edu/ata/abas.html) and the current collection of ABASs (http://www.sei.cmu.edu/ata/abas_collection.html).

ABASs are similar in spirit to object-oriented design patterns, which document recurring solutions to recurring problems in object-oriented design. ABASs differ from these patterns in two ways: (1) they differ in scope—ABASs are typically broader in scope, and one would expect to find a small number of ABASs describing the complete architecture for a large, complex system; and (2) they add an explicit reasoning framework to the design, so that the implications of using a pattern can be rigorously explored before committing to it.

ABASs are powerful because they provide a designer or analyst with the concentrated wisdom of many preceding designers faced with similar problems. ABASs promote a disciplined way of doing design and analysis of software architecture based on reusing known patterns of software components with predictable properties.

The Parts of an ABAS

We are currently developing a handbook of ABASs. An early version of this handbook can be found at http://www.sei.cmu.edu/publications/documents/99.reports/99tr022/99tr022abstract.html

Part of the process of creating the handbook was to decide upon a consistent way of presenting and reasoning about an ABAS. An ABAS is documented as follows:

1. Problem Description

This is a description of the application of this technique in terms of real-world problems that the ABAS helps to reason about and solve. This section includes a description of the criteria for choosing the ABAS, including when the ABAS is appropriate and the assumptions underlying the reasoning framework. For example, a performance ABAS might only be appropriate for calculating worst-case latency but not average-case latency.

2. Architectural Style

This will describe the architectural style in terms of its components, connectors, topology, and constraints. It will also describe known extensions to the style. For example, we

might describe a strictly layered architectural style where a component in a layer can only access a component in a lower layer, and then discuss extensions to the style where a component can access any adjacent layer, whether above or below, or can access non-adjacent layers.

3. Analysis

This section presents the attribute-specific analysis model (the reasoning framework). A single architectural style may result in several ABASs, because ABASs are quality-attribute specific. So one might have a Security Client-Server ABAS, a Modifiability Client-Server ABAS, a Performance Client-Server ABAS, and so forth. Each will have the same style but a different analysis section. The analysis section will describe the key stimuli to the style, the responses that the analysis allows one to predict, and the architectural properties assumed to be provided as input to the analysis. This section will also include a set of analysis and design heuristics (a set of questions and guidelines that aid the architect in understanding how to achieve the desired response, concentrating particularly on how to manipulate the architectural parameters), and a set of extrapolations (typical ways in which this ABAS is extended relating to the choice of architectural parameters).

4. Examples

In this section we present the ABAS in action, showing how a sample architecture is described, reasoned about, and analyzed. This section may include several iterations of the analysis, showing how the reasoning allows one to discover and deal with design problems in an effective, disciplined way.

5. Related ABASs

This section presents a set of ABASs that either use the same topology, but with a focus on a different quality attribute (e.g., Security Client-Server ABAS vs. Performance Client-Server), or which aid in reasoning about the same response, or a closely related response (e.g., average-case latency vs. worst-case latency).

6. Tradeoffs

In this section we detail the known, frequently seen tradeoff points experienced when dealing with this ABAS. For example, layer bridging is a modifiability/performance tradeoff that frequently occurs in the layering ABAS.

7. Implementation Considerations

This section contains a discussion of the considerations and problems that are known to occur with this style in actual use. For example, layers can be realized in code through such means as coding and naming conventions and restrictions on access to interfaces and the internals that implement them. The engineer using a layering ABAS would have to choose one of these options to realize the ABAS in code.

8. Further Reading

Finally, we complete the ABAS by listing a set of references for the reasoning behind the ABAS, foundations of the analytic model, and so forth. We assume that the information in the ABAS will cover the majority of uses, but if the user has a particular need that goes beyond the presented analysis then he or she can consult the background references.

Designing with ABASs

Architectural design using ABASs is different from ordinary design. Rather than designing first from principles or by limiting reasoning to topology, we envision a process in which a designer considers the problem's constraints, its inherent topology, and the quality-attribute responses that must be controlled for the system to be successful. From these considerations, the designer can then look up appropriate ABASs from a handbook that is indexed by attribute responses and topology. The bulk of the design and analysis work is ready for the architect to reuse wholesale, and this, we believe, is an enormous benefit. We have already reaped this benefit in practice many times by using ABASs and ABAS-style reasoning in analyzing software architectures.

Architectural analysis and design, as it is currently practiced, is *ad hoc* in that it always starts from first principles. ABASs hold the promise of changing that, giving software engineers *proven* starting points and techniques for design and analysis. This holds the promise of making architectural design a true engineering discipline. We expect to publish a handbook of ABASs in 2001, including ABASs on performance, availability, security, usability, testability, and modifiability. We currently have about 15 ABASs in draft or final form and anticipate roughly double that number in the published version of the handbook.

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

Mark Klein is a senior member of the technical staff at the SEI, where he is a technical lead in the Architecture Tradeoff Analysis Initiative. He has more than 20 years of experience in research and technology transition on various facets of software engineering and real-time systems. Klein's most recent work focuses on the analysis of software architectures. His 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. He has co-authored many papers and is a co-author of the RMA Handbook, titled *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*.

Wheels Within Wheels: Model Problems in Practice

Kurt Wallnau

In my Spring 2000 *COTS Spot* column I explained the role of technology competence in the design of COTS-based systems, and described how to obtain this competence quickly by building toys and model solutions to model problems. In this article I will take the next step and describe how to put model problems into action by using the 3-Rs of design risk reduction in component-based systems: **R**ealize model solutions, **R**eflect on their utility and risk, and **R**epair the risks. I will then relate the 3-Rs--which is denoted typographically as R^3--to increasingly popular iterative software development processes.

The R^3 Process

The characteristics of COTS software components--what those components do and how they do it--can and do influence the design activity. This influence might be felt as early as in the conception phase of a software project. For example, it might be known that a component provides a capability that would be difficult or costly to implement as a custom solution. In this situation the component capability could very well wind up as a requirement for the system. In other words, the component capability would effectively contribute to defining the scope of a system, and the act of defining this scope would effectively lead to a de facto component selection decision. The same is often true later in the design activity, and quite often (I would go so far as to say *usually*) this involves not just the characteristics of single components but ensembles of components acting in concert to provide some service.

It is irrelevant whether we view the component as having scoped the system or the system as having defined requirements that led to the selection of a component: in either case competency about component capabilities is essential. Where there are gaps in our competence, and competence gaps are inevitable as the number of commercial components used in a system increases, there is risk. As I discussed in the May *COTS Spot*, toys and model problems can be used to generate this competence on the fly. But where does the idea for a particular model problem come from? How do we know which model problems to solve and, having solved them, what to do with the solutions? Answering these questions is what the R^3 process is all about. An outline of this process is depicted in Figure 1, which includes (naturally) three key steps:

1. *Realize* *a model solution.* R^3 begins with assumptions about system needs and a component ensemble that is believed to satisfy those needs. The designers sketch the workings of an ensemble, perhaps using component-interaction blackboards. Inevitably, questions will arise about how an ensemble works, or the manner in which the ensemble satisfies a need. If the need to be satisfied is critical, then it is essential to increase the level of understanding of the ensemble. The unknown property will itself suggest what

kind of toy to build; previous design commitments (for example, component selections) will constrain how the toy is built, and the needs will define the evaluation criteria.

2. *Reflect* *on the qualities of the model solution*. Model solutions are implementations that must be evaluated against criteria. Did the solution satisfy the criteria? Were additional evaluation criteria discovered that must be considered and, if so, how did the model solution stack up to these new criteria? (The discovery of new criteria usually heralds some sort of failure). Answering these questions may involve benchmarking, snooping, or other invasive "black box visibility" techniques. In any event, one of two possibilities arise from this reflection: the model solution passes muster, in which case it becomes part of the design baseline, or it fails in some way to satisfy the evaluation criteria. Failure is not necessarily fatal to an ensemble's prospects.

3. *Repair* *the ensemble*. It pays to be an optimist--or at least to be doggedly persistent--when developing COTS-based systems. Ensembles can be repaired by introducing new components, by using alternative components or component versions, by developing wrappers, or by any number of other strategies. Indeed, there are often several possible repair strategies for each deficiency detected. This has led us to develop evaluation techniques such as risk/misfit (the topic of a future column) to help structure component selection decisions that are dominated (or complicated) by the presence of multiple repair options. In any case, non-trivial repairs are hypotheses that must be tested, triggering a new iteration of R^3.

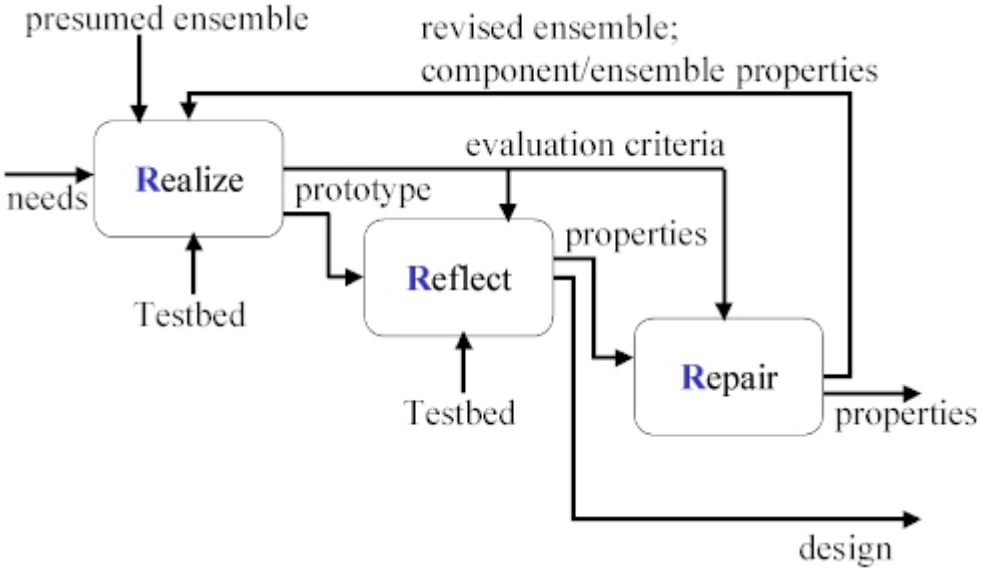

Figure 1: The R^3 Process for Design Risk Reduction

What happens if, despite all optimism and doggedness, an ensemble simply will not pass muster? In this case all is still not lost for the ensemble, but salvaging the situation may require a different repair strategy--one which involves altering the requirements that gave

rise to the ensemble, rather than changing the ensemble itself. But before we tackle this issue (which will involve us with the theory of iterative development), a practical illustration of the R^3 process from our own case book will be useful.

R^3 in Action

Once upon a time my colleagues (Scott Hissam and Robert Seacord) and I were helping a Defense Department program migrate a large, custom legacy system to an open, COTS-based system. The program had made an early commitment to a Web-based solution, and was considering whether to repackage legacy services as CORBA-based services and provide thin-client access to these services via Java applets running in Web browsers. Even three years ago this ensemble was known to be feasible (Web browser and server, CORBA servers, Java applets). The real question, however, was whether this ensemble could, in a practical way, be made to work securely.

To support identification, authentication, authorization, and confidentiality, the ensemble illustrated in Figure-2 was proposed. (You can guess the age of the example from the component version numbers--they were current releases at the time.) The Web server would identify and authenticate (I&A) users accessing the system from their Web browsers. An authenticated user would receive a CORBA interoperable object reference (IOR) from the Web server. This CORBA object would contain the user's permissions on the system for a particular session. Each operation performed by the user would be tagged with this session IOR, allowing back-end services to check a user's authorization to invoke a service. Using CORBA interceptors would allow us to do this session-tagging transparently to applet developers. Last, all interactions between the requesting applet and back-end services would be transmitted using the CORBA IIOP protocol over a secure socket layer (SSL) connection, thus guaranteeing confidentiality.

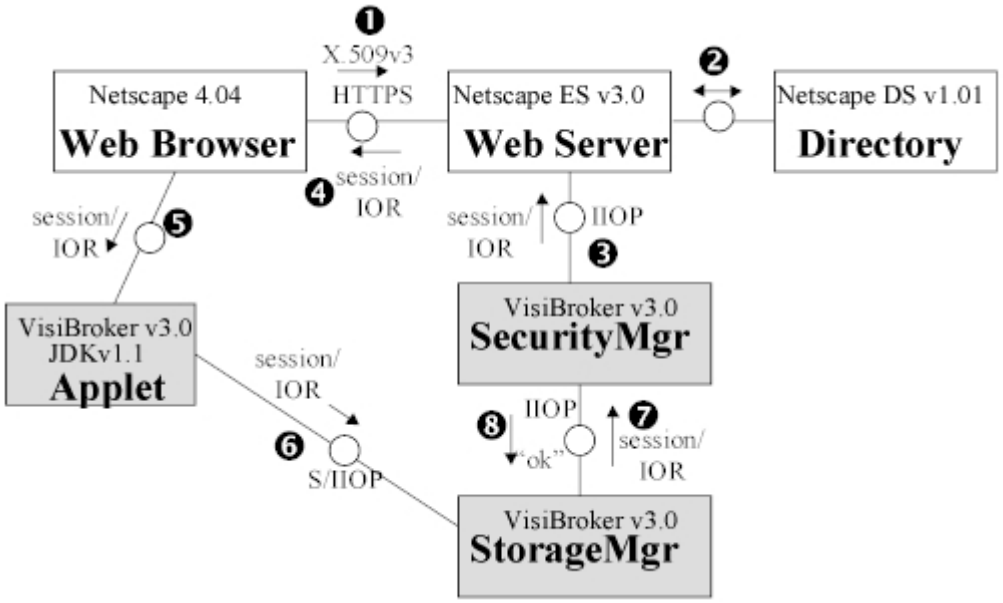

Figure 2: A Secure Web Ensemble

Recall that model problems are toys with design context and evaluation criteria. In this ensemble the design context included all of the concrete component versions in the proposed ensemble. The evaluation criteria were simple. First, it had to work. That is, the sequence of steps numbered 1-5 should transpire once when an applet is downloaded to the browser and steps 6-8 whenever the applet requested a service. Second, we wanted the security services for authorization and confidentiality (all of the interactions among the shaded components) to use the same digital certificate information used by Netscape to do its identification and authentication in steps 1-2. This was important because we did not want to have duplicate security infrastructures, and we did not want to add any more client-side administrative burdens beyond that which was already required to use a browser with digital certificates.

The toy we constructed was as simple as we could make it, but despite our best efforts we still had to do a significant amount of component configuration. For example, setting up the certificate-management infrastructure required that we develop a security policy for our toy that addressed matters such as who is allowed to issue certificates and whose certificates are trusted. As with all good toys, the amount of application-level code was minimal--the security manager and storage manager involved only a few dozen lines of code, and the applet and server-side JavaScript needed to generate the applet were also quite small. As small as it was, the toy was sufficient to test against the evaluation criteria.

What was the result? Problems arose when we tried to get the applet to establish an SSL connection for secure IIOP between the applet and storage manager. VisiBroker required the applet to supply a private key to configure the client side of the SSL connection. Although VisiBroker would accept the private key portion of the public/private key found in the user's digital certificate--the same certificate used to authenticate the user in steps 1 and 2 of Figure 2--there was no way to programmatically retrieve this key from the Netscape browser. We suspected that this was because Netscape feared that if they did so they would be in violation of export control laws concerning key management technology. Regardless of the reason, the end result was that the ensemble provided all of the required security attributes except confidentiality.

This did not mean, however, that we were prepared to abandon the secure Web ensemble. We had a fallback position of course (we always do), based on simple use of Netscape's server-side JavaScript in lieu of applets (Java servlets were not yet available). But as a rule we try wherever possible to avoid relying on vendor- or component-specific languages, as these are liable to change causing later maintenance headaches (to say nothing of vendor lock). The secure Web ensemble was based on de facto industry standards--HTTP, Java, CORBA, X.509--and we thought there were better long-term prospects for this design approach even if there were near-term risks. After some reflection, we hit upon two possible repair strategies. One strategy involved trying to discover how Netscape did its key management on the browser side so that we could impose our own application-programming interface to extract the private key. Another repair strategy was to achieve confidentiality through a hardware encryption with a network interface card.

Both of these repair options became design contingencies. Both of these contingencies required further investigation including, ultimately, the development of model problems to validate their feasibility. Readers interested in discovering the results of these investigations might wish to read *Into the Black Box: A Case Study in Obtaining Visibility into Commercial Software* and *COTS in the Real World: A Case Study in Risk Discovery and Repair*. But the real point of the illustration is not to provide the specific technical details of the model problems (most of which have been suppressed), but rather to reinforce the message that design risk reduction in COTS-based systems involves very detailed, implementation-level investigations. Any area of uncertainty in how a component or ensemble works to satisfy a key need is an area of risk that must be resolved as quickly as possible.

In this illustration we were able to expose the design risk and find a workaround based on our discovery about how Netscape manages its public and private keys. As I mentioned already, we had a fallback in case the ensemble would fail. Sometimes, however, a fallback is not available, especially if, as is frequently the case, system requirements were influenced by the perceived capabilities of an ensemble. What happens if these perceived

capabilities turn out to fall short of the mark? This brings us to the topic of iterative development, and "wheels within wheels."

R^3 and Iterative Development

Sometimes an ensemble will not work, no matter how hard you try to repair it, or just as likely, the repairs prove to be too expensive to justify. You can see from Figure 1 that the R^3 process yields two results: a design (or, rather, the ensemble's portion of a design), and the properties of that ensemble. If we assume that an ensemble cannot be repaired and there are no fallback positions, then there is only one thing left to do: reassess the original needs. In some cases, Mohammed must go to the mountain. But this involves us in another level of process iteration beyond that of R^3. Thus, we assume to a large extent that R^3 works within an iterative development process, and that it is, in effect, a small wheel within a larger wheel.

This should not be surprising, and I suspect that anyone who has read this far is already familiar with, if not comfortably experienced with, iterative software development processes. But be on guard: COTS software requires a form of process iteration that is unique to COTS, and is not to be found in the development of custom systems. To appreciate this dire warning we examine the currently popular Rational Unified Process (RUP), an iterative software development process championed by Rational Software. RUP is depicted in Figure 3. Please be aware that the discussion that follows is not a critique or criticism of RUP, but rather an elaboration of how a naïve interpretation of RUP can cause trouble in systems that are COTS-software intensive.

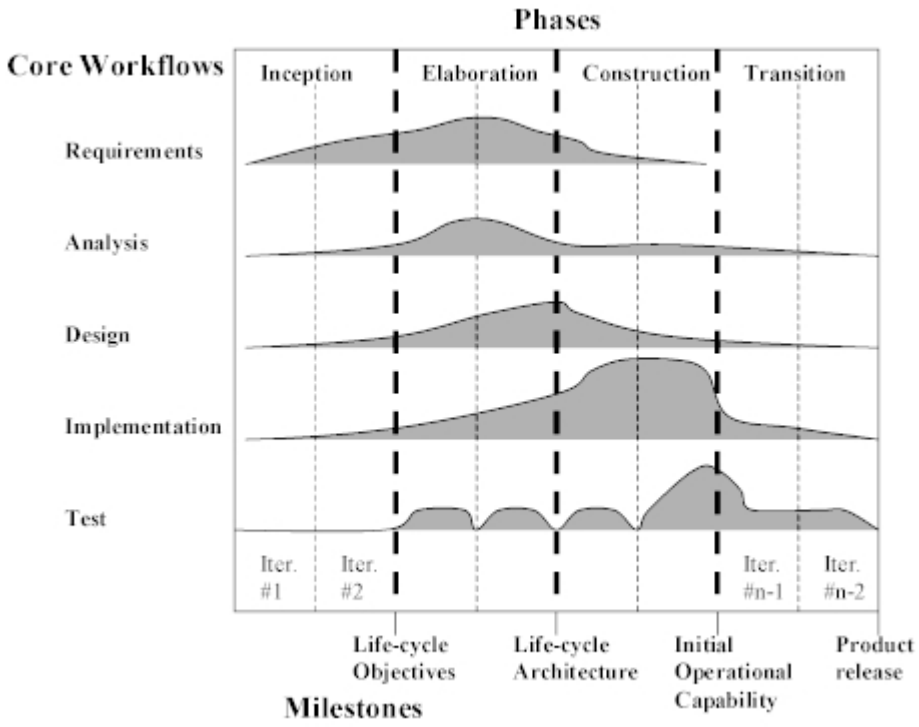

Figure 3: The Rational Unified Process

One of the key ideas in RUP is that an iterative development process is partitioned into four discrete phases: inception, elaboration, construction, and transition. This partitioning is quite useful because it answers one difficult question posed by iterative development: where are we going? Because each phase has its own milestones it is possible to focus each iteration in a constructive way. So, for example, the purpose of the inception phase is to produce the life-cycle objectives milestone, which describes the scope of a system, its business case, and how the most major risks were mitigated (among other things). The elaboration phase ends with a description of the system architecture and mitigation of second-tier risks.

Note, however, that in RUP the inception phase terminates with a definition of system scope, while the elaboration phase terminates with a definition of system architecture. But we know from our own experience that the selection of COTS software components and their assembly into ensembles--key architecture-defining activities in COTS-based systems--can greatly influence how we define the scope of a system. Indeed, in the R^3 illustration we showed how an ensemble failure might require an adjustment of system requirements (i.e., system scope) in order to effect a repair.

When I mentioned that COTS requires a special form of iteration, this is what I was referring to. Put another way, decisions about system scope and component and ensemble selection are co-dependent: a designer must often make tentative decisions about system scope in light of what is thought to be known about COTS component capabilities, and

then revisit those scope decisions as more is learned about the components. Conversely, component and ensemble selection decisions may be informed by system requirements, but those selection decisions may be revisited as more is learned about the requirements.

On the surface this co-dependence of component and ensemble selection decisions and system scope decisions seems to be a problem, because RUP allows no iteration among life-cycle phases--that is, elaboration begins with a defined system scope. However, the folks at Rational Software are smart, and the ideas underlying RUP are sufficiently sound and flexible to accommodate the development of systems that are COTS-software intensive. There are at least two ways to do this:

1. You will observe that in Figure 3 the inception phase includes some design and implementation effort. This effort refers to risk-reduction prototyping as needed to scope the system and identify and mitigate the most serious risks. Given this, it might be useful to concentrate prototyping efforts in the inception phase to support component and ensemble selection decisions as a necessary adjunct to defining life-cycle objectives.

2. Requirements and analysis work continues (and in fact accelerates) in the elaboration phase, even after the basic scope of the system has been defined. Given this, it may be possible to consider the life-cycle objectives as a flexible rather than rigid milestone. This would allow the discovery of COTS component capabilities and liabilities to change the scope of a system, even if the life-cycle objectives milestone has been satisfied.

Option 1 is fine for projects that do not use state-of-the-art COTS components. Projects that use cutting-edge components must make a conscious tradeoff between advanced capabilities and component stability. In these situations the designer may be required to manage multiple design contingencies (system scope and architecture) far into the development process. Option 2 is fine if the stakeholders are willing to be flexible about their requirements and can tolerate some degree of uncertainty about some very fundamental decisions--what the system will do and which COTS components will be used to do it.

The best solution may be to combine elements of both options. That is, do as much risk reduction as possible during system inception, where more stable COTS components are used, and then work with the stakeholders to manage their expectations about system capabilities that depend upon more cutting-edge but unstable components during system elaboration. In either case the iterative R^3 process for obtaining component competence and proving the feasibility of component ensembles will be useful in identifying and resolving risks related to the use of COTS software.

I've mentioned several times that the designer may need to manage multiple design contingencies--specifically, contingencies that reflect ensemble repair options. This presents the designer with a variety of new challenges. What is the design of a system at

any point in time if there are multiple contingencies being explored? How much should a project invest in the "just-in-time competency" attending the exploration of any particular contingency? And, at what point should this investment be terminated and a contingency foreclosed? I will take up these issues of "contingency management" in the next issue of *The COTS Spot*.

About the Author

Kurt Wallnau is a senior member of the technical staff in the Dynamic Systems Program at the SEI, where he is co-lead of the COTS-Based Systems Initiative. Before that he was a project member in the SEI Computer-Aided Software Engineering (CASE) integration project. Prior to coming to the SEI, Wallnau was the Lockheed Martin system architect for the Air Force CARDS program, a project focused on "reusing" COTS software and standard architectures.

Cybersleuthing: Means, Motive, and Opportunity

Larry Rogers

Editor's Note: This column originally appeared in the June 2000 issue of InfoSec Outlook, a joint publication of the Information Technology Association of America (http://www.itaa.org) and the CERT® Coordination Center (http://www.cert.org) at the Software Engineering Institute.

We've all seen television police dramas where the detectives nab the criminal by determining who has the means, a motive, and the opportunity to commit a crime. They ask questions such as "Did the suspect have the means to commit the crime? Did they have something to gain? Did they have the opportunity to carry out the crime?" We can view trends in cyber attacks using the same three categories: means, motive, and opportunity.

Means

To commit an Internet-based crime, intruders need either personal expertise or the tools that are freely available through the Internet. It is the sum of the two that defines the means to do the job at hand.

The means for attacking computer systems has changed over the years. Ten years ago, intruders attacked computer systems primarily "by hand." For example, they tried to guess passwords by brute force techniques such as repeatedly trying to login to an account by using a dictionary of passwords. They also used social engineering methods to trick people into revealing passwords. Today, there are tools that encrypt dictionary words and their variations (such as replacing the letter "o" with the digit "0") and compare them to the encrypted password.

The level of sophistication of intrusion tools has become high and is getting higher. Intruders have harnessed the power of the Internet itself, building automated tools to coordinate large-scale attacks involving hundreds of hosts aimed at Internet sites. These tools are well documented and freely available on the Internet. Members of the intruder community share programs and improve on each other's work.

Sophisticated tools have given birth to a class of script kiddies, intruders who use tools to break into computer systems although they lack the knowledge to craft the tools themselves or to even understand the nuances of their inner workings. There have been reports of break-ins where the script kiddies used a sophisticated tool to gain access to one operating system but then typed commands that work only on another operating system.

Motive

Motives for computer attacks have evolved just as the means have. In the early years of the Internet (then called the ARPAnet), there were no .com sites, only government and university research information. In 1981 only 213 computers were connected to the Internet. The small network made it easy for researchers at diverse locations to cooperate on work to their mutual benefit. There was a collegial atmosphere of sharing among people who either knew each other or knew of each other.

Contrast that to today's Internet. The January 2000 Internet Domain Survey reports that .com sites make up more than one-third of the Internet, which has now passed the 72-million computers mark. You can find nearly everything on the Internet today: proprietary information about companies and people, corporate strategic plans, access to financial resources, and most commercial products.

Computer power has increased from the days of the VAX-11/780 with its 1 MIPS (million instructions per second) processing power, to 1Ghz (gigahertz) Pentium III processors, an increase of more than 800%. As a result, attackers can steal computer cycles without the knowledge of the computer owner.

Long gone are the days of users and administrators knowing and trusting each other. Users on the Internet are anonymous, and their number grows daily. The atmosphere is not collegial, and trust is neither automatic nor always warranted.

Opportunities

Opportunities for computer attacks are readily available for two reasons: the number of vulnerable systems on the Internet and the ease of connecting to the Internet. Ten years ago, there were about 300,000 hosts on what was then the ARPAnet; today there are more than 72 million. Even if the same percentage of vulnerable hosts exists, that's nearly 25,000% more vulnerable hosts today.

The number of computers on the Internet and the difficulty of configuring them securely means that attackers have more chances of finding a way into systems than they did a decade ago. Along with low-cost Internet access, computers are inexpensive and the price is dropping. This means that more attackers can afford both the computer and the Internet access needed for an attack.

Also, there are many more opportunities for computer access. Some libraries provide free Internet access. My children’s school invites family members to use its computing facilities one evening a week. These Internet access points are a convenience and a helpful service, but they are also an opportunity to commit a crime, and are readily available to anyone so inclined.

Who Wants to be a Millionaire?

Whatever the motive—money, curiosity, politics, power—this is all it takes to commit a crime on the Internet:

- Means—the tools are there, nicely catalogued and ready to go.
- Motives—with so much on the Internet, motives are there.
- Opportunity—there are many, many access points to the Internet. Most are inexpensive, some are free.

Intrusions are going to happen; it's inevitable. Administrators, their managers, and senior executives all need to know what they're up against so that they are better equipped to deal with attacks and be aware of what intruders are doing. Because attack techniques and tools are constantly changing, we must maintain constant vigilance.

About the Author

Larry Rogers is a senior member of the technical staff in the Networked Systems Survivability Program at the Software Engineering Institute. The CERT® Coordination Center is a part of this program. Rogers' 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. Before joining the SEI, Rogers worked for ten years at Princeton University, first in the Department of Computer Science on the Massive Memory Machine project, and later at the Department of Computing and Information Technology (CIT). 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.

Moving the Goal Posts

Watts S. Humphrey

In this column, I talk about the nature of process improvement and why it is such a dynamic and challenging field. The future will be much like the past in many respects, but it will also be very different. However, as we look ahead, there are some reasonably reliable guides that can help us to address the problems we will face.

Brownian Motion

Just about every time I visit an engineering organization, the people tell me, "We're different." Of course they are. We are all different, but what is surprising is how often truly different organizations behave in the same way. However, it is also surprising how often seemingly similar organizations, when faced with nearly identical conditions, behave quite differently. People are both predictable and unpredictable. Much like Brownian motion in physics, there is no way to precisely predict individual behavior. However, on average, overall behavior is highly predictable. So, what does this mean for process improvement? Essentially, the following:

- First, there is no one best way.
- Second, every situation is different. Each solution must consider the people and their backgrounds, beliefs, and circumstances.
- Third, the lessons of the past are the only practical guides we have for the future. While we cannot predict precisely what will work in any specific case, we can establish highly reliable general guidelines.
- Fourth, the principles behind the quality movement are just as sound today as they were in the past. Those who argue that the new Internet age changes all the old truths will continue reliving the same history that many of us have painfully survived.

What these lessons tell us is that a single-minded approach to solving any human problem will almost certainly be wrong, if not for everybody, at least in many cases. There is no single best answer. People are extraordinarily creative, both in the ways that they solve problems and in how they create problems. Therefore, we must recognize that problems will change, and we must continually seek newer and better ways to address the problems that we face at each point in time.

When the Problems Change, the Solutions Must Also Change

The other day, I read the following newspaper headline: "The quality of U.S. automobiles lags behind Japan and Europe." As Yogi Berra once said, this is "déjà vu all over again." After 20-plus years, can quality still be a problem for Detroit? It almost certainly is, and the best way to tell is that the General Motors board of directors cut executive bonuses. That is a guaranteed way to get management's attention.

GM, Ford, and Chrysler have been working on quality improvement for more than 20 years, but they still have about 150 defects per 100 new cars. However, unlike 20 years ago, these are not primarily manufacturing defects. Most are design problems. Detroit solved the quality problems of 20 years ago, and if the Japanese had not kept moving the goal posts, Detroit would be in fat city. But the world did change, and Detroit is still dead last in the quality sweepstakes.

The world changes, and it does not change all by itself. Everything we do changes it. In another lesson from physics, Heisenberg showed that you can know a particle's location or its velocity but not both. When you measure one, you change the other. People are just like that. As soon as you fix the process, the problem changes. Does that mean that we should give up? Not at all; it just means that we cannot relax. We must keep thinking, and resist the temptation to blindly rely on the solutions and formulas of the past. Continue to follow the same principles, certainly, but don't blindly follow the same path. Sooner or later it will lead to a dead end.

Finding the Goal Posts

While process problems are often unique, they all stem from human failings, and these are common to all of us. Because the same human failings have persisted through the ages, we cannot expect to eliminate them. The process improvement challenge is to devise ways to live with and compensate for normal human behavior. We must recognize, however, that soon after we compensate for a given set of failings, human nature will find creative countermeasures. So, in spite of all our efforts, the battle for improvement will continue indefinitely. Hopefully, however, technology will keep improving and each step will move the goal posts a little further down the field.

Human Failings

While software professionals are marvelously creative and highly energetic, we sometimes feel lazy or want to take a break. We are also a race of procrastinators, and when we can't avoid or put off some difficult or unpleasant task, we try to replace it with an easier task or get someone else to do it. If we find that we still must do the job, we tend to do it as quickly and superficially as we can get away with. This means that for

every complex and difficult task, the process improvement challenge is to devise ways to get people to consistently do their work in a highly professional way.

What makes this so challenging is that once we figure out some way to do this, it is only a matter of time before people devise a clever way around our fancy new process. Take estimating, for example. The Capability Maturity Model® (CMM) calls for engineers to be involved in and agree with the project estimate. However, soon after an organization puts a new planning procedure in place, some group will almost certainly find an estimating method that uses expert estimators or complex and arcane tools. Experts will then make the estimates and the engineers won't be involved. Even though this destroys the intent of the planning process, unless processes are defined very carefully, people will adopt new practices that conform to the letter of the defined process but not to its intent.

What this Means for Process Improvement

What this means for you and me is that process improvement must not be directed at only the process. The principal objective must be to change human behavior. However, to change human behavior, we must consider and compensate for normal human failings.

For example, we now find that even in CMM Level 5 organizations, people have learned to compensate for their new processes. In some of the Level 5 organizations I have visited, the measurement and process analysis work is handled by the process and quality groups, and the engineers continue to work essentially as they did at Level 1. This totally misses the point of CMM Levels 4 and 5, which is to have engineers *use* data, not just gather and report it. This implies that even the goal posts defined by the CMM levels must be moved to keep pace with our rapidly changing technology.

In the last analysis, to improve engineering performance, organizations must change the behavior of the engineers and their managers. If you find that some change has stopped producing the desired results, find out why and then devise another improvement to solve the new problems.

The Implications for the Future

We have made great strides in the last 10 or more years, and we must continue to build on our successes. However, the goal posts are moving, and the problems we will face in the future will almost certainly be different from those of the past. Think of it this way: You could build a 10-foot boat in your garage, but a 1,000-foot ship would require entirely different tools, technologies, and processes. Similarly, in transportation, going from 3 to 300 miles per hour requires several changes in technology. In the software business, we think nothing of factors of 100. We use the same tools, methods, and processes for a program with 10,000 lines of code (LOC) as we do for a 1,000,000-LOC programming system.

Our ability to master the software-intensive technologies of the future will be largely guided by the ability of engineering teams to match their behavior to the more demanding tasks they will face. We cannot expect that our current tools, technologies, and processes will be adequate in the future. The challenges will keep increasing, and we must continually evolve our methods to keep pace. We must think of process improvement in multi-dimensional terms and include the educational system, as well as industry. An informed customer community will also be important, and we must consider all levels of the engineering organization: executives, managers, teams, and engineers. Much as in the automobile industry, we must retain the solutions of the past, but we must broaden our perspective to consider all the relevant aspects of the problem.

While it is always risky to predict the future, some trends are now pretty obvious:

- Systems will get larger, more complex, and more integrated.
- Engineering teams must also become more highly integrated.
- Compatibility, reliability, usability, privacy, and security will be increasingly important.
- While schedules must be as short as possible, they must be absolutely reliable.
- The quality of every engineer's personal work will be even more important than it has been in the past.

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 Noopur Davis, Jim McHale, Don Mcandrews, Julia Mullaney, 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 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 in 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.

Evaluating Risks in COTS Acquisition Programs

Bob Lang

The importance of managing risk is well understood in the software engineering community. Department of Defense (DoD) directives and mandates, such as DoD 5000.1 and 5000.2m, specify the use of risk reduction activities. And the SEI's Software Risk Evaluation (SRE[1]) has been a significant part of acquisition for several years.

In an acquisition that will include extensive use of commercial off-the-shelf (COTS) products, several problems emerge that are not present in non-COTS-intensive acquisitions. For example, the requirements process must become more flexible, yielding to the realities of commercial products, such as the inability to control when products are released, their features, and their ability to interface with other products. Such problems contribute to a program manager's loss of control, and hence, added risk. To help managers manage these risks, the SEI has developed a COTS Usage Risk Evaluation (CURE[SM]).

About CURE

CURE is a risk evaluation and mitigation approach aimed specifically at COTS-related issues in acquisition. CURE is most useful as a "front-end" analysis tool for predicting the areas where the effect of COTS components will be most prominent in a program. It can also be used as an overall improvement tool for organizations wishing to become more aware of COTS-related risk areas.

CURE consists of a detailed questionnaire completed by personnel in the program being evaluated and an evaluation report prepared by the SEI about the program. In addition, organizations can request optional follow-up visits to reassess high-priority risks, assess impact of program changes, identify new areas of risk.

"CURE came about, in part," says David Carney of the SEI, "because of our experience on 'red teams' for programs that were in trouble, and that were making heavy use of COTS components. We were going in and asking investigative questions to help find the source of the troubles in these programs. And we were asking the same questions over and over. So we distilled these questions into the questionnaire."

The questionnaire is used during on-site visits to an acquiring organization, a system developer/integrator, or both. One month before an on-site visit, the SEI sends the

---

[1] For more information about SRE, see http://www.sei.cmu.edu/topics/products/services/sw..risk.eval-service.html

organization the 35-page questionnaire with questions targeted at decision makers—the program manager, lead architect, and system engineer—within a specific program or project. The questionnaire is intended to gather information about past programs as well as the current program.

Next, the evaluation team conducts on-site interviews with program personnel to analyze key topics in greater depth and discuss additional topics. "This is where we do the digging," says Carney. "The questionnaire includes a list of topics related to each question. This is to let the organization know the kinds of things we'll be looking for." See the sidebar accompanying this article for an example questionnaire item and the list of topics related to that item.

After the on site visit, the evaluation team produces an evaluation report for the organization. The report focuses on the COTS related risks in the program and suggests mitigations to those risks.

For each serious risk identified by the evaluation team, the report includes

- a concise statement of the risk and a detailed review of the team's analysis
- the sources of information that led to the analysis (for example, specific answers in the questionnaire or the portion of the interview)
- the team's assessment of the criticality of the risk
- some potential mitigations to the risk

CURE Questionnaire

Below is an excerpt from the CURE questionnaire. Note that in addition to seeking written responses, the questionnaire prompts interviewees to think about related topics that the SEI evaluation team will be probing in more detail during the site visit.

- Describe the degree to which [the system] will require program-specific tailoring, extensions, or enhancements to COTS products.
- Identify aspects of the system's design that are (or are expected to be) dependent on modifications to specific COTS products.

Potential discussion topics for on site interviews:

- Strategy for product replacement when necessary
- Decision factors indicating that modification is necessary
- How the extent of modification was determined
- Estimates for the cost and schedule of modifications

What CURE Is Not

Other risk management tools, such as the SEI's SRE method, emphasize the need for risk management as a routine way of doing business; use of SRE can be an important catalyst for bringing about in an organization a philosophical change in attitude toward risks.

CURE, by contrast, is diagnostic, and specific to a given program. It is not aimed at bringing a philosophical change, but at fixing a program before it gets out of control.

It is also important to note that the evaluation is not product oriented. "We don't come in and say, 'this is a good product' or, 'this is a good testing tool,' " says Carney. "We don't talk about specific products; we are process oriented." So rather than evaluating, for example, a specific testing approach and making a judgement about whether the organization should be using that approach, a CURE team would look for evidence that the approach has been used successfully in the past, asking questions such as, " What kind of program has this approach been used on? Was the program successful? Who else is using this approach?" Similarly, the questions that team might ask about specific products would be: "What is the evidence that this product behaves as advertised? What is the evidence that your contractor understands the product and has used it before?"

Results and Transition

CURE has been used successfully in several contracting and acquiring organizations, and in one case, both the contractor and acquirer on the same program. The SEI is currently working with one DoD agency on transition of CURE to the community and is seeking other organizations for a similar partnership.

For more information contact-

Customer Relations
Phone
412 / 268-5800

David Carney
Email
djc@sei.cmu.edu

Word Wide Web
http://www.sei.cmu.edu/cbs/products.html#CURE

Identifying Quality Attributes

Bill Thomas

In large software systems, the achievement of qualities such as performance, security, and modifiability is dependent not only on code-level practices but also on the overall software architecture. Thus, it is in developers' best interests to determine, at the time a system's software architecture is specified, whether the system will have the desired qualities.

With the sponsorship of the U.S. Coast Guard's Deepwater Acquisition Project, the SEI is testing the concept of a "Quality Attribute Workshop" in which system stakeholders focus on the discussion and evaluation of system requirements and quality attributes. The goal of the Deepwater Project is to create a system of systems, using commercial and military technologies and innovation to develop a completely integrated, multi-mission, and highly flexible system of assets-including cutters, patrol boats, and short-, medium- and long-range aircraft-at the lowest total ownership cost. The project is the largest and most comprehensive recapitalization effort in Coast Guard history.

Workshop Overview

The purpose of a Quality Attribute Workshop is to identify scenarios from the point of view of a diverse group of stakeholders and to identify risks and possible mitigation strategies. Scenarios are used to "exercise" the architecture against current and future situations, and include the following types:

- Use-case scenarios reflect the normal state or operation of the system. If the system is yet to be built, these would be about the initial release.

- Growth scenarios are anticipated changes to the system. These can be about the execution environment (e.g., double the message traffic) or about the development environment (e.g., change message format shown on operator console).

- Exploratory scenarios are extreme changes to the system. These changes are not necessarily anticipated or even desirable situations. Exploratory scenarios are used to explore the boundaries of the architecture (e.g., message traffic grows 100 times, operating system is replaced).

The stakeholders—including architects, developers, users, maintainers, and others—generate, prioritize, and analyze the scenarios, and identify tradeoffs and risks from their points of view, depending on the role they play in the development of the system, and their expertise on specific quality attributes. Together, the scenarios, risks and mitigations strategies are used as input to the architecture developers.

Quality Attribute Roadmap

Figure 1 illustrates the Quality Attribute Roadmap, the process used during the workshops to discover and document quality attribute risks and tradeoffs in the architecture.

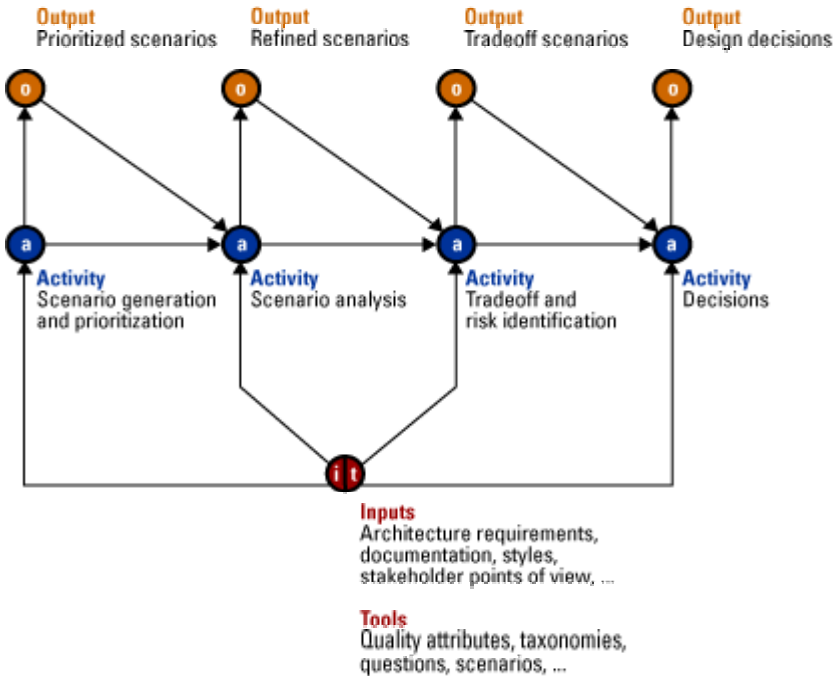

Figure 1: Quality Attribute Roadmap

During the workshop, participants engage in several activities aimed at generating various outputs or products:

- Scenario generation takes place during a facilitated brainstorming process; stakeholders propose scenarios that test the effectiveness of a candidate or conceptual architecture to achieve specific quality attributes within a specific Deepwater mission and geographic context.

- During scenario analysis, for each of the high-priority scenarios, the stakeholders choose an appropriate architectural fragment as an artifact for analysis, and apply the scenario to the artifact. The purpose of the analysis is to identify important architecture decisions and sensitivity points. As a result of this activity, the stakeholders might decide to conduct additional, more detailed or formal analyses of the scenarios or artifacts, but these analyses take place "off line," not during the workshop.

- During tradeoff and risk identification, the stakeholders use the results of the analysis activity to identify and document risks-i.e., potential future problems that might affect the cost, schedule, or quality attributes of the system. Various sources are used as inputs for the activities, including architecture documentation, stakeholder points of view, and architecture styles.

"The Architect" column in the Spring 2000 issue of news@sei interactive provides additional information about Quality Attribute Workshops:
http://interactive.sei.cmu.edu/news@sei/columns/the_architect/2000/spring/the_architect.htm

The Coast Guard Web site provides additional information about the Deepwater Project:
http://www.uscg.mil/hq/g-a/Deepwater

For additional information about Quality Attribute Workshops contact-

Mario Barbacci
Email
mrb@sei.cmu.edu

Customer Relations
Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

Questions for Collecting and Analyzing Information

Various types of questions are used to collect and analyze information about current and future system drivers and architectural solutions. The types of questions fall into the following categories:

Screening questions are used to quickly narrow or focus the scope of the evaluation. They identify what is important to the stakeholders. Screening questions are qualitative; the answers are not necessarily precise or quantifiable. The emphasis is on expediency. If the quality attribute of concern was security, an example screening question might be: "What are the trusted entities in the system and how do they communicate?"

Elicitation questions are used to gather information to be analyzed later. They identify how a quality attribute or a service is achieved by the system. Elicitation questions collect information about decisions made; the emphasis is on extracting quantifiable data. Elicitation questions for security might be: "What sensitive information must be protected? What approach is used to protect that data?"

Analysis questions are used to conduct analysis using attribute models and information collected by elicitation questions. Analysis questions refine the information gathered by elicitation. An analysis question for security might be: "Which essential services could be significantly affected by an attack?"

CMMI: Getting to Version 1.0

Bob Lang

Because of the existence of several maturity models, organizations undergoing process improvement efforts often encounter the problem of deciding which model to choose, or how to interpret differences in terminology or guidance.

The Capability Maturity Model® (CMM®) Integration effort is focused on eliminating these difficulties. The goal is to improve efficiency, return on investment, and effectiveness by using a single model that integrates disciplines such as systems engineering and software engineering, which are inseparable in a systems development endeavor.

The CMMI project team is a collection of government, industry, and SEI participants. Since the inception of the CMMI project, user and stakeholder feedback has played a key role in the definition and evolution of the model. In particular, since September 1999, the CMMI team has been continuously gathering feedback and improving the model in preparation for the release of version 1.0 later this year. This process of "getting to" version 1.0 was an essential part of building a robust, useful model.

Gathering Feedback

In the early development of the model, the team relied on a set of stakeholders already intimately involved in the development and use of one or both of the source models from which CMMI is derived—the CMM for Software (SW-CMM) and the Systems Engineering Capability Model (SECM). This group reviewed an initial version, v0.1. By December 1999, with the help of these stakeholders, the CMMI project team was released version 0.2 for public review.

For this review, the model was made available on the Web, where reviewers could download it and submit feedback. "This was a public review," says Mike Phillips, project manager of the CMMI project, "and we wanted to make sure the right people were aware of the opportunity: we sent emails to SEI stakeholders, systems engineering groups, lead assessors, transition partners, and others."

This review provided the first opportunity for the broader community to give feedback on the model. "Sometimes it's not the people who are most familiar that give you the best input, says Phillips. "The most useful input may be from the person who's struggling just to get started. So we wanted to listen to all of them." This version was made available for public review September 1, 1999, for a 90 day period.

In addition to the 90-day public review, two other means of gathering feedback were used: First, the team members acted as observers, accompanying the assessment teams that were piloting the CMMI model, training, and assessment method. Observations made during the pilots—for example, problems with interpreting or using the model—were written up as change requests. Observers and members of the assessed organizations provided the input for these change requests.

The other method used to collect feedback was at focus groups conducted in conjunction with SEI training classes such as the High Maturity Practices Workshop. Participants in these workshops were personnel from organizations rated at high maturity levels, typically CMM levels four and five. After the workshop, CMMI team members asked volunteers to participate in a focus group. As the CMMI project team piloted the Introduction to the CMMI course, attendees were encouraged to spend an additional half day to help improve the model and the training.

About 3000 change requests were received from the public review, pilots, and focus groups. The team has spent the first half of 2000 processing these change requests, making decisions about the input received, and updating the model based on their decisions. "It's important to point out," says Phillips, "that it's not just the SEI deciding whether to accept a change request. In fact, the majority of the CMMI project team reviewing CRs--about 60-70%--are from industry and government. We developed a voting process because we weren't able to come to consensus on everything. If anybody voted against the majority, the minority opinion was heard and then a revote was taken. In some cases, the minority made their point clearly enough that they changed the decision of the group."

Revising the Model

As in any review, conflicting requirements often meant making difficult tradeoffs. For example, a large number of reviewers indicated that they would like the model to be smaller. "Concentrating" the model, while maintaining its technical integrity, proved to be one of the team's biggest challenges. For example, "peer reviews" was a dedicated key process area in the software CMM. But because of the need to reduce the size of the model, in addition to the need to include in the model other methods for verifying the quality of work products, the team chose to put peer reviews at the goal level rather than as a process area. "That can be difficult, says Phillips, "because some organizations have said 'this is one of the most valuable process areas that you had captured, but now you seem to be giving it less emphasis.' But a goal is still at a level where a weakness there would be noted in any kind of assessment. So I believe we've captured the essence of the request. But those are the kind of difficult decisions that, to keep the model from growing out of bounds, we've had to make."

All change requests and the decisions made about them were captured in a decision log, which will be made available for individuals interested in tracking changes to the model.

Releasing the Model

The model is scheduled for release later this year. Team members encourage the organizations that will be using CMMI to continue to provide feedback on how to further refine the model to meet their need for improving how they develop software intensive systems.

For more information contact--

Customer Relations
Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu/cmm/cmms/cmms.integration.html

Information Security Training and Education

Barbara Laswell

The complexity of today's software, hardware, and networking products presents challenges to technical staff attempting to configure systems and networks to use the strongest security measures appropriate; these challenges are significant even for trained, skilled technical staff. In this environment, small mistakes can leave organizations and stores of information assets at risk and vulnerable to attack.

The Software Engineering Institute's Networked Systems Survivability (NSS) Program in conjunction with its CERT® Coordination Center is working to mitigate these risks by providing leading-edge training and education programs in the area of information security for both technical and non-technical staff and managers.

A National Call for Action

The need for information security training programs is underscored by the White House's National Plan for Information Systems Protection,[1] which calls the shortage of adequately trained information systems security personnel "acute." Moreover, Program Seven of the national plan states plainly its objective as being to "train and employ adequate numbers of information security specialists." Looking at the number of people and the skills required for information security specialists both within the federal government and elsewhere, Program Seven points to a clear need to take decisive action to train current federal information technology workers and recruit and educate additional personnel to meet anticipated shortfalls.

In addition, a report published last year by the Office of the Secretary of Defense[2] outlined five "critical" information assurance functions to incorporate into training programs. Of these five functions, four are being met already:

- system/network administration and operations
- threat and vulnerability assessment
- computer emergency response team
- Web security

---

1 Available on the Web at http://www.ciao.gov/National_Plan/national_plan%20_final.pdf

2 Information Assurance and Information Technology: Training, Certification, and Personnel Management in the Department of Defense, August 27, 1999

These functions are incorporated into existing training media in the form of security improvement modules (a series of documents that each address a single, narrowly defined problem in network security) and the existing SEI curriculum. The only critical information assurance function not being directly addressed by the SEI is computer and network crime-though the SEI is working collaboratively with several national law enforcement agencies to help them understand the relevant technical issues.

Current Course Offerings

The SEI currently offers six courses (please see the sidebar accompanying this article). Three of the course offerings derive from out of the work of the CERT Coordination Center. As such, these courses are aimed at providing introductory and advanced training for technical staff and the management of computer security incident response teams.

The NSS Program also offers three courses centered around broader Internet security issues. Its Information Security for System and Network Administrators is an intensive five-day course for technical staff. Other offerings in the program are geared toward educating policymakers, managers, and senior executives who in some capacity are charged with or are responsible for the security of information assets.

The program also is developing courses in support of its Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE^{SM}) method. OCTAVE is a comprehensive, repeatable, self-directed technique for identifying and analyzing information security risks. These courses will allow an organization to independently direct and manage its own OCTAVE evaluation.

These training and education activities are aimed at improving the information security practices of technical staff and management. Public courses are offered periodically and can be attended by anyone, with a reduced charge for U.S. military and other U.S. government personnel. In addition, customer-site courses are offered to individual organizations with-again-a reduced fee for U.S. government organizations.

Extending Impact and Reach

In addition, the NSS Program is in the process of licensing its course materials to outside organizations. Licensing courses will extend the reach and impact of both the NSS Program and the CERT Coordination Center.

In another effort to extend the impact of its work, the program collaborated with the Carnegie Mellon University H. J. Heinz III School of Public Policy and Management and developed a curriculum in information security management. NSS Program and CERT

Coordination Center staff members are teaching courses in the Information Security Management specialization of the Master of Information Systems Management program.

Use of the Internet is replacing other forms of electronic communication. As the technology is being distributed, so is the management of that technology. As such, system administration and management often falls upon people who do not have the training, skill, resources, or interest needed to operate their systems securely: the number of directly connected homes, schools, libraries, and other organizations without trained system administration and security staff is rapidly increasing.

Information security training will enhance the ability of administrators and managers to use available technology to safeguard systems from attack and further maintain the integrity of information assets.

CERT® Coordination Center Training and Education in Network Computing Security

The CERT Coordination Center (CERT/CC) offers learning opportunities in network computing security through its training courses aimed at private and public sector professionals. Courses are offered onsite at SEI facilities in Pittsburgh, Pennsylvania and Washington, DC (Arlington, Virginia). The CERT/CC also offers customer-site deliveries of its courses and establishes licensing agreements with organizations interested in providing instruction.

The following are currently scheduled courses:

Incident Handling Courses

Managing Computer Security Incident Response Teams
Managing Computer Security Incident Response Teams (CSIRTs) is a three-day course that provides managers and future managers of computer security incident response teams with a pragmatic view of the issues that they must face and pitfalls to avoid.

Computer Security Incident Handling for Technical Staff
Computer Security Incident Handling for Technical Staff (Introductory) is a four-and-one-half-day course designed for CSIRT technical personnel who are new to incident handling and would like to improve their skills through formal training.

Computer Security Incident Handling for Technical Staff (Advanced)
Computer Security Incident Handling for Technical Staff (Advanced) is a four-and-one-half-day course designed for CSIRT technical personnel with several months of incident handling experience.

Network Security Courses

Concepts and Trends in Information Security
This one-day overview gives system administrators, network administrators, information systems security officers, and information systems security managers improved knowledge of security issues, techniques, and trends related to the confidentiality, integrity, and availability of information assets on an organization's computer systems.

Information Security for System Administrators
This five-day course is for system, network, and security administrators who have some on-the-job experience with information security. Using exercises and demonstrations featuring the latest trends in information security, this course provides the information and approaches that technical professionals need to protect and secure their organization's information assets.

Managing Risks to Information Assets
This intensive two and one-half-day course enables managers to develop an effective approach for managing risks to their organizations' information assets and computer infrastructures under the conditions of uncertainty created by an increasingly global networked environment.

Executive Role in Information Security: Risk and Survivability
This one-day overview of technology security provides executives with a foundation for understanding the activities and resources required to address the information security needs of their organization. Included is discussion of current security attacks and elements of executive response related to these attacks. This one-day course is also available as a customized two-hour overview.

Executive Education
Staff is currently teaching executive courses in information security through the H. John Heinz III School of Public Policy and Management at Carnegie Mellon University. The courses provide information and techniques relevant to issues of information security at all levels- systems, organizations, and policy. In the courses, NSS Program staff focuses on security technology and systems risk.

For more information contact-

Customer Relations
Phone
412 / 268-5800

Joe McLeod
Email
jmcleod@cert.org

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