Carnegie Mellon
Software Engineering Institute

news @ sei interactive

Volume 5 | Number 3 | Third Quarter 2002

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

news @ sei interactive

From the Director

Regular readers of this column know that I am passionate about software quality. In the last issue, I proposed a Quality Doctrine for Software. And in a previous issue, I expressed my opposition to the Uniform Computer Information Transactions Act (UCITA), which says, in part, that vendors of software cannot be held liable for defects in the products that they produce.

A study commissioned by the Department of Commerce's National Institute of Standards and Technology (NIST) in June quantifies the effect of software defects on the U.S. economy. The study, conducted by the Research Triangle Institute (RTI) in North Carolina, concludes that software defects cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross domestic product. Because of the rapid growth of software usage, this number has undoubtedly increased in the past, and unless we do something to change current practices, it will almost certainly increase in the future.

These results confirm what we at the SEI have been saying for years. Commercial software products are riddled with defects—commonly known as "bugs"— that are introduced during the software's design and development. Defects in products that are accessible to the Internet render them vulnerable to cyber attacks. The CERT® Coordination Center (CERT/CC) at the SEI documented more than 2,500 commercial-product vulnerabilities last year and determined that more than 95% of the 53,000 unique cyber incidents it investigated were a direct result of intruders exploiting such vulnerabilities. Yet there is only a small number of root causes that underlies the massive number of vulnerabilities seen in commercial software. These defects, and hence most cyber attacks, could be prevented if vendors used the proven best design techniques of software engineering.

While I agree with the diagnosis of the NIST study, I disagree with the study's conclusion: that higher software quality can best be achieved by improved software testing. In the SEI's view, the best way to ensure the quality of our software is not to allow defects into software in the first place. Practices for designing high-quality software already exist; at the SEI, we have been identifying, developing, and advocating such practices since 1984.

Commonly used software-development practices—including an overreliance on testing—result in lost productivity, as time and money are wasted on rework. The Standish Group in 1999 reported that 23% of software and information-technology projects were can-celled before completion, while only 28% finished on time and budget with expected functionality.[1] Standish Group data also indicate that 60-80% of the cost of software development is rework—that is, fixing defects that are found during testing. This cost would be avoided if better design practices were used.

---

[1] CHAOS Chronicles II, The Standish Group, 2001, ttp://www.standishgroup.com/reports/reports.php

We in the software development community can reduce development costs and improve quality by paying more attention to the early phases of the development life cycle. For example, the Team Software Process^{SM} (TSP^{SM}) and the Personal Software Process^{SM} (PSP^{SM}) provide specific quality measures and practices, together with training and introduction methods to guide their proper use. For more information, see http://www.sei.cmu.edu/tsp/?ns. At the SEI, our work will continue to emphasize defect prevention through improvement of process and product quality at the requirements and design stages of systems development.

We hope you find these articles stimulating and informative. As always, we invite you to send your comments and suggestions to news-editor@sei.cmu.edu.

Stephen E. Cross
SEI Director and CEO

The Evolution of Quality Attribute Workshops as an Architecture-Evaluation Technique

Mario Barbacci

In previous columns,[1] I described initial experiences applying Quality Attribute Workshops (QAWs) to evaluate the implications of system-design decisions. This column provides an update on the development of the method and provides lessons learned from applying the QAW method in four different U.S. government acquisition programs. Most of these lessons were integrated into the method incrementally, as described in a recent SEI technical report [1].

QAWs provide a method for analyzing a system's architecture against a number of critical quality attributes, such as availability, performance, security, interoperability, and modifiability, that are derived from mission or business goals. The QAW does not assume the existence of a software architecture. It was developed to complement the Architecture Tradeoff Analysis Method℠ (ATAM℠) (http://www.sei.cmu.edu/ata/ata_method.html) in response to customer requests for a method to identify important quality attributes and clarify system requirements *before* there is a software architecture to which the ATAM could be applied. The QAW analysis is conducted by applying a set of test cases to a system architecture, where the test cases include questions and concerns elicited from stakeholders associated with the system. In this column, I describe the activities in the QAW method, how it has been adapted to specific customer needs, and several lessons learned during the evolution of the process.

The QAW process, shown in Figure 1, can be organized into four distinct groups of activities: (1) scenario generation, prioritization, and refinement; (2) test case development; (3) analysis of test cases against the architecture; and (4) presentation of the results. The first and last segments of the process occur in facilitated one-day meetings. The middle segments are undertaken independently by those developing or analyzing the test cases, and may involve experimentation that continues over an extended period of time.

The process is iterative in that the test-case architecture analyses might lead to the development of additional test cases or to architectural modifications. Architectural modifications might prompt additional test-case analyses, and so forth.

The first activity in the QAW process is to generate, prioritize, and refine scenarios. In the QAW, a scenario is a statement about some anticipated or potential use or behavior of the system (see sidebar 1).

---

[1] http://interactive.sei.cmu.edu/news@sei/columns/the_architect/2001/2q01/architect-2q01.htm, http://interactive.sei.cmu.edu/news@sei/columns/the_architect/2000/spring/the_architect.htm

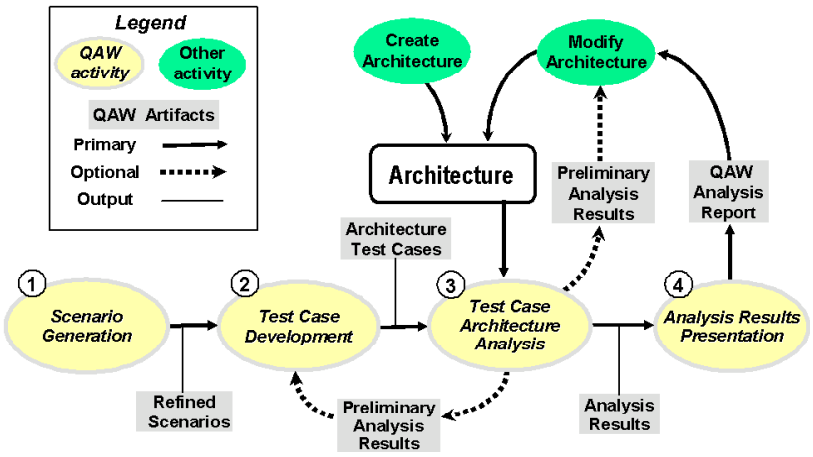


Figure 1: The QAW Process

Scenarios are generated in a brainstorming, round-table session and capture stakeholders' concerns about how the system will do its job. Only a small number of scenarios can be refined during a one-day meeting, so stakeholders must prioritize the scenarios generated previously by using a voting process. Next, the stakeholders refine the top three or four scenarios to provide a better understanding of their context and detail (see sidebar 2). The result of this meeting is a prioritized list of scenarios and the refined description of the top three or four scenarios on that list.

The next activity in the QAW process is to transform each refined scenario from a statement and list of organizations, participants, quality attributes, and questions into a well-documented test case. The test cases may add assumptions and clarifications to the context, add or rephrase questions, group the questions by topic, and so forth (see sidebar 3). Who is responsible for developing the test cases depends on how the method has been applied and who carried out the task (e.g., sponsor/acquirer or development team).

Sidebar 1

Scenarios

The quality attributes are characterized by stimuli, responses, and the architectural decisions that link them. Stimuli and responses are the activities (operational or developmental) that exercise the system and the observable effects, respectively. A good scenario clearly states which stimulus causes it and which responses are of interest.

There are several types of scenarios:

- **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 the operator's console).
- **Exploratory scenarios** involve extreme changes to the system that may be unanticipated and that may occur in undesirable situations. Exploratory scenarios are used to explore the boundaries of the architecture (e.g., message traffic grows 100 times, requiring the replacement of the operating system).

The distinction between growth and exploratory scenarios is system or situation dependent. What might be anticipated growth in a business application might be a disaster in a deep space probe (e.g., 20% growth in message storage per year). For example, a use-case scenario might be "Remote user requests a database report via the Web during peak period and receives it within 5 seconds." A growth scenario might be: "Add a new data server to reduce latency in scenario 1 to 2.5 seconds within 1 person-week," and an exploratory scenario might be "Half of the servers go down during normal operation without affecting overall system availability." There are no clear rules other than stakeholder consensus that some scenarios are likely (desirable or otherwise) and others are unlikely. (If the unlikely ones occur, it would be useful to understand their consequences.)

The test-case architecture analysis is intended to clarify or confirm specific quality attribute requirements and might identify concerns that would drive the development of the software architecture. Some of the test cases could later be used as "seed scenarios" in an ATAM evaluation (e.g., to check if a concern identified during the test-case analysis was addressed by the software architecture). The results of analyzing a test case should be documented with specific architectural decisions, quality attribute requirements, and rationale (see sidebar 4).

The results presentation is the final activity in the QAW process. It is a one- or two-day meeting attended by facilitators, stakeholders, and the architecture team. It provides an opportunity for the architecture team to present the results of its analysis and to demonstrate that the proposed architecture is able to handle the cases correctly.

Sidebar 2

Scenario Refinement

The refinement activity elicits further details, such as the expected operational consequences, the system assets involved, the end users involved, the potential affects of the scenario on system operation, and the exceptional circumstances that may arise. For example, a scenario such as "a communication relay node failed" does not really capture the consequences or implications of the failure. For this scenario, further details would include which facility or node detects failure, what is the expected automated response to failure (if any), what is the expected manual intervention, which capabilities will be degraded during the outage, and what are the expected actions to be taken to return the relay to service.

<System/Organization Title>—Scenario Refinement	
Reference Scenario(s)	"Mars orbital communications relay satellite fails."
Organizations	Authorities in multiple organizations and control centers
Actors/ Participants	Flight director, mission director
Quality Attributes	Performance (P) and Availability (A)
Context	One of three aero-stationary satellites around Mars fails. Mars surface elements and Mars satellites know that it failed and report the failure to the Earth control element. Traffic rerouting is to be performed and network reconfiguration dictated by the flight director, perhaps postponed to limit the possibility of further failure. Service assessments are done at the control center (within two days) using a well-defined decision-making process, leading to the mission director (as the final authority). Multiple missions will be running simultaneously. Currently there are two, but coordination is complex.
Questions	• How long does it take to detect the failure? • Is there a way to send information to Earth for analysis? • Can a crew in the space station help in the transmission? • How long does it take to reconfigure the system? • What redundancy is available? • Can the crew participate in the repair? • Can the customer participate in the notification (e.g., "Please send a message to the other satellite")? • Is there a way for customers to simplify their procedures to handle more missions?

Tailoring QAW

The application of the method can be tailored to the needs of a specific acquisition strategy and might include incorporating specific documents or sections of documents into the request for proposals (RFP) or contract [2].

Application Before Acquisition

In one application, the QAW method was used in a pre-competitive phase for a large system. Stakeholders involved laboratories and facilities with different missions and requirements. An architecture team (with members from various facilities) was building the architecture for a shared communications system before awarding a contract to a developer, and tailored the QAW process as follows:

- Stakeholders from different facilities held separate meetings to generate, prioritize, and refine scenarios.
- The architecture team turned these refined scenarios into test cases and analyzed the proposed architecture against them.
- The architecture team then presented the results of the analysis first to a review team, and later to the original stakeholders.

Application During Solicitation and Proposal-Evaluation Phases

Figure 2 illustrates a common acquisition strategy. Starting with an initial request for proposals, the acquisition organization evaluates proposals from multiple contractors and chooses one to develop the system.

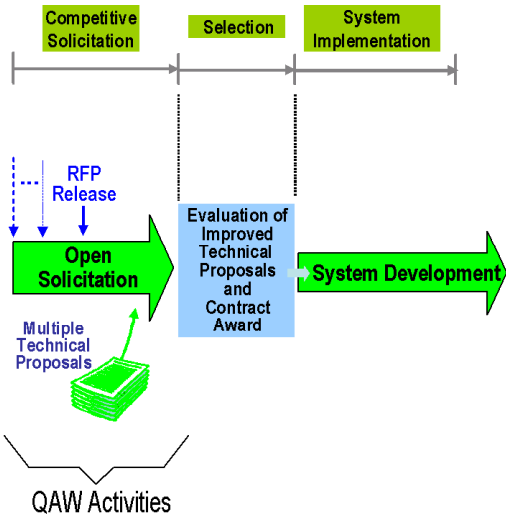


Figure 2: Common Acquisition Strategy

Test Cases

A test case has a context section outlining the important aspects of the case, the mission or activity, the assets involved, the geographical region, the operational setting, the players, and so forth. The context is followed by a description of the issues implied by the context and a list of questions that connect these issues to quality attributes. The utility tree provides a visual summary of the quality attributes of importance and the specific issues and questions that pertain to the attributes.

Example Test Case

Test-Case Context and Activities

Humans and robotic missions are present on the Mars surface when one of three aero-stationary satellites has a power amplifier failure. The primary communications payload is disabled for long-haul functions, but the proximity link to other relay satellites and customers on orbit and on the surface still works. Secondary Telemetry and Tele-Command (TTC) for spacecraft health is still working for direct-to-Earth with a low data rate. The remaining two satellites are fully functional. Communications with the crew have been interrupted. The crew is not in an emergency situation at the time of the failure, but reconnection is needed "promptly." The crew on the surface is concentrated in one area, and the other missions in the Mars vicinity are in normal operations or non-emergencies, or are performing mission-critical events. The event occurs late in the development of the communications network, so the system is well developed.

Quality Attribute Questions

Issue Mission safety requires consistent and frequent communications between the crew and Earth. (P, A)

Questions a) How long does it take to detect the failure?
b) How long does it take to reconfigure the system to minimize the time the crew is without communication?

Issue: System operation will be degraded. (P, A)

Questions a) Is there a way for customers to simplify their procedures so they can handle a larger number of missions with less trouble than coordinating two as they do now?
b) What redundancy is required?
c) Is there a way to send information about the degraded satellite back to Earth for analysis?

Issue System recovery (P, A)

Questions a) Can the crew participate in the repair?
b) Is there any expectation for a human interface between Mars and the Earth (e.g., crew in space station)?
c) Can the customer participate in the notification (e.g., "Please send a message to the other satellite")?

Utility Tree

Root	Quality Attribute	Specific Attribute Issue	Question
Utility	Performance	...of communications...	(1b) How long to reconfigure?
		...degraded operation...	(2a) Can decisions be simplified?
			(2c) How is information sent back?
	Availability	...mission safety...	(1a) How long to detect the failure?
		...redundancy...	(2b) What redundancy is required?
		...recovery...	(3a) Can the crew help?
			(3b) Can space station help?
			(3c) Can other assets help?

In one application of the QAW method, the QAW activities took place during the competitive selection, and were customized as follows:

- Before the competitive solicitation phase, scenario-generation meetings were conducted at three different facilities. These were representative of groups of users with similar needs and responsibilities (e.g., technicians, supervisors, analysts at headquarters).
- Early in the competitive solicitation phase and before the release of the RFP, the acquirer conducted bidders' conferences to inform potential bidders about the need for conducting architecture analysis.
- The acquirer developed several test cases for each type of user, drafted sections of the RFP to incorporate architecture-analysis requirements, and included the test cases as government-furnished items (GFIs) in the RFP proper.
- As part of their proposals, the bidders were expected to conduct an architecture analysis of the RFP test cases, present their results to the acquirer, and write reports consisting of the results of their analysis, their response to requests for clarification, risk-mitigation plans for the risks identified during the presentation, and any new or revised architecture representations.

Application During a Competitive Fly-off

Figure 3 illustrates a "rolling down select," a different acquisition strategy. Starting with an initial request for proposals, the acquisition organization awards contracts to a small number of contractors to conduct a "competitive fly-off." In this phase, the contractors work on a part of the system, still competing for award of the complete contract. At the end of the phase, the contractors submit updated technical proposals, including additional details, and the acquirer makes a final "down select," or selection of one of the competing contractors.

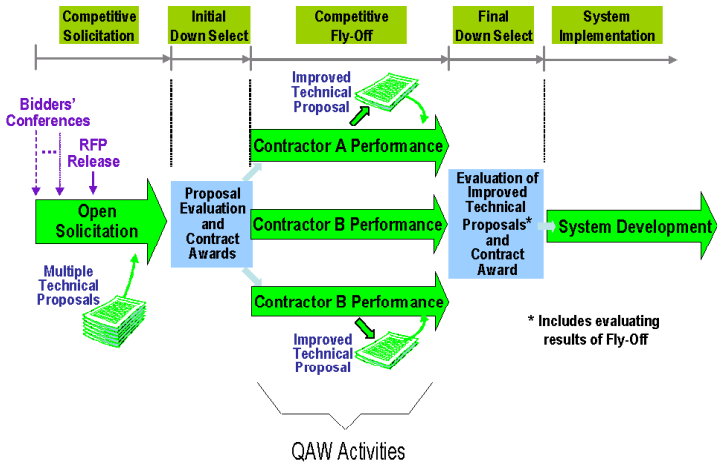


Figure 3: Acquisition Strategy Using Competitive Fly-Off

In this application, the QAW method was used during the Competitive Fly-Off phase (with three industry teams competing) of the acquisition of a large-scale Command, Control, Communications, Computers, Intelligence, Surveillance, and Recognizance (C4ISR) system. In this case, the QAW process was customized as follows:

- The scenario-generation meetings were conducted with each contractor separately. As a result of these meetings, participants gained an understanding of the process, a list of prioritized scenarios, and a set of refined high-priority scenarios.
- A government technical assessment team (TAT) used these scenarios to develop a number of test cases. Changes were made to hide the identity of the teams and extend the coverage of the scenarios over a set of assets, missions, and geographical regions. An example was developed to make the process more understandable, and copies were distributed to all industry teams.
- The contractors performed the analysis and presented the results in a *dry-run* presentation. There was a large variation in the presentations for these meetings, ranging from performing only one test case in great detail, to performing all test cases in insufficient detail. Each contractor was then informed of how well it did and how it could improve its analysis. The contractors then completed the analysis in a final presentation of the results, allowing them to correct any flaws.

Lessons Learned

Scenario Generation and Refinement

The scenario-generation meeting is a useful communication forum to familiarize stakeholders with the activities and requirements of other stakeholders. In several cases, the developers were unaware of requirements brought up by those with responsibility for maintenance, operations, or acquisition. In one case, potential critics of the project became advocates by virtue of seeing their concerns addressed through the QAW process. We also learned that the facilitation team has to be flexible and adapt to the needs of the customer, as the following observations indicate:

- The approach relies on identifying the right stakeholders and asking them to do some preparatory reading and attend the meeting for a day. In one case, the task of inviting these stakeholders fell on the architecture team, which created some awkward situations. The hosts of the meeting need a way of attracting the right people to the meeting. This could include invitations explaining the advantages of participating, and recommendations from upper management to cause interest in attending.
- The process of generating scenarios in a brainstorming session is usually inclusive, but the process for refining the high-priority scenarios might not be. Some stakeholders might feel left out of the refining effort if other, more vocal stakeholders dominate the process. It is the responsibility of the facilitators to make sure that everyone can contribute. The template describing specific details to be identified during the scenario refinement was a great improvement over the initial refinement exercises, because it kept the stakeholders focused on the task at hand and avoided diversions.

- Some of the scenarios or questions generated during the refinement might not be focused on quality attributes. This is usually an indicator that the issues involved are "hot buttons" for some of the stakeholders. Although we normally try to focus the scenarios on quality attributes, the underlying issues could be important, and on occasion, we have allowed the scenarios and questions to stand.

- The scenarios generated in a meeting can be checked against system requirements in two ways. First, unrefined scenarios can be checked for whether they relate to existing requirements. If they do, requirements details might be used to refine the scenarios. Second, refined scenarios can lead to a better understanding of some requirements. Undocumented requirements can be discovered by both means.

- The scenarios generated in a meeting can be checked against the expected evolution of the system. In projects planning a sequence of releases, the scenarios should specify the release to which they apply, ensuring that the projected deployment of assets and capabilities match the scenarios and test cases.

Developing the Test Cases

Building the test cases from the refined scenarios takes time and effort.

In one case, the QAW facilitators did not extract sufficient information during the refinement session to build the test cases, and the facilitators had to organize additional meetings with domain experts to better define the context and quality- attribute questions. An unintended consequence was that the resulting

Sidebar 4

Example Test-Case Architecture Analysis

In our example, there are two important issues that affect availability and crew safety: satellite locations and monitoring the health of the satellites.

Satellite Locations. *The initial architecture has three operational aero-stationary satellites. Each satellite has visibility over a fixed area of Mars. When the satellite fails, its area of responsibility is no longer in communication with Earth and this area contains the crew.*

The risk of a satellite failure is having inadequate communications with the crew on Mars. This is a serious problem but it can be alleviated by a number of architectural approaches:

- Move one or two of the other stationary satellites to provide communications with the crew. This will degrade communications between the relocated satellites and the assets in their original area of responsibility. This solution constitutes a tradeoff between the quality of communications in different areas of responsibility.
- Place one or more in-orbit spare satellites. A spare can be moved to the location of the failed satellite and take over its area of responsibility. This solution constitutes a tradeoff between cost (additional satellites) and speed of repair (the more spares, the quicker one could be moved to the desired location).
- Place "feeder antennas" on the surface of Mars to relay communications from the crew to (eventually) another satellite (i.e., rerouting the traffic in case of failure). This solution is a tradeoff between cost (the antennas) and the quality of the communications (the volume and latency of messages).
- Place the satellites in a "slow-drift" orbit, such that the loss of a satellite will not cause a complete communications failure. There will be times when one or more of the other satellites will be in sight and times when no satellite is in sight. This is probably the least disruptive solution, provided the periodic (but predictable) loss of communications can be tolerated. In case of a satellite failure, the crew will have communications when the next satellite drifts over the area.

Health of the Satellites. Monitoring the health of the satellites will improve the availability of the communications with the crew. A health monitor could help predict imminent failures, detect recent failures, and plan corrective actions.

Without a health-monitoring system, there is a risk that satellite failures could go unnoticed until they are needed to provide communications to the crew on the surface. There is also an additional risk of having extended down times due to the long transit time from Earth (e.g., sending a replacement satellite could take months). There are a number of alternatives, which are not mutually exclusive:

- The satellites could exchange periodic health messages among them (e.g., pushing "I am here!" messages and pulling "Are you there?" messages) and inform the ground stations of their health states. This is a tradeoff with performance because of the additional message traffic.
- The satellites could run self-tests and inform the ground stations whenever a problem is detected. This is a tradeoff with performance and probably cost (i.e., extra components to conduct the self-tests).

The mission customers could notice degradation in communications and alert the communications system operators. This is a tradeoff with usability (e.g., how to avoid false alarms).

test-case context was far more detailed than if it had been generated during the scenario-refinement session. As a result, only portions of the larger test-case context were relevant to the test-case questions. We learned that having an extremely detailed test-case context is not worthwhile. It takes too long to develop, may be hard to understand, and does not lead to focused questions. A test case should not be more than a few sentences.

Since the software architecture is not yet in place, the questions and expected responses should not force design decisions on the development team. Hence, the questions must be quite general, and the expected responses may suggest architectural representations (for example, "what is the availability of this capability?") but not design solutions (for example, "use triple modular redundancy for high availability").

Analyzing the Architecture Using Test Cases

The test-case architecture analysis might reveal flaws in the architectures and cause the architecture team to change the design. The test cases generated by the QAW process often extend the existing system requirements.

In one case, the new requirements seemed to challenge the requirements-elicitation effort and raised concerns on the architecture team. A typical comment was, "The system wasn't meant to do that." Some judgment must be made as to which test cases can be handled and at which phase of system deployment. While this can lead to extended arguments within the team, it is a useful exercise, since these concerns must be resolved eventually.

In another case, the stakeholders were concerned because the process only analyzed a few test cases out of a large collection of scenarios. They wanted to know what was to be done with the remaining scenarios. This issue should be resolved before the scenario-generation meeting. One approach is to analyze the architecture incrementally against an ever-expanding set of test cases and, if necessary, adjust the architecture in each increment. However, this approach is constrained by budgets, expert availability, and participants' schedules.

Results Presentation

Like in the scenario-generation meeting, participants are provided with a handbook before the meeting. The handbook includes the test cases and provides a test-case analysis example so the participants know what to expect at the meeting. In some applications of the QAW, we have conducted the results presentations in two phases: first as a rehearsal, and then as a full-scale presentation. The following observations are derived from conducting a number of QAWs:

- In one case, the initial example given in the participants' handbook was too general. This reduced the level of buy-in from participants. We corrected this by developing another example with the right level of detail.

- A dry-run presentation should be conducted when the architecture team making the presentation is unsure about (*a*) the level of detail required; (*b*) the precision expected from its answers to the test-case questions; (*c*) how to incorporate other analysis results (for example, reliability, availability, and maintainability analysis; or network loading analysis); or (*d*) what additional architectural documents might be needed.

- The full-scale presentation takes place after "cleaning up" the results of the dry-run presentation. Concerns that arise in the full-scale presentation have to be addressed as potential threats to the architecture.

The process for conducting QAWs is solidifying as we continue to hold them with additional customers, in different application domains, and at different levels of detail. The approach looks promising. The concept of checking for flaws in the requirements before committing to development should reduce rework in building the system.

References

[1] Barbacci, M. et al. *Quality Attribute Workshops, 2nd Edition* (CMU/SEI-2002-TR-019). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports/02tr019.html>

[2] Bergey, J. & Wood, W. *Use of Quality Attribute Workshops (QAWs) in Source Selection for a DoD System Acquisition: A Case Study* (CMU/SEI-2002-TN-013). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports/02tn013.html>

About the Author

Mario Barbacci is a Senior Member of the staff at the Software Engineering Institute (SEI) at Carnegie Mellon University. He was one of the founders of the SEI, where he has served in several technical and managerial positions, including Project Leader (Distributed Systems), Program Director (Real-time Distributed Systems, Product Attribute Engineering), and Associate Director (Technology Exploration Department). Prior to joining the SEI, he was a member of the faculty in the School of Computer Science at Carnegie Mellon University.

His current research interests are in the areas of software architecture and distributed systems. He has written numerous books, articles, and technical reports and has contributed to books and encyclopedias on subjects of technical interest.

Barbacci is a Fellow of the Institute of Electrical and Electronic Engineers (IEEE) and the IEEE Computer Society, a member of the Association for Computing Machinery (ACM), and a member of Sigma Xi. He was the founding chairman of the International Federation for Information Processing (IFIP) Working Group 10.2 (Computer Descriptions and Tools) and has

served as chair of the Joint IEEE Computer Society/ACM Steering Committee for the Establishment of Software Engineering as a Profession (1993-1995), President of the IEEE Computer Society (1996), and IEEE Division V Director (1998-1999).

Barbacci is the recipient of several IEEE Computer Society Outstanding Contribution Certificates, the ACM Recognition of Service Award, and the IFIP Silver Core Award. He received bachelor's and engineer's degrees in electrical engineering from the Universidad Nacional de Ingenieria, Lima, Peru, and a doctorate in computer science from Carnegie Mellon.

CMMI and Process Improvement Themes

Mike Phillips

This is the first column that we intend to use to convey information to those in our community who are either adopting the Capability Maturity Model® Integration (CMMI®) Framework or are actively considering this approach to enterprise process improvement. We will use this column to explore some of the ideas that guided what is in the CMMI models, as well as some of the issues that we hear about from the community in courses, at conferences, or from direct email to cmmi-comments@sei.cmu.edu.

In this article, we would like to explore how the CMMI models map to three process improvement themes that inspire much of the work done at the SEI. The three themes are "move to the left," "reuse everything," and "never make the same mistake twice." While each of these has broad applicability in improving the practice of software engineering, we will explore some of the specific ways that CMMI supports these ideas.

Move to the left

This first theme recognizes the value of effective early preparation to avoid introducing defects into a product. CMMI accomplishes this in several ways. One of the most obvious, and a direct heritage of the Capability Maturity Model (CMM®) for Software (SW-CMM), is the emphasis on planning. In CMMI, one of the fundamental process areas is the Project Planning process area. The practices of this process area, implemented up front, provide the basis for all of the management and engineering tasks. While CMMI avoids specifying any particular plans, this process area describes the activities that will enable an effective development plan to be created. Two particular innovations that were not addressed in the SW-CMM are the attention to planning the involvement of various stakeholders across the development life cycle and the inclusion of planning for data management. Also, to remind us that planning is fundamental to all activities of the enterprise, all CMMI process areas include a generic practice, Plan the Process, that helps the organization achieve a level 2 (managed) capability in the appropriate CMMI process areas.

In the area of requirements, CMMI takes another step to the left by focusing attention on requirements development. This focus includes the key steps of eliciting and validating requirements, reminders to keep the ultimate users in mind to ensure that the product will be suitable in meeting the customer's needs. Also, our experience at the SEI is that too often insufficient attention is paid to effectively architecting the product. As a result, the SEI does significant work in architecture tradeoff analysis (http://www.sei.cmu.edu/ata/ata_init.html).

Several books[1] have been written by SEI members of the technical staff about improving the practices of performing this essential up-front work. CMMI now includes practices to help ensure that effective product architectures are designed and evaluated.

As process area capability and organizational maturity increases through implementation of CMMI practices, attention to defects gains more and more focus.

Reuse Everything

The concept of reuse is very familiar to the software community. The most effective demonstration of this has been the strategy of product line development to maximize the ability to reuse major subsystems in a variety of systems. (This requires proper attention to architectural issues across the product suite, and for that reason, the SEI initiative for product lines (http://www.sei.cmu.edu/plp) is linked closely with the one described above for architecture tradeoff analysis.) In CMMI, we decided not to require product line thinking, but to support it across the model. A recent technical note (http://www.sei.cmu.edu/publications/documents/02.reports/02tn012.html) describes the way that CMMI supports the move to a product line framework.

Probably one of the best examples of reuse within CMMI is the emphasis on the Organizational Process Focus and Organizational Process Definition process areas. This emphasis allows development of organizational standard processes, which can be reused by new projects and tailored to meet project-specific needs, becoming defined processes. As comfort grows with the suite of standard processes, the Organizational Process Performance process area further aids their effective reuse. Reuse of proven processes is of course a legacy from the SW-CMM, but in addition, CMMI takes reuse one step further by providing more attention to identifying up front when product development may benefit from reuse, and adapting the product requirements and design accordingly.

Another area with strong linkage at the SEI is the increased attention to using commercial off-the-shelf (COTS) components. Since COTS products are obtained outside of the project, the guiding information is included in the Supplier Agreement Management and Integrated Supplier Management process areas. However, this is another rich field benefiting from greater guidance, and it can be found within the COTS-Based Systems Initiative (http://www.sei.cmu.edu/cbs) at the SEI.

Never Make the Same Mistake Twice

We often are reminded of the quote “insanity is repeating the same procedure and expecting different results.” This theme covers first defect identification and removal, and then, as capability increases, defect prevention. These concepts again were well expressed in the SW-

---

[1] See the following page for a complete list: <http://www.sei.cmu.edu/ata/books.html>.

CMM, but have been worded now in ways that users suggest is a bit clearer in the CMMI process areas. At lower levels of capability and maturity, we focus on detecting problems and taking corrective actions. These are addressed in the Project Monitoring and Control and the Measurement and Analysis process areas. By implementing practices of these process areas, problems in achieving project and organizational objectives are identified and corrective actions are tracked to closure. However, peer reviews, another heritage from the SW-CMM (and now a goal within the Verification process area), provide opportunities to peer review not just the development items, such as software code, but the planning elements themselves—again, a move to the left.

With the higher level attention to quantitative and statistical process control, we gain the ability to look first at special causes of variation (level 4 activities in Quantitative Project Management) and then at common "root" causes of variation (level 5 activities in Causal Analysis and Resolution). We can then improve processes to avoid introducing defects rather than removing them once they are observed. This effort is closely linked with the companion advanced process area, Organizational Innovation and Deployment, which manages the selection and deployment of improvements across the organization.

Conclusion

These three themes apply in many ways to improving the practices of software engineering. The CMMI represents a valuable framework for inculcating these themes into the development effort to reduce chaos and rework—to make the development effort more rewarding to both developers and end users.

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.

Building Systems from Commercial Components using EPIC

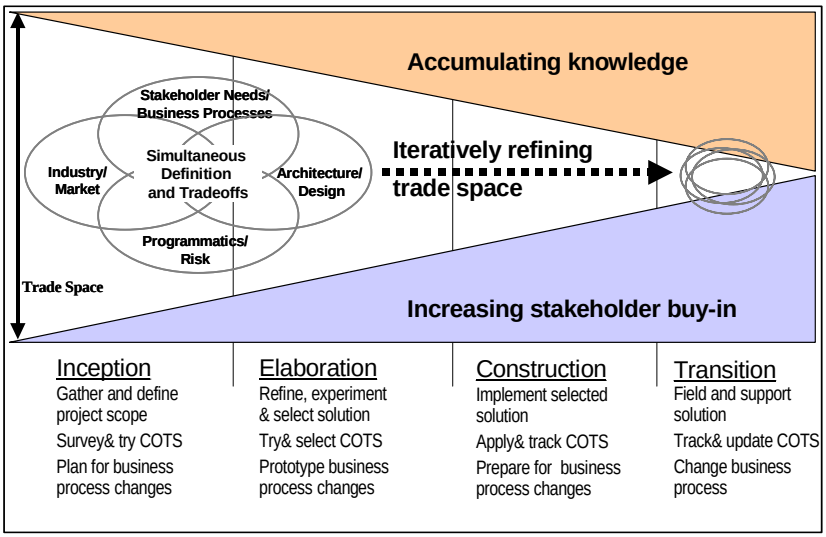
Robert C. Seacord and Russ Bunting

Experience has shown that new management and engineering processes are needed for building, fielding, and supporting systems that make extensive use of commercial components. Existing processes tend to take either a management or an engineering perspective but not both. As a result, these processes fail to directly address the needs of the other group. Instead, disparate processes are often combined in an ad hoc fashion. This is complicated further because these processes are often based on disparate assumptions.

The COTS-Based Systems (CBS) initiative at the SEI has performed extensive research on processes needed in the management of COTS-based systems, engineering techniques for designing and evolving COTS-based systems, and evaluation techniques for assessing COTS-based program risks. While these CBS initiative products have been developed separately to meet the needs of different customers, they are compatible in that they share a common understanding of the underlying issues in building systems from commercial components.

The Evolutionary Process for Integrating COTS-based systems (EPIC), for example, extends the Rational Unified Process (RUP) to accommodate COTS-based system development. EPIC shares the RUP principles that a development process must be risk- and use-case driven, architecture centric, iterative, and incremental. As shown in the figure below, EPIC accomplishes this through concurrent discovery and negotiation of stakeholder needs and business processes; applicable technology and components; the architecture and design; and the programmatics and risk across four RUP phases of inception, elaboration, construction, and transition.

EPIC provides a structured flow of activities and artifacts that keeps the four spheres fluid and balanced as the project evolves from a strategic vision to an implemented and sustained *solution*. In EPIC, a solution is composed of the integration of one or more COTS products with any legacy or other reuse components, any required custom code (including wrappers and “glue”), and any changes to end-user business processes necessary to match the capabilities of the COTS products. EPIC makes no attempt to be complete; the detailed information necessary for implementation is left to RUP and other methods.



The CBS initiative has also produced a book in the SEI Series in Software Engineering titled *Building Systems from Commercial Components* (BSCC). BSCC describes component-based design methods, including a variety of design and engineering techniques that have been effectively applied in practice in the development of large, complex systems from commercial components. A central proposition in BSCC is that a principal source of risk in component-based design is a lack of knowledge about how components should be integrated and how they behave when integrated. BSCC describes the software engineering methods and techniques necessary to respond to the engineering challenges posed by the commercial component market.

BSCC identifies design and engineering techniques that respond to the complex, idiosyncratic, and unstable nature of commercial components. The table below shows the core elements from BSCC.

Ensembles	A design abstraction that exposes component incompatibilities by shifting emphasis from individual components to collections of integrable components.
Blackboards	A design notation that depicts what is currently known, and what remains to be discovered, about an ensemble.
R^3	A risk-driven discovery process that exposes design risk and defines ensemble feasibility criteria.
Model problems	A prototyping process for generating situated component expertise and for establishing ensemble feasibility.

How do BSCC techniques map to EPIC?

While designed to meet different needs, EPIC and BSCC share a common foundation and are complimentary. EPIC provides BSCC with a prescriptive framework that links the disparate

system stakeholders into a coherent team. Conversely, the methods and techniques described in BSCC can be used to implement the design activities and workflows of EPIC. With that in mind, the following are a few examples of how the BSCC techniques map to EPIC and how the EPIC artifacts support BSCC techniques.

Model problems are a useful mechanism in the inception and elaboration phases. In these phases, stakeholders are focused on discovering what the components do, how they are combined with other components to form an *ensemble*, and how best to structure a design solution that makes the best use of components. A model problem expresses a design question pertaining to integrating software components in its simplest, most primitive form. It defines a problem that needs to be solved, and defines criteria for evaluating solutions. Using model problems encourages identification of relevant component sources, can assist in characterizing available components, and provides a vehicle for understanding the behavior of the components in the context of ensembles that support the solution's design.

A model problem's inputs are structured as a design question or unknown that is expressed as a hypothesis with the minimum relevant constraints and the starting evaluation criteria that describe how the hypothesis will be supported or refuted. The EPIC artifact, *Component Screening Criteria and Rationale*, maps nicely to the necessary input elements of a model problem. A *model solution* is an executable prototype that answers the question posed by the model problem and demonstrates that the evaluation criteria is satisfied, cannot be satisfied, or is conditionally satisfied. This is complementary to EPIC, which emphasizes the importance of an executable representation to demonstrate the stakeholder consensus on decisions made in each iteration.

Model problems may also be used in concert with more formal evaluation techniques such as Multi-Criteria Evaluation and Risk/Misfit to evaluate applicable components. The EPIC *Component Dossier* (one for each examined component) artifact supports this evaluation by capturing a wealth of information for each component. The component data provided by the vendors is supplemented with information discovered during prototyping to provide knowledge about how the components interact in the context of the solution. Over the lifetime of a system, components may undergo significant change. The dossier acts as an audit trail or log to view the component's evolution. BSCC *blackboards* are one mechanism for representing component dossiers, used to capture component knowledge. This knowledge is labeled with *credentials* that define the source and level of confidence in the data.

Where model problems ensure that prototyping is highly efficient, R^3 (**R**isk Analysis, **R**ealize model problem, **R**epair residual risk) provides a workflow that ensures that the right questions are posed, that model solutions are developed only where significant design risk exists, and that the risks are mitigated by "repairing" them where practical. For risks that are not repaired, BSCC describes contingency-planning approaches that are fundamental to EPIC's risk-driven approach.

Summary

BSCC and EPIC can be readily used together because they are based on a common understanding of the underlying issues affecting the success of building, fielding, and supporting systems built from commercial components. Both identify an investment in risk identification, analysis, and mitigation to identify and negotiate product mismatches early in the development process as central to any successful project. BSCC provides engineering techniques such as model problems and Risk/Misfit that can provide concrete mechanisms to accomplish EPIC life-cycle activities.

References

Wallnau, Kurt C.; Hissam, Scott A.; and Seacord, Robert C. *Building Systems from Commercial Components*. Boston: Addison-Wesley, July 2001.

Krutchen, Phillippe. *The Rational Unified Process: An Introduction*, 2nd ed. New York: Addison-Wesley Object Technology. March 2000.

Albert, Cecilia; Brownsword, Lisa. *Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview*. Technical Report CMU/SEI-2002-TR-009.

About the Authors

Robert C. Seacord is a senior member of the technical staff at the SEI and an eclectic technologist. He is coauthor of the book *Building Systems from Commercial Components* as well as more than 40 papers on component-based software engineering, Web-based system design, legacy system modernization, component repositories, search engines, security, and user interface design and development.

Russ Bunting is a member of the technical staff in the COTS-Based Systems Initiative at the SEI. Before joining the SEI, Bunting led teams developing component management, requirements management, visual modeling, and code generation tools, and was a practicing Rational Certified Trainer in Object Oriented Analysis and Design\UML.

File Cabinets and Pig Latin: Guards for Information Assets

Larry Rogers

Think about your checkbook, your insurance policies, perhaps your birth certificate or passport, and other important documents and papers you have around your house. Where are they? Probably, they are stored in a filing cabinet or a safe that can be or is routinely locked. Why did you divide your information into the important and the unimportant, and then store the important items in a locked container?

Without realizing it, you were satisfying one of the three components of information security—confidentiality. Confidentiality attempts to keep secrets secret. Only those who are supposed to see the information in question should have access to it. You were keeping information sensitive to you and others away from those who should not be able to get to it. By the way, the other two components are integrity (Has my information changed?) and availability (Can I get at my information whenever I need it?).

You went beyond simply recognizing information confidentiality when you enforced it by using an access-control device, namely the locked filing cabinet or safe. This device stands between the information and those seeking access, and it grants access to all who have the combination, the key, or whatever tool unlocks the container. As a defense in-depth measure—where several layers of access-control devices are used—you may also find that those containers are themselves in locked rooms. Would-be intruders must pass through several levels of controls before finally gaining access to the information they seek.

Now, think of a computer system such as the one at home or in your office. The job here is to control access to files and databases, generally called information assets. The access control device is the access control list or ACL. ACLs define who can perform actions on an information asset and the actions that are allowed: reading and writing, for example. ACLs are the locked filing cabinet and safe equivalent for more traditional paper assets.

Different computer systems provide different types of ACLs. Some have fine-grained controls while others have virtually none. The key is to use all the controls that are available on your system. In some cases, you may have the choice of selecting which computer system you use to house your information assets. Select the system with the ACLs most appropriate to keeping your assets safe.

Frequently computer system vendors define ACLs that are overly permissive. This satisfies their need to ensure that access limitations don't get in the way of you using their systems. Your challenge is to tighten those ACLs so that they properly restrict access to only those who need access. This means that you need to do something to the ACLs guarding information assets on your computers when you buy them.

Returning to the home environment, do you remember when adults in your house wanted to say something to one another that the children shouldn't understand? They spelled their message or

used something like pig Latin (ig-pay atin-lay) to conceal the meaning of their conversation. This worked for a while, until the children learned to spell or could otherwise understand what was being said. What's really happening here?

Very simply, the adults could not control who could hear their conversation. It was inconvenient or perhaps impossible for them to go to another room where they couldn't be heard by anyone else. So they had to talk in a way that only those who knew the concealing scheme could understand what was being said.

On a computer system, when access to information cannot be limited, such as an e-commerce transaction carried out over the Internet, that information is concealed through a mathematical process called encryption. Encryption transforms information from one form (clear text) to another (cipher text). Its intent is to hide information content from those who have neither the transformation method nor the particulars (the decryption keys) needed to transform the cipher text back to its original clear text form. The cipher text is gibberish and remains so when you don't have the scheme or the keys.

Eventually, the children learned how to spell and also learned the ways of pig Latin. They could understand the conversations the adults were having. While they could also understand the conversations held weeks, months, or even years ago, the information in those conversations was no longer important. The encryption scheme was strong enough to guard the information during its useful lifetime.

Computer-based encryption schemes must also withstand the test of time. For example, if a credit card encryption scheme needs six months to break, the resulting credit card number is likely to be still valid and, therefore, useful to an intruder after that six-month period. In this case, the encryption scheme isn't strong enough to guard the information for its entire useful lifetime.

In summary, to guard information assets, be they paper or computer files, you need to limit who has access to them by using the access-control devices of filing cabinets, safes, and, on a computer system, access-control lists. For assets where access cannot be sufficiently limited, you need to encrypt them strongly enough so that the time it takes to decrypt them is longer than the useful life of the asset.

Appy-hay omputing-cay!

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.

Product Lines Are Everywhere

Paul C. Clements

Editor's note: A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. As Paul Clements wrote in the September 1999 issue of SEI Interactive (http://interactive.sei.cmu.edu /Features/1999/September/Spotlight/2-Spotlight.sep99.pdf):

> *Product lines promise to become the dominating production-software paradigm of the new century. Product flexibility is the new anthem of the marketplace, and product lines fulfill the promise of tailor-made systems built specifically for the needs of particular customers or customer groups. What makes product lines succeed from the vendor's (and developer's) point of view is that the commonalities shared by the products can be exploited to achieve economies of production.…But software product lines based on interproduct commonality are a relatively new concept, and the community is discovering that this path to success contains more than its share of pitfalls.*

In this new column, Clements, Linda Northrop, and other software product line experts at the Software Engineering Institute will contribute insights into how the software engineering community can reap the advantages of the product-line approach, while avoiding some of the pitfalls.

In his bestseller *Chaos: Making a New Science*, James Gleick related how some of the pioneers of chaos theory would, while relaxing in their favorite coffeehouse, compete to find the nearest example of a certain kind of chaotic system [1]. A flag whipping in the breeze, a dripping faucet, a rattling car fender—they seemed to be everywhere.

I can relate. Lately it seems that no matter where I turn I see a product line. At airports I see product lines of airliners (such as the Airbus A-318 A-319, A-320, and A-321, a family that ranges from 100 to 220 seats but clearly share production commonalities) powered by product lines of jet engines and equipped with product lines of navigation and communication equipment. When I arrive at my destination, I rent an American mid-size car that is always pretty much the same except for cosmetic factors and features, even though it could have one of four nameplates on it. I wonder how much more expensive the cars would be if they had nothing in common? The hotel leaves a copy of the local newspaper at my door: the morning edition of the city-wide version. Someone else will get the afternoon edition of the upstate version, but it will have most of the same stories, all of the same comics, and come off the same presses. On my way to work I

pass residential subdivisions where the houses are all variants on a few basic designs. Even the street signs are the same except for the names of the streets on them. While the actual street name is fundamental to a street sign's function, it is inconsequential to its fabrication and is just a variation point.

We know that product lines have been around in manufacturing almost since there was manufacturing. Remember Eli Whitney's idea of interchangeable parts for rifles in the early 1800s? They enabled a product line of firearms to be built that shared components. Remember the IBM System/360 family of computers? From the Principles of Operation:

> *Models of System/360 differ in storage speed, storage width (the amount of data obtained in each storage access), register width, and capabilities for processing data concurrently with the operation of multiple input/output devices. Several CPUs permit a wide choice in internal performance. Yet none of these differences affect the logical appearance of these models to the programmer. An individual System/360 is obtained by selecting the system components most suited to the applications from a wide variety of alternatives in internal performance, functional ability, and input/output (I/O).*

This was clearly a product line, and the operating system that powered it was a software product line. And town plans where the buildings look like each other pre-date post-War suburbia by at least eight centuries. During the Pei Sung dynasty of northern China (960-1127 AD) a book called the "Ying-tsao fa-shih" was written by Li Chieh, the state architect of the emperor Hui-tsung, in 1100 AD and published in 1103 AD. This was a set of building codes for official buildings. It described in encyclopedic detail the layout, materials, and practices for designing and building official buildings. It listed standard parts and standard ways of connecting the parts as well as recognizing and parameterizing variations of the parts such as allowable lengths, load capacities, bracketing, decorations, allowed components based on the building's purpose, and the options available for various component choices. The book also included design construction details that provided a process for building design and implementation of the design. While it was influential in spreading the most advanced techniques of the time of its first publication in 1103, by codifying practice it may also have inhibited further development and contributed to the conservatism of later techniques. Some scholars even claim that because of it, Chinese architecture remained largely unchanged until the beginning of the 20th century. (In a product line, you've got to know when your architecture has outlived its usefulness.)

But, like Gleick's scientists, I find some of the best examples of product lines in places where I go to eat. Here's something I saw on the menu at a little Mexican restaurant recently:

#16 Enchiladas verdes: Corn tortillas baked with a zesty filling, covered with a green tomatillo sauce. Your choice of chicken, beef, pork, or cheese.

#17 Enchiladas rojas: Corn tortillas baked with a zesty filling, covered with a red ancho chile sauce. Your choice of chicken, beef, or pork.

See what I mean? This restaurant clearly produces an “enchilada” product line. (Well, all right, “clearly” only applies to those of us who have been thinking about this for too long.) While admittedly a cheesy example—sorry—it actually provides a pretty good analogy with *software* product lines and the central concepts they embody.

The enchilada product line consists of seven separate products, differentiated by filling and sauce. This defines their variabilities. The corn tortillas are core assets because they’re used in every product. The red and green sauces are also core assets because they’re used in four and three products, respectively. And the meat fillings are also core assets, used in two products each. But the cheese is a product-specific asset, only used in the enchiladas verdes.

Some of the core assets have attached processes that indicate how they are to be instantiated for use in products. Here, the beef, pork, and chicken have attached processes that dictate how they’re chopped, seasoned, and cooked. The processes call for different spices to be added depending on the sauce.

All of the products share an “architecture”—tortillas wrapped around a filling, covered with sauce. And they also share a “production plan”—prepare filling, wrap filling in tortilla, cover with sauce, bake at 350 degrees for 15 minutes, garnish, serve.

This little product line provides economies of scope; the common ingredients let the restaurant stock a small number of food items delivered from a small number of suppliers. They provide personnel flexibility: the same person who makes the pork enchiladas rojas is, I would bet my house, the same person who makes the cheese enchiladas verdes. And by limiting the choices, many of the ingredients can be pre-prepared, allowing for rapid time-to-market, which in this case means time-to-table.

As a family, the products define a clear scope that leaves little doubt what’s in and what’s out. Chicken enchiladas are in. Beef enchiladas are in. And if you wanted cheese enchiladas with the red sauce instead of the green? Well, that’s probably open for discussion—a scope definition with a pronounced gray area is a healthy
thing—but duck enchiladas with a white sauce are definitely out.

Finally, by making the commonalities and variabilities exquisitely clear, it’s easy to see how this product line’s scope could be expanded—offer new fillings and new sauces and perhaps new combinations. You could even see how this efficient production capability could be used to launch an entirely new product line to capture a new market segment: Replace corn tortillas with flour tortillas, lose the sauce, add lettuce and tomato and other condiments, and open a new restaurant chain that sells “wraps.”

If you already had a strong grasp of the concepts underlying software product lines, then this little culinary diversion probably had no effect, except possibly to make you hungry. If you didn't, then perhaps the concepts are now a bit more palpable. In either case, the next time you're at a coffeehouse or restaurant, try looking around to see how many product lines you can spot.

¡Buen provecho!

References

[1] Gleick, J. *Chaos: Making a New Science*. New York, NY: Penguin Books, 1987, p. 262.

[2] Clements/Northrop, *Software Product Lines* (http://www.aw.com/catalog /academic/product/1,4096,0201703327,00.html) (pages 6-8, 38-40, 41-44, 46-48, 180-183, 195-197, 226-229, 264-272, 295-299, and 427-430). © 2001 Pearson Education, Inc. Reproduced by permission of Pearson Education, Inc. All rights reserved.

For Further Information

The Product Line Practice Initiative (http://www.sei.cmu.edu/plp)

A Framework for Software Product Line Practice (http://www.sei.cmu.edu/plp/framework.html)

Software Product Line Acquisition - A Companion to A Framework for Software Product Line Practice (http://www.sei.cmu.edu/plp/companion.html)

Product Line Technical Probe (http://www.sei.cmu.edu/plp/pltp.html)

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.

Learning from Hardware: Planning

Watts S. Humphrey

There has been a long-term effort to apply traditional engineering methods to software. While some portray these methods as the answer to software's many problems, others argue that they are rigid, constraining, and dehumanizing. Who is right? The answer, of course, is that the appropriateness of any method depends on the problems you are addressing. While highly disciplined methods can be bureaucratic and reduce creativity, a complete lack of structure and method is equally if not more damaging.

Engineering and Craftsmanship

One way to look at this issue is as a continuation of the craft versus engineering debate that has raged for over a century. In his recent paper, Kyle Eischen describes the long-running argument about individual craftsmanship versus structured, managed, and controlled engineering methods [1]. Before the advent of software, craft-like methods had always had a serious productivity and volume disadvantage. There simply were not enough skilled craftsmen to meet society's demands for quality goods and services.

Factories were developed as a way to meet the need for volumes of quality goods, and manufacturing plants continue to serve these same needs today. The objective, of course, was not to destroy the crafts but to devise a means for using large numbers of less-expensive workers to produce quality products in a volume and for a cost that would satisfy society's needs. However, the widespread use of factories has required orderly processes and products that were designed to be economically manufactured in volume. This has led to many of today's engineering practices.

While these volume-oriented engineering methods have generally been effective, they have also often been implemented improperly. This has caused resentment and has reduced engineering efficiency and produced poorer quality products. However, as Deming, Juran, and many others have pointed out, this is not because of any inherent problem with engineering methods but rather with how these methods have been applied [2, 3, 4, 5, 6].

The Software Problem

As far as software is concerned, our current situation is both similar and different. The need again is to economically produce quality products in volume. Further, since the volume of software work is increasing rapidly, there is a need to use larger numbers of workers. While this might imply the need for factory-like methods that use lower skilled people, this cannot be the case with software. This is because software is highly creative intellectual work.

The push toward more of an engineering approach for software is not caused by a shortage of skilled people. Even though we have periodically had programmer shortages, these shortages have not been caused by a lack of potential talent. There appears to be an almost unlimited supply

of talented people who could be trained for software jobs, if given suitable incentives. The push for engineering methods comes from a different source, and it is instructive to examine that source to see why we face this craft-engineering debate in the first place.

The Pressure for Improvement

The source of pressure for software process improvement is the generally poor performance of most software groups. Products have typically been late, budgets have rarely been met, and quality has been troublesome at best. From a business perspective, software appears to be unmanageable. Since software is increasingly important to most businesses, thoughtful managers know that they must do something to improve the situation.

From a management perspective, software problems are both confusing and frustrating. Businesses require predictable work. While cost and schedule problems are common with technical work, most engineering groups are much better than we are at meeting their commitments. Senior managers can't understand why software people don't also produce quality products on predictable schedules and with steadily declining costs. They need these things to run their businesses and they expect their software groups to be as effectively managed as their other engineering activities. This management unhappiness spans the entire spectrum from small one- or two-person software projects to large programs with dozens to hundreds of professionals.

Software Background

While the performance of software projects has been a problem for decades, software people have not historically applied traditional engineering methods. They have not typically planned and tracked their work, and their managers often either didn't believe the software problems were critically important or they didn't know enough about software to provide useful guidance.

This situation is now changing, and the pressure for better business results is causing the software community to apply the principles and practices that have worked so effectively for other engineering groups. Among the most important of these practices is project planning and tracking. Therefore, the pertinent questions are

> Do engineering planning and tracking methods apply to software?
> If they do, need they be rigid and constraining?

To answer these questions, we need to look at why planning and tracking were adopted by other engineering fields and to consider how they might be used with software.

Plan and Track the Work

For any but the simplest projects, hardware engineers quickly learn that they must have plans. The projects that don't have plans rarely meet their schedules and, during the job, nobody can tell

where the project stands or when it will finish. On their very first projects, most hardware engineers learn to make a plan before they commit to a schedule or a cost. They also learn to revise their plans every week if needed and to keep these plans in step with their current working situation. When engineering groups do this, they usually meet their commitments.

Some years ago, I was put in charge of a large software group that was in serious trouble. Their current projects had all been announced over a year earlier, and the initial delivery dates had already been missed. Nobody in the company believed any of the dates, and our customers were irate. The pressure to deliver was intense.

When I first reviewed the projects, I was appalled to find that no one had any plans or schedules. All they knew was the dates that had been committed to customers, and nobody believed them. While everyone agreed that the right way to do the job would be to follow detailed plans, they didn't have time to make plans. They were too busy coding and testing.

I disagreed. After getting agreement from senior management, I cancelled all the committed schedules and told the software groups to make plans. I further said that I would not agree to announce or ship any product that did not have a plan. While it took several weeks to get good plans that everyone agreed with, they didn't then miss a single date. And this from a group that had never met a schedule before.

The Key Questions

If planning is so effective for everybody else, why don't software people plan? First, software people have never learned how to make precise plans or to work to these plans. They don't learn planning in school, and the projects they work on have not generally been planned. They therefore don't know how to plan and couldn't make a sound plan if they tried. Second, nobody has ever asked them to make plans. When plans are made in the software business, the managers have typically made them and the engineers have had little or nothing to do with the planning process. The third reason that software people don't plan is that, without any planning experience, few software people realize that planning is the best way to protect themselves from unrealistic schedules. The fourth reason is that management has been willing to accept software schedule commitments without detailed plans. When management realizes the benefits of software plans, they will start demanding plans and then, whether we like it or not, software people will have to plan their work.

The Answers

So the answer to the first question, "Do these engineering methods apply to software?" is a clear and resounding yes. The answer to the second question, "Are these engineering methods really rigid and constraining?" depends on how the methods are introduced and used. Any powerful tool or method can be misused. The guideline here is this: Does the method's implementation assume that some higher authority knows best, or is the method implemented in a way that requires the agreement and support of those who will use it?

Any method that requires unthinking obedience will be threatening and dehumanizing to some, if not to all, of the people who use it. This is true whether the method requires you always to plan, refactor, or document, just as much as if the method requires that you never plan, refactor, or document. All methods have costs and advantages, and any approach that dictates how always to do something is rigid and constraining. The key is to learn the applicable methods for your chosen field, to understand how and when to use these methods, and then to consistently use those methods that best fit your current situation.

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, Julia Mullaney, Bob Musson, and Marsha Pomeroy-Huff.

In closing, an invitation to readers

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

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

References

[1] Eischen, Kyle. “Software Development: An Outsider’s View.” *IEEE Computer 35*, 5 (May 2002): 36–44.

[2] Crosby, Philip B. *Quality is Free, The Art of Making Quality Certain.* New York: Mentor, New American Library, 1979.

[3] Deming, W. Edwards. *Out of the Crisis.* Cambridge, MA: MIT Center for Advanced Engineering Study, 1982.

[4] Humphrey, W. S. *Managing the Software Process.* Reading, MA: Addison-Wesley, 1989.

[5] Humphrey, W. S. *Winning with Software: An Executive Strategy.* Reading, MA: Addison-Wesley, 2002.

[6] Juran, J. M. & Gryna, Frank M. *Juran’s Quality Control Handbook, Fourth Edition.* New York: McGraw-Hill Book Company, 1988.

Accelerating CMMI Adoption with Technology Adoption Tools

Suzanne Garcia

For those making the transition to Capability Maturity Model® Integration (CMMI®) from another process improvement model or methodology, understanding CMMI adoption as a technology adoption and applying technology-adoption concepts can smooth the process considerably and provide strategies that can be applied when implementing other new technologies.

CMMI Adoption as Technology Adoption

What is technology adoption? Generally, it is the set of practices and factors related to organizations selecting, deploying, and sustaining the use of a technology. Why look at CMMI adoption as *technology* adoption? First of all, CMMI *is* a technology—a *process* technology—and what's more, it's *radical*. "Radical innovation is the process of introducing something that is new to the organization and that requires the development of completely new routines, usually with modifications in the normative beliefs and value systems of organization members."[1] Treating CMMI as a technology adoption activates a different mindset than the one typically applied to process improvement and enables CMMI adopters to benefit from some of the tools and concepts of technology adoption described below.

Dealing with Dimensions

People whose organizations have successfully implemented the Capability Maturity Model for Software (SW–CMM®) may think they can simply apply the same transition strategies in implementing CMMI. While there are similarities between the SW–CMM V1.1 and the CMMI Framework, CMMI provides an opportunity to expand the scope of application of CMM concepts beyond just the software organization into the other parts of the organization involved in product or service development. This means involving new players in the CMM adoption and expanding the scope of effect of CMMs on the subsystems of the organization. The CMMI adoption effort should include strategies for dealing with different audiences with different needs at any one time. One way to address this is to ensure that the engineering process group or equivalent has adequate representation from all the stakeholders in CMMI, not just the experienced software engineering process group members from the software part of the organization.

---

[1] Schein, Edgar. "The Three Cultures of Management: Implications for Organizational Learning." *Sloan Management Review 38*, 1 (Fall 1996): 9-20

Understanding the Audience

Who in the organization has to change something in their behavior, attitudes, or values to adopt CMMI? Executives, managers, technology users, support groups? Distinct factors will have to be addressed for each organizational subculture in developing an adoption strategy.

Within subculture groups, *individuals* also differ in their responses to a technology adoption. Different "adopter types" move through adoption at different speeds. These groups are distinguished from each other by their characteristic responses to an innovation (either process or technology) that requires a change in their behavior. Figure 1 illustrates where each adopter category falls in the technology adoption life cycle.[1]

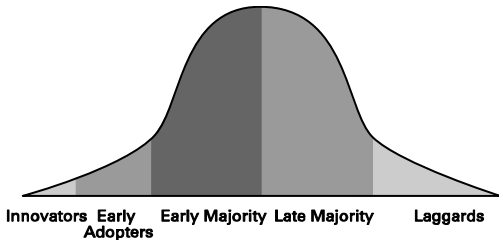


Figure 1: The Technology Adoption Life Cycle

Adopter types can be used in planning who should get the technology when and in determining which kinds of adoption–support mechanisms are likely to be successful. For example, if a pilot for new practices affects a group composed primarily of late-majority or laggard participants, the pilot is much more likely to succeed if (a) a completely packaged solution is provided and (b) adoption of the new practices is mandated by the organization, with sanctions for not adopting.

A brief description of the classic adopter types from Everett Rogers's research[2] can be found in *The Road to CMMI: Results of the First Technology Transition Workshop* on the SEI Web site.[3] Geoffrey Moore's *Inside the Tornado* describes their use in high–tech marketing.[4]

---

1 Moore, Geoffrey A. Crossing the Chasm. New York, NY: Harper Collins, 1991.

2 Rogers, Everett. Diffusion of Innovations. New York, NY: The Free Press, 1995.

3 http://www.sei.cmu.edu/publications/documents/02.reports/02tr007.html

4 Moore, Geoffrey A. *Inside the Tornado: Marketing Strategies from Silicon Valley's Cutting Edge*. New York: HarperCollins, 1995.

Diffusion vs. Infusion

In considering a process technology adoption such as CMMI, some time should be spent determining the goals of the adoption for the different roles within the organization. Some of the other concepts described above, such as what kinds of adopters the adoption is targeting, and what elements of the organization need to be realigned, can help in setting some of these goals. Another area that should be considered is the relative emphasis that will be placed on CMMI *diffusion* (how widespread the use of CMMI will become) versus CMMI *infusion* (how deeply embedded into the organizational infrastructure CMMI will become).[1] An emphasis on diffusion may get broad acceptance and knowledge of CMMI within the organization, but may not achieve the differences in behavior that actually contribute more heavily to improved business results. With increasing infusion, the degree of workflow interconnectedness related to CMMI use increases, and the degree of visibility of the technology increases within the management and oversight structures of the organization, usually leading to more permanent behavior changes that have a positive result on return on investment.

To measure infusion, one can measure "levels of use" of a technology. For example, the evolution of the infusion of CMMI use in an organization might look something like this:

1. CMMI adoption has occurred in a few projects whose local procedures and processes have been changed to reflect the new practices.

2. One of the divisions of the organization has changed its policies to reflect the practices recommended in CMMI and has formulated and published a set of standard process assets that are used as the basis for initiating and managing new product development projects.

3. Reward and incentive systems in the new projects adopting CMMI practices have been examined and changed where necessary to encourage productive use of the new processes. Existing projects within the division have been evaluated to determine which parts of the set of standard process assets might beneficially be applied to the projects at their current point in the life cycle, and the projects are being provided the training and other support needed to make it feasible for them to adopt new practices in mid-project.

4. Members of projects in the division adopting CMMI are being recruited for projects in other parts of the organization due to the reputation of the projects for meeting customer expectations; however, many of them choose to stay within the division rather than move to the other parts of the organization that are less disciplined in their management and engineering practices.

Each of these scenarios could be considered a "level of use" measure for the infusion of CMMI adoption within the organization. With increasing levels of use, the degree of workflow interconnectedness related to the CMMI use increases, and the degree of visibility of the technology within the social subsystem is increased, as exemplified in the fourth scenario.

---

[1] Zmud, R.W. & Apple, L.E. "Measuring Technology Incorporation/Infusion." *Journal of Product Innovation Management 9*, 2 (June 1992): 148-155.

Measuring diffusion at the start of a CMMI adoption would result in an organizational profile similar to that shown in Figure 2. As time goes on, the profile should shift to something like that shown in Figure 3 as more and more members of the organization participate in the activities of CMMI adoption. One use of this measure is to help senior managers understand the time needed to see tangible return on investment of a CMMI implementation. When they understand how many people have to go through several events before one can expect their behavior, and therefore their results, to change, it can help them tolerate some of the time lag that is typical between starting an adoption effort and seeing business results.

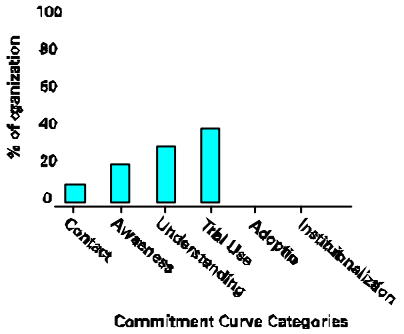


Figure 2: Notional Profile Early in Adoption

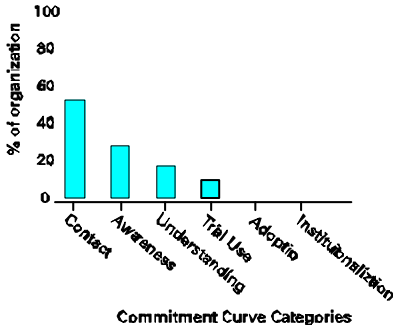


Figure 3: Notional Profile Later in Adoption

Transition Mechanisms

The SEI has been using a variation of the Patterson–Conner commitment curve for years to help organizations understand both the communication and implementation planning needed to ensure that a change is fully adopted (see Figure 4).[1] It is also a useful framework for categorizing and understanding the types of transition mechanisms that are needed to help individuals and groups within an organization to progress in their adoption of a new technology. Transition mechanisms

[1] Conner, Darryl R. & Patterson, Robert W. "Building Commitment to Organizational Change." *Training and Development Journal 36*, 4 (April 1982): 18-30.

are products, activities, events, and methods that help accelerate progress from one commitment milestone to another (from awareness to understanding, for example).

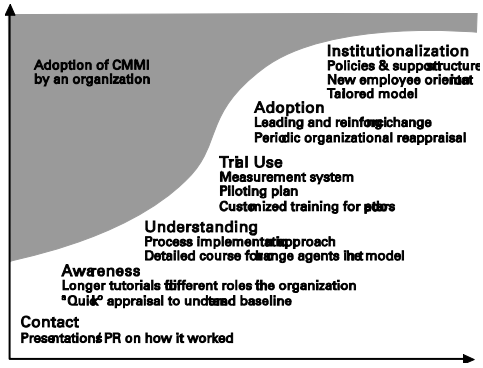


Figure 4: The Patterson-Connor Commitment Curve

As the adoption proceeds, the mechanisms move in character from communication and education more toward implementation support and incentives management. Mechanisms used in early stages might include CMMI reference cards and mappings of other models that the organization is using to CMMI. Later, mechanisms might include CMMI tailoring guidance for specific organizational contexts.

Timing of transition mechanisms is critical. Showing someone the measurements to be used to monitor the detailed implementation of CMMI before they even understand what parts of the organization are affected by the model is an example of a good transition mechanism being used too early. Conversely, waiting until everyone in the organization has been through Introduction to CMMI training before management communicates its vision of how CMMI will fit into the overall business strategy is an example of a good transition mechanism being used too late to be effective. For a list and discussion of transition mechanisms that early adopters of CMMI have used (and for their opinion on what types of transition mechanisms are also needed for CMMI to be successful), see *The Road to CMMI: Results of the First Technology Transition Workshop*.

Communities of Practice

Once problems are being solved with a technology, the possibility exists to seed a community of practice, which contains members of the organization who are motivated to continue learning about the technology and its implementation. They might build "translations" of the technology for other users who may not be as far along in their adoption of the technology, and communicate and solve problems with each other to improve their use of the technology.

In CMM adoption history, many of the Software Process Improvement Networks (SPINs) exhibit characteristics of communities of practice. Bringing the ideas of continued learning and involvement by the practitioners and change agents inside the organization can accelerate the

adoption of CMMI, since this approach tends to access the informal networks of influence that exist within the organization outside the normal organizational structure.

Summary

There are many approaches from the technology–adoption arena that can be useful in making effective use of CMMI as it matures. A few of these have been highlighted in this article that either are "classics" worth repeating or that reflect some of the newer practices that the SEI is exploring as part of its research. Early adopters of CMMI should be prepared to invest in creating the transition mechanisms their organizations will need to be successful and to apply creative approaches to making progress. Understanding and applying technology–adoption concepts can help maximize return on investment by adding to tools already in place for an improvement effort.

Editor's Note:

Portions of this article were originally published in the March 2002 issue of CrossTalk (http://www.stsc.hill.af.mil/crosstalk/).

For more information, contact—

Suzanne Garcia

Phone
412-268-9143

Email
smg@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu/asta

Carnegie Mellon Educates Next Generation of Information-Security Experts

Kelly Kimberland

Carnegie Mellon University is working on a program designed to increase the number of information-security experts in the workforce. Stephen Cross, director of the SEI, explains that the program, which is funded by the National Science Foundation (NSF), provides participants with the knowledge and expertise to develop and deliver curricula in information security. The program is intended to increase the number of PhD-level researchers in information security at historically black colleges and universities and Hispanic-serving institutions.

"Creating a more diverse workforce, both within the SEI and within the software engineering community as a whole, is one of our top priorities at the SEI," Cross says. "We are very excited to be partnering with these educational institutions. The training and experiences shared in this program lay the foundation to help create a new generation of Internet-security experts who will help assure the protection of our information infrastructure."

The need for qualified information-security personnel and educators is great. A June 1999 Department of Commerce Report, "The Digital Workforce," estimates that the United States will require more than 1.3 million new highly skilled information technology workers between 1996 and 2006. The National Plan for Information Systems Protection also identifies this critical shortage and further highlights the acute shortage in the number of trained information-security personnel. The National Plan recognizes training and education as key solutions in defending America's cyberspace.

Participants from Howard University, Morgan State University, and the University of Texas at El Paso gathered recently in Pittsburgh, PA, to acquire knowledge and educational resources to teach survey-level courses in information security to advanced undergraduate and first-year graduate students at their universities.

The courses were delivered by staff of the SEI and CERT® Coordination Center®, the nation's first and best-known computer emergency response team. Other distinguished faculty members from Carnegie Mellon's H. John Heinz III School of Public Policy and Management (Heinz School), School of Computer Science, and Department of Electrical and Computer Engineering also participated.

During the first two weeks, participants received basic instruction and training in information security, including discussion of how information security intersects with other academic disciplines. The third week was devoted to curriculum development. The participants worked with instructional-design experts from the SEI and with Dr. Corey Schou, director of the National Information Assurance Training and Education Center (NIATEC) at Idaho State University and chairperson of the National Colloquium for Information Systems Security Education. The final week of the program was devoted to presentation of current and future research by the

participants and to the development of research collaborations between participants and researchers at Carnegie Mellon.

One of the participants, Dr. Wayne Patterson, is a senior fellow at Howard University's graduate school and professor of computer science with the school's Department of Systems and Computer Science in the College of Engineering, Architecture, and Computer Science. Patterson found the program very beneficial. He says that the Carnegie Mellon program meshed well with his department's goal of developing a PhD program in information security. "This program will help us develop a computer- security emphasis in our doctoral program," Patterson says. "We [the participants] are all committed to looking for joint collaboration in research and curriculum development. We are all interested in continuing to move this forward." He adds that the four universities are developing a proposal to submit to the NSF to fund curriculum and research development.

The NSF funding paid for the participants' salary, plus lodging, per diem, and incidentals. Additionally, three round trips to Pittsburgh and two additional trips during the 2002-2003 academic year will enable the participants to build relationships and continue research collaborations and mentoring.

In May 1999, the National Security Agency (NSA) designated Carnegie Mellon as a Center of Academic Excellence in Information Assurance. The NSA established the Centers of Academic Excellence in Information Assurance Education Program to increase the capacity of U.S. higher education institutions to produce professionals in this field. This program is an example of the outreach and partnership efforts called for in the National Plan for Information Systems Protection.

Donald J. McGillen, executive director of Carnegie Mellon's Center for Computer and Communications Security, is the coordinator of Carnegie Mellon's activities as a Center for Academic Excellence in Information Assurance Education. "With the availability of a great resource like the CERT Coordination Center, along with distinguished Carnegie Mellon faculty members, Carnegie Mellon is uniquely qualified to help other institutions develop new programs and expand existing programs in information security," McGillen says. "The expertise and knowledge that the CERT/CC has developed since 1988 provide an unequaled level of quality, relevance, and credibility to both degree-based and executive programs."

In addition to funding this program, the NSF also competitively awarded grants to six of the Centers of Academic Excellence to fund scholarships for students who enroll in programs in information security and, upon graduation, enter service with a government agency as members of the Federal Cyber Corps. Currently, 18 students are attending Carnegie Mellon on these scholarships.

For more information, contact—

Dr. Donald J. McGillen,
Executive Director of the Center for
Computer and Communication Security

Phone
412-268-6755

Email
dmcgille@sei.cmu.edu

E. Carter Jones,
Manager of Diversity Outreach Programs for the SEI

Phone
412-268-7724

Email
ecj@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu

Software Architecture Book Provides Practical Guidance about Documentation

Erin Harper

The construction of a successful system depends on an appropriate architecture—the way the system is decomposed into parts and the ways in which those parts interact. Yet even the best architecture is useless if others cannot understand it, making the documentation of software architectures crucial.

As part of the SEI Series in Software Engineering,[1] a new book called *Documenting Software Architectures: Views and Beyond* has been written by Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford.

Intended as a handbook for practitioners in the field, this book helps readers decide what information about an architecture is important to document and provides guidelines, notations, and examples for documenting that information. The authors provide an extended example of a software-architecture documentation package that demonstrates the concepts covered in the book. "Documenting an architecture is a formidable task without firm guidance," Stafford says. "We break it down into three basic viewtypes, then we allow refinements, called *views*, that are based on architectural styles."

Because the same system can be seen in many different ways, choosing the right views depends on who the audience will be and what they will need to do with the information. "Different stakeholders don't have the time or energy to sift through a lot of unnecessary information," Stafford remarks. For example, a project manager may be interested in the system's overall purpose and constraints, but not in the detailed design. Members of the development team, on the other hand, may be given responsibility for elements they did not implement, and will need to know the details of those elements and how they interact.

A simple three-step process for choosing the best views to document for a particular system is included in the book, along with detailed templates for practitioners to use for documenting interfaces, views, and additional information beyond the views.

Documentation beyond views consists of three major aspects, which the authors summarize as "how/what/why": *how* the documentation is organized (consisting of a roadmap and a view template); *what* the architecture is (consisting of a short system overview); and *why* the architecture is the way it is (consisting of background information, external constraints, and the rationale for decisions). All of these aspects of documentation are important.
"Documentation speaks for the architect, today and 20 years from today," says Clements. Many

---

[1] http://www.sei.cmu.edu/products/publications/sei.series.html

people may need to look at the documentation after a system is created: those maintaining or updating the system, those trying to expand it, or those building similar systems.

Although many individuals and organizations are beginning to understand the importance of software architecture, there is little practical guidance on how to capture an architecture independent of language or notation. “Box-and-line drawings aren’t enough. They’ve been masquerading as architecture for years now, but they need to be supported by additional information and explanations,” Clements says. Based on their years of research and experience in the field, the authors provide needed guidance on creating a complete documentation package.

More information about this book can be found on the SEI Web site at http://www.sei.cmu.edu/ata/books.html.

For more information, contact—

Paul Clements

Phone
512-453-1471

Email
pclement@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu/ata/books.html

New Book Helps Organizations Take Charge of Information Security

Erin Harper

Most organizations today store their information electronically and share it over networked systems, making the protection of that information more complex than ever. Information security requires more than buying the latest tool or hiring a consultant to evaluate the security of systems.

A new book in the SEI Series in Software Engineering, *Managing Information Security Risks: The OCTAVE[SM] Approach*, provides a complete and systematic approach to evaluating and managing information-security risks. The book was written by Christopher Alberts and Audrey Dorofee, SEI staff members, and the principle developers of the Operationally Critical Threat, Asset, and Vulnerability Evaluation[SM] (OCTAVE) approach. The book helps organizations learn about the OCTAVE approach by providing evaluation work-sheets, a catalog of best practices, and examples based on the authors' experiences with real organizations.

The OCTAVE approach puts organizations in charge of their own security, which Alberts and Dorofee say is critical to the success of any security program. "We did an evaluation for an organization in the past to identify their security risks, and we presented them with our results, but they never took action. Once the experts leave, people often go back to what they were doing before," Alberts says. When the same organization later used the OCTAVE approach, they did make changes. "Because they found the problems themselves, someone within the company took ownership of the situation," Alberts says.

Getting everyone involved in security is also an important key to success. "A lot of organizations delegate security to their information technology (IT) department, and assume everything will be taken care of, but the IT department may not understand the organization's business-related needs and priorities," Dorofee explains. "Organizations need to stop looking at security as a technology problem, and begin to look at it as a business practice."

There is a tradeoff between the services your organization chooses to offer and the security risks that develop. "For example, you may offer ordering over the Web, which might help you get more business, but it also exposes you to more threats," Alberts says.

These tough decisions make the participation of senior management imperative. "We acknowledge that we live in the real world with limited resources. Managers have to ask, 'Where do I want to put the few dollars that I have for security?'" Alberts notes. The OCTAVE approach can help organizations decide which assets to protect through their systems for ranking and identifying key assets. Using OCTAVE's catalog of security practices to protect critical assets then causes security benefits to cascade down through the organization.

Protection, however, is only one element of information-security risk management. Monitoring systems and developing mitigation strategies for use in the event of a security breach are also key elements covered in the book. "You can never say, 'I am 100% secure.' You need to ask yourself

what happens to your customers, your finances, and your reputation if there is a security breach," Alberts says. Using the OCTAVE approach, business units and IT departments can work together to develop a complete security strategy based on their organization's business concerns.

More information about the OCTAVE approach is available at *http://www.cert.org/octave/*.

For more information, contact—

Bob Rosenstein

Phone
412-268-8468

Email
br@sei.cmu.edu

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