Carnegie Mellon
Software Engineering Institute

news@sei

http://www.sei.cmu.edu/news-at-sei/

Editor’s Message

In August Dr. Paul Nielsen joined the SEI as Director and Chief Executive Officer. Most recently, Dr. Nielsen was the Air Force’s technology executive officer and determined the investment strategy for all Air Force science and technology activities. You’ll find a full interview with Dr. Nielsen, but a comment he makes on the importance of transition activities seems particularly apt for this issue of *news@sei*. He says, “For every research organization, an engineer’s work is not done until it transitions, until it makes people’s lives better. Sometimes this is forgotten, and sometimes it’s hard to pull off. A big portion of the job has to be developing new technologies, but transition, making sure it gets out there, is critical.” Getting the word out, and making sure the work of the SEI brings specific and concrete benefits to our customers, is the topic of the three other feature articles in this issue of *news@sei*.

In 2002, the U. S. Army’s acquisition executive, Claude Bolton, asked the SEI to help with a multi-year program to improve the way the Army acquires software-intensive systems. To determine how to improve acquisition processes, the SEI first set out to determine the current state of the practice within Army acquisition. Benchmarking for Improvement, or BFI, was used by the SEI to identify improvement needs across Army acquisition. The article “Benchmarking for Improvement in Army Acquisition” discusses BFI and some of the emerging results from its application on Army programs.

“CMMI for Small Businesses: Initial Results of the Pilot Study” describes a program begun in July 2003 as part of a joint project between the SEI and the U. S. Army. Initial results from the pilot look promising: participating organizations described significant benefits from using the Capability Maturity Model® Integration (CMMI®), especially in the areas of project management and change management, and now the SEI is in the process of documenting and disseminating these findings so that others can learn from these experiences.

“Making the Use of the DoDAF Easier for DoD Organizations” covers a different kind of transition effort. The Department of Defense Architecture Framework (DoDAF), which is mandated by the DoD for large-scale systems, describes how the architecture for a system or system of systems should be documented. Because using any framework for the first time can be difficult, the SEI set out to discover how DoD organizations and their contractors can make the process easier.

Researchers at the SEI did this in two ways. First, they held a workshop with representatives from government, industry, and academic institutions. Second, they conducted interviews with architects, designers, architecture documentation reviewers, and program managers who are associated with acquisition programs.

Thanks for reading *news@sei*, another kind of transition effort at the SEI. If you have questions or comments about what you see here, let me know at *news-editor@sei.cmu.edu*.

Janet Rex
Editor-in-Chief

The Architect

Integrating Architecture Methods: The Case of the Rational Unified Process

RICK KAZMAN, ROBERT L. NORD

In a previous column (“Rethinking the Software Life Cycle,” DATE), we took a look at the traditional software-development life cycle in the context of the architecture-centric methods that we have developed at the Carnegie Mellon® Software Engineering Institute (SEI) over the past 10 years. These methods include the Architecture Tradeoff Analysis Method® (ATAM®) [Clements 02], the SEI Quality Attribute Workshop (QAW) [Barbacci 03], the SEI Attribute-Driven Design (ADD) Method [Bass 03], the SEI Cost Benefit Analysis Method (CBAM) [Bass 03], and SEI Active Reviews for Intermediate Design (ARID) [Clements 02]. This column shows how these architecture-centric methods fit into the framework of the Rational Unified Process (RUP).

The SEI’s architecture-centric methods were developed at the same time that the RUP was being developed. The RUP is an object-oriented development framework. It provides guidelines, templates, and examples for all aspects and stages of a software-intensive system’s life cycle, although it treats software architecture obliquely.

The SEI’s architecture-centric methods have long demonstrated that they can illuminate important characteristics of architectures and the quality-attribute requirements that shape them. Until now, such considerations have been relegated to a separate “supplementary requirements” document in the RUP. Also, business drivers, long a key part of SEI methods, have just recently found a place in the RUP.

The SEI architecture-centric methods can provide explicit and detailed guidance on eliciting the architectural requirements, on designing the architecture, and on analyzing the resulting design.

- The architecture-centric methods place an emphasis on quality attributes rather than functionality.
- The architecture-centric methods help fill gaps in the RUP design process by providing specific advice on

  -the elicitation and documentation of quality-attribute requirements
  -which design operation will achieve a desired quality-attribute response
  -how to understand and predict the consequences of the design decisions in terms of risks, tradeoffs, and ultimately return on investment

2 The architecture-centric methods all use common concepts: quality attributes, architectural tactics, and a “views and beyond” approach to documentation that leads to increasingly efficient and synergistic use [Clements 03].

Table 1 shows where specific SEI architecture-centric methods can help to produce artifacts required in different RUP phases, or how the methods can enhance the activities of the RUP. More details are available in a forthcoming technical report [Kazman 04].

Table 1: The Architecture-Centric Methods as RUP Activities

Method	Role	Discipline	Workflow Detail	Artifacts Affected
QAW	System analyst	Requirements	Understand Stakeholder Needs	Business case Supplementary specifications
ADD	Software architect	Analysis & Design	Define a Candidate Architecture Perform Architectural Synthesis	Software architecture document
ATAM/ CBAM	Technical reviewer	Analysis & Design	Refine the Architecture	Review record Software architecture document
ARID	Technical reviewer	Analysis & Design	Refine the Architecture Analyze Behavior	Review record

Through the process of the QAW, vague requirements would be refined into several quality-attribute scenarios. The ADD Method defines a software architecture by basing the design process on the quality-attribute requirements of the system. The ADD approach follows a recursive decomposition process where, at each stage in the decomposition, design decisions are made to satisfy a chosen set of high-priority quality scenarios.

It is clear that design decisions interact. For this reason, we need an organized method for understanding the interaction of the many decisions that are made in creating a complex system architecture. The ATAM provides software architects with a framework for understanding the technical tradeoffs and risks that they face when making architectural design decisions. In addition, the CBAM helps software architects consider the return on investment of any architectural decision and provides guidance on the economic tradeoffs involved. Finally, the ARID evaluates whether the design can be used by the software engineers who must work with it.

The benefit of including the SEI methods is to address quality attributes in an explicit, methodical way. Quality-attribute requirements drive the software architecture, and architecture-centric activities (with an explicit focus on quality attributes) drive the software system life cycle. Properly managed, the architecture-centric methods can be a low-cost addition to the RUP that will increase the quality of the systems and products developed.

References

[Barbacci 03]	Barbacci, M. R.; Ellison, R.; Lattanze, A. J.; Stafford, J. A.; Weinstock, C. B.; & Wood, W. G. *Quality Attribute Workshops (QAWs), Third Edition* (CMU/SEI-2003-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html.
[Bass 03]	Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice*, 2nd edition. Boston, MA: Addison-Wesley, 2003.
[Clements 02]	Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2002.
[Clements 03]	Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison-Wesley, 2003.
[Kazman 04]	Kazman, R.; Kruchten, P.; Nord, R.L.; Tomayko, J.E. *Integrating Software Architecture-Centric Methods into the Rational Unified Process* (CMU/SEI-2004-TR-011). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004. http://www.sei.cmu.edu/publications/documents/04.reports/04tr011.html.

About the Authors

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

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

CMMI in Focus

CMMI—V1.2 and Beyond

MIKE PHILLIPS

My last two columns have focused on appraisal issues (“myths” and “due diligence”), and it is a good time to update you on where we see ourselves going with the CMMI Product Suite. As many of you know, there is a CMMI Steering Group that controls CMMI Product Suite content. In Steering Group meetings, the future plans for CMMI are debated and approved. We are now moving into the initial phases of work on the next product suite revisions.

What changes will be in V1.2?

The upcoming revision to the CMMI Product Suite is called Version 1.2. The theme for the Version 1.2 update is “one book; one course.” The “one book” approach is based on the “blue book” in the Addison-Wesley series, *CMMI: Guidelines for Process Integration and Product Improvement*. The book’s authors (also CMMI Version 1.2 development team members) found that it was possible to combine both the continuous and the staged representations of the models into a single book—an improvement over the two separate representations released as technical reports on the SEI Web site for Version 1.1. This hardbound book version of CMMI models will serve as the basis for our update to Version 1.2.

While some have desired resolution to a single representation, we think that both representations have shown value for different purposes. The staged representation provides a roadmap for organizational improvement; the continuous representation provides the ability to focus selective attention on specific problem areas and treat them effectively. So a book that contains both approaches together recognizes these complementary objectives for model use.

As for the course approach, by fall of this year, we will be piloting a single *Introduction to CMMI* course that includes coverage of both the staged and continuous representations. The original approach of having two separate courses, one for staged and one for continuous, made sense initially because it helped users to make the transition from other models. Users of the Software CMM were more comfortable attending the staged course, and users of standards such as EIA 731 and SPICE were more comfortable attending the continuous course.

However, now that CMMI has been available for nearly four years, those that attend the *Introduction the CMMI* courses tend to be new to process improvement and have little or no experience with legacy models or standards. Their inexperience makes their choice of a course difficult. Students are uncertain about what a representation is, much less which representation to choose.

Additional changes to the CMMI Product Suite are currently being considered, such as further integrating supplier sourcing practices into Software and Systems Engineering model practices and other simplifications.

What are you planning to do to simplify the model?

Both users and developers have sought ways to simplify the model from its inception. We have plans to make changes that will simplify CMMI models for Version 1.2. First, we think that combining the two representations into one document for each model reduces complexity. We have been pleased to find that the book version has been well received by CMMI users and, believe it or not, has held the total number of model pages (as seen in the comparable technical reports) to a little over 700 pages.

We also plan to modify the architecture of CMMI models to eliminate common features and advanced practices. These two types of model components were passed down from the source models. Advanced practices are a type of model component that came from EIA 731. Common features are a type of model component that came from the SW-CMM. While each of these has had value in its source model, their value in CMMI is marginal, and their removal effectively simplifies the CMMI models. In addition to simplifying the numbering scheme, a few of the engineering practices will be consolidated, specifically those that are subsumed by a more advanced practice within the relevant engineering process area. Generic practices will continue as the main measure of capability levels but will no longer be characterized by their common feature identifier.

While no final decisions have been made, we are also investigating other consolidations based on change requests. One of these is the integration of the two supplier-related process areas—Supplier Agreement Management (SAM) and Integrated Supplier Management (ISM). This consolidation would not only eliminate one process area from the CMMI mix, but also would simplify the number of model choices by eliminating the supplier sourcing (SS) discipline as a separate choice when selecting a model. Such a change might also be perceived as increasing the difficulty of achieving maturity level 2, but is still worthy of consideration since supplier sourcing is not a true discipline like systems engineering and software engineering.

What kinds of clarifications are envisioned?

Thus far, we have organized the change requests by process area. While all process areas have change requests associated with them, the engineering and IPPD process areas have received the greatest attention. Therefore, these process areas are likely to change more than others for Version 1.2. In addition, we plan to clarify the importance of achieving customer satisfaction in the informative material. Those who are familiar with the emphasis on the “voice of the customer” in Six Sigma will recognize the importance of clarifying this coverage that was always intended as part of CMMI best practices.

Are there any expansions planned?

Many of the suggestions for expansion will require clear sponsorship to expand CMMI model coverage. Such sponsorship is described in the CMMI Concept of Operations, viewable on the CMMI Web site at http://www.sei.cmu.edu/cmmi/background/conops.html. One area that the CMMI Steering Group has approved for expansion of model coverage is actually a clarification—coverage of the hardware discipline. In some organizations, hardware development centers are not being included in CMMI-based process improvement programs because the organizations that manage these centers perceive CMMI models as being “just for systems engineering and software engineering.” We anticipate that by adding hardware amplifications, typical work products, and other informative hardware examples, we can illustrate? demonstrate? assert? that the existing CMMI models are applicable not only to product and service development, but also to hardware development—and therefore all of the engineering disciplines involved in the development effort.

An expansion of model coverage that we will investigate is a broadening of the coverage of the development environment. Currently, a single process area, Organizational Environment for Integration (OEI), provides some coverage of the processes associated with the development environment. One of the source models, EIA 731, had many practices covering the engineering environment that are not included in CMMI best practices.

Change requests have included comments that “coverage of the work environment is necessary to address safety- and security-related issues.” A few provocative change requests have suggested that we need to include some of the best practices associated with “business continuity,” particularly considering the experiences and organizational shocks experienced after September 11.

Most challenging about all of these ideas is that we must balance the desire for expanded coverage with the desire to make the CMMI models simpler, clearer, and smaller.

What if my organization doesn’t develop, but provides services or operates systems?

This question has been a most challenging one. From the earliest development of CMMI, the desire to cover process improvement more widely was acknowledged. Today, organizations that focus on service delivery or the operation of systems can use the continuous representation to focus attention on all areas of the model that are relevant to them and improve their capabilities in those areas. But many organizations that do little or no development tell us that their customers (usually government) want to know their suppliers’ maturity levels when selecting providers of services and support beyond the engineering development phases of the life cycle. These organizations would benefit if a way to achieve a maturity level existed for non-development, non-engineering organizations.

One of our ongoing objectives is to maintain confidence in the meaning of maturity level ratings. If one organization is a maturity level 3, it must be comparable to others who have also achieved maturity level 3. Simply declaring several of the engineering process areas “not applicable” while claiming maturity level 3 would erode the confidence in maturity levels. We are committed to maintaining the integrity of maturity levels while attempting to meet the needs of organizations that don’t quite fit the engineering paradigm.

Our architecture team has investigated this and other issues and has defined architectural requirements necessary to accommodate future possibilities. This team decided that, with fairly minor adjustments, future models can be created to meet the needs of different domains. These needs will be met by “CMMI constellations,” or groups of related CMMI models that serve domains such as “development” or “services.” Each constellation can be created to reuse much of existing model content, but allows areas that are “not applicable” for a given domain (i.e., constellation domain) to be excluded.

Each constellation would include process areas and practices that are relevant to that domain and not to other domains. For example, in a service domain, the management of service level agreements is vital, but the focus on design, architecture, and product integration would not add value to the workforce. A “service management” process area category might replace the current “engineering” process area category, with a mixture of level two and level three elements. The current collection of models would be called the “development constellation,” and the new constellation in this example would be the “service constellation.”

By constructing future constellations from a common architecture, the commonalities across the constellations can be maximized. These commonalities are particularly important where companies and government agencies span multiple parts of the system life cycle. As one team member noted, “Some organizations must focus on multiple areas—development, service delivery, and operations—to be successful, and thus there may need to be a single constellation covering these areas across the enterprise.”

We want to maximize commonality for these multi-faceted organizations. Such an approach can aid process improvement across other boundaries. For example, we may have developers who maintain close relationships with the operators of the systems they provide. Using common approaches to enhance process relationships may be mutually beneficial to both groups.

Is there any plan to improve the appraisal method (SCAMPI)?

When we looked at the change requests submitted for the CMMI appraisal method, SCAMPI, there seemed to be two classes: (1) thoughtful recommendations for improvements that require extended consultation within the appraisal community, and (2) a smaller set of clarification changes critical to the use of the SCAMPI method that can be implemented relatively quickly. As a result, we are exploring the possibility of a SCAMPI update early in 2005 in addition to the Version 1.2 update, coinciding with the model update in mid-2006.

Is there a plan for expansion after the V1.2 release?

Yes, the possibilities of CMMI coverage for future constellations are already considered in the architecture, and coverage expansion can be initiated whenever suitable sponsorship is shown by the community. The *CMMI Concept of Operations* guides such model expansions; it requires the provision of expertise and funding to craft new CMMI practices, goals, process areas, and other artifacts needed to ensure that these new best practices can successfully be integrated into the CMMI Framework.

We envision that some proposed expansions may fall within the development scope of the current CMMI models, and thus might be numbered as “Version 1.x.” If the model expansion requires the addition of a new constellation, we would seek to treat it as a more major change, as we must ensure that the distinctions from the original development “parent” are clear.

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.

Eye on Integration

Emergent Issues in Interoperability

DAVID A. FISHER, DENNIS SMITH

Standard software engineering practice assumes that the requirements for a system are knowable and that the art of software development attempts to develop a system with careful fidelity to these requirements. We refer to systems that are developed with full knowledge and that can be controlled centrally as “closed systems.”

With modern systems development and the need to develop complex systems of systems, most systems are no longer “closed”; rather they are “unbounded” because they involve an unknown number of participants or otherwise require individual participants to act and interact in the absence of needed information.

In this column we discuss the distinction between unbounded and closed systems. We then outline the impact of unbounded systems on interoperability, and suggest that unbounded systems exhibit emergent properties that cannot be fully known in advance. We conclude with an initial set of implications for the interoperability problem.

Unbounded Systems Versus Closed Systems

Unbounded systems of systems are fast becoming the norm in many of the most demanding military and commercial applications. These include command-and-control systems, air traffic control systems, the electric power grid, the Internet, individual aircraft, enterprise database systems, and modern PC operating systems. For example, in net-centric warfare as applied by U.S. troops at the beginning of the current war in Iraq, agility and rapid progress were achieved by direct interactions among ground troops, helicopters, artillery, and bombers using equipment whose designs did not anticipate such usage and the accompanying mission changes.

Most systems of systems use their component systems in ways that were neither intended nor anticipated. Assumptions that were reasonable and appropriate for individual component systems become sources of errors and malfunction within systems of systems. As a result, the individual systems – and the system of systems as a whole—acquire vulnerabilities that can be triggered accidentally by normal actions of users and automated components, or exploited consciously by intelligent adversaries. For the complex systems of systems being constructed today and defined for the future, it is no longer possible for any human or automated component to have full knowledge of the system. Each component must depend on information received from other systems whose capabilities, intentions, and trustworthiness are unknown.

Unlike closed automated systems and traditional mathematics where neither correct nor useful results are required in the absence of complete and correct data, unbounded systems must function effectively with incomplete data and with data that cannot be fully trusted. Unfortunately, the primary mechanisms for achieving data integrity and trust in closed, tightly coupled, and fully understood systems will not achieve the same results in unbounded systems.

On the other hand, closed systems typically rely on effective mechanisms involving centralized control, centralized data, or hierarchical structures both in the development and execution of the system in order to provide the required degree of trust. However, when users or automated components are incompletely known, are untrustworthy, can be compromised, or can make errors, these mechanisms can amplify problems and undermine success. In particular, centralized control cannot be employed effectively against participants and components that are unknown or against those for which there is no effective enforcement mechanism.

Simply put, centralized data and control create a single-point target for attacks, accidents, and other failures. They also create communications vulnerabilities by increasing communication delay, transaction time, and ultimately user response times. Any hierarchical structure in a complex system has the unfortunate property that every node and link of the hierarchy constitutes a single point of failure for the system as a whole. That is, if the success of a function or system depends on the success of each of its components and subsystems, then an error, compromise, or failure in any one component propagates to the system as a whole and undermines system-wide success.

Interoperability in Unbounded Systems

Problems that arise from the unbounded characteristics of systems of systems are normally manifest as interoperability problems. We have categorized interoperability problems as programmatic, constructive, or operational depending on whether they arise from diffusion of management responsibility, diverse technical approaches, or user interactions with the system, respectively.

Traditional approaches to interoperability have focused on strengthening coordination among the organizations involved by tightening centralized control, increasing visibility and transparency of components, imposing more and stronger standards, and imposing additional coordination mechanisms. Obvious as this approach may be to those familiar with closed fully-understood systems, in systems of systems they become less and less effective as their size, distributiveness, and unboundedness increase. The continuing advance of memory, processor, and communications technologies ensures ever-increasing demands for systems and systems of systems that are more complex and more geographically distributed with more poorly understood and unknown components. Even if complete and accurate information could be obtained, it would be outdated rapidly by continuously changing circumstances within a system of systems. Instead, effective solutions must be developed that recognize, act upon, and exploit the inherent characteristics of unbounded systems.

Often when problems of interoperability arise in complex systems, there is a tendency to try to gain greater visibility, to extend central control, and to impose stronger standards. Not only are these actions are ineffective in complex systems, they also increase the likelihood of certain kinds of accidents, user errors, and other failures. What are called normal accidents are inherent and occur naturally in complex systems [1]. The frequency of normal accidents increases with the degree of coupling in systems. Coupling is increased by central control, overly restrictive specifications, and broadly imposed interface standards. Developers of systems of systems should strive for loose coupling.

Interoperability and Emergent Algorithms

Emergent properties are those properties of a whole that are different from, and not predictable from, the cumulative properties of the entities that make up the whole. The concept of emergent properties becomes increasingly important as the number and type of “actors” in a system of systems increase. Thus, large-scale networks such as the Internet (and in the future, networks that support net-centric warfare) are likely to experience emergent properties. Such networks are composed of large numbers of widely varied components (hosts, routers, links, users, etc.) that interact in complex ways.

Of necessity, each participant in such real-world systems (both the actor in the network and the engineer who constructed it) acts primarily in his or her own best interest. As a result, perceptions of system-wide requirements are interpreted and implemented differently by various participants, and local needs often conflict with overall system goals. Although collective behavior is governed by control structures (e.g., in the case of the networks, network protocols), central control can never be fully effective in managing complex, large-scale, distributed, or networked systems.

The net effect is that the global properties, capabilities, and services of the system as a whole emerge from the cumulative effects of the actions and interactions of the individual participants propagated throughout the system. The resulting collective behavior of the complex network shows emergent properties that arise out of the interactions among the participants.

The effect of emergent properties can be profound. In the best cases, the properties can provide unanticipated benefits to users. In the worst cases, emergent properties can detract from overall capability. In all cases, emergent properties make predictions about behavior such as reliability, performance, and security suspect. This is potentially the greatest risk to wide-scale networked systems-of-systems. The SEI recognizes that any long-term solution must involve better understanding and managing of emergent properties.

Recent research in the area of emergent algorithms [2, 3] has begun to identify, develop, and refine the methods first developed for other sorts of systems to solve problems of constructive interoperability. These methods and techniques are derived by analogy from approaches that have been effective in social, biological, and economic systems, but are applicable to the design, implementation, and evolution of software in a systems-of-systems context.

The methods of emergent algorithms as they apply to interoperability include cooperation without coordination, dynamic adaptation, continuous trust validation, dynamic capability assessment, opportunistic actions, anticipatory neighbor assistance, encouragement and influence, perturbation, and survivable architectures. At the same time, emergent approaches demonstrate an aversion to the risks imposed by tight coupling of systems, dependency on centralized control and data, and the vulnerabilities of hierarchical structures.

At their most fundamental level, emergent algorithms exploit cascading effects of loosely coupled, dynamically changing, and partially trusted neighbors to achieve a common purpose shared by a subset of the participants. Only a limited repertoire of emergent methods has been identified, and they are only partially understood. The complete range of effects, whether positive or ill, resulting from cascading interactions with dynamically changing neighbors, is unknown. Phase shifts are a particularly difficult class of emergent effects that can occur in any physical system. They are poorly understood in most physical domains, but offer the potential for both dramatic benefits and catastrophic failures. Well-known examples of phase shifts include the transition of an airplane wing angle from one that provides lift to one initiating a stall, an overload in a power system that initiates a blackout, or the action of a fuse in breaking a circuit.

Conclusions

This article offers some starting points and future directions for interoperability. Systems of systems now being constructed are characterized by vast complexity, many unknown aspects, and limited control with multiple organizations involved in their management, implementation, and use. The environment is increasingly unbounded. In unbounded systems, traditional software engineering methods become more and more constrained in their ability to effectively achieve interoperability. Emergent algorithms, applying methods analogous to those used in natural systems, may offer viable alternatives to traditional software engineering approaches.

References

[1] Perrow, Charles. Normal Accidents – Living with High-Risk Technologies. New York: Basic Books, 1984.

[2] Fisher, D. A. and Lipson, H. F. “Emergent Algorithms—A New Method for Enhancing Survivability in Unbounded Systems,” Proceedings of 32nd Annual Hawaii International Conference on System Sciences (HICSS-32), Maui, HI, Jan. 5-8, 1999. Los Alamitos, CA: IEEE CS Press, 1999.

[3] Fisher, David A. “An Emergent Approach to Interoperability in COTS-Based Systems,” submitted to International Conference on COTS-Based Software Systems 2005 (ICCBSS-2005), Bilbao, Spain, Feb. 7-11, 2005.

About the Authors

David A. Fisher is a Senior Member of the Technical Staff at the Software Engineering Institute (SEI) at Carnegie Mellon University, Pittsburgh PA. He also holds faculty appointments in the H. John Heinz III School of Public Policy and Management and in the Department of Engineering and Public Policy (EPP) and supervises theses in the Information Network Institute (INI) program of the Electrical and Computer Engineering (ECE) Department, all at Carnegie Mellon University.

He currently conducts research in interoperability in systems-of-systems with emphasis on emergent approaches and survivable architectures. Earlier he led the design and implementation of the Easel modeling and simulation system, developed algorithms in the areas infrastructure assurance, cooperative unmanned autonomous vehicles, mobile ad hoc networks, propagation of epidemics, and assessment of communications protocols. Before coming to the SEI, he managed program in component-based software and learning technologies at the Advanced Technology Program of the U.S. Department of Commerce, developed a theory of property based types, lead research projects in design of compilers, operating systems, and instruction set architectures, was Vice President for Advanced Development at Western Digital Corporation, proposed and orchestrated the development of the Ada language as a joint effort of the US DoD and the EU, and served as Staff Specialist for C4 in the Office of the Secretary of Defense.

Fisher holds a Ph.D. in Computer Science from Carnegie Mellon University, an M.S.E. from Moore School of Electrical Engineering, University of Pennsylvania, and a B.S. in Mathematics from Carnegie Institute of Technology.

Dennis Smith is the leader of the SEI initiative on the integration of software-intensive Systems. This initiative focuses on interoperability and integration in large-scale systems and systems of systems. Earlier, he was the technical lead in the effort for migrating legacy systems to product lines. In this role he developed the method "Options Analysis for Reengineering" (OAR) to support reuse decision making. He has also been the project leader for the computer-aided software engineering (CASE) environments project. Smith is a co-author of the book, Principles of CASE Tool Integration. He has an M.A. and PhD from Princeton University, and a B.A from Columbia University.

Security Matters

Install and Use Those Anti-Virus Programs

LAWRENCE R. ROGERS

If someone rang your doorbell and wanted to come into your living space to sell you something or to use your telephone, you'd need to make a decision whether or not to let them in. If they were a neighbor or someone you knew, you'd probably let them in. If you didn't know them but believed their story and found them to be otherwise acceptable—say they were neat and clean and not threatening—you'd probably also let them in, but you'd watch them closely while they were in your space.

What are you doing here? You are profiling this person and then deciding what to do based on that profile. It's your responsibility to be concerned about who enters your home. Further, if you have children, you've probably also taught them how to deal with strangers who come to your door.

Anti-virus programs work much the same way. These programs look at the contents of each file, searching for specific patterns that match a profile—called a virus signature[1]—of something known to be harmful. For each file that matches a signature, the anti-virus program typically provides several options on how to respond, such as removing the offending patterns or destroying the file.

To understand how anti-virus programs work, think about scam artists—people who visit your home to try to get you to buy a phony product or service, or to let them in. Once inside, they may try to steal your valuables or try to harm you in some way.

There are a variety of ways you might find out about a specific scam artist lurking in your neighborhood. Perhaps you see a television report or read a newspaper article about the person. They might include pictures and excerpts of the story the scam artist uses to persuade victims to lower their guard. The news report gives you a profile of someone you need to be on the lookout for. You watch for that person until either the story fades away or you hear that the person has been caught.

Anti-virus programs work much the same way. When the anti-virus program vendors learn about a new virus, they provide an updated set of virus signatures. Through features provided by the updated anti-virus program, your home computer also automatically learns of this new virus and begins checking each file for it, along with checking for all the older viruses. However, unlike scam artists, viruses never completely fade away. Their signatures remain part of the master version of all virus signatures.

1. http://www.cert.org/homeusers/HomeComputerSecurity/glossary.html#virussignature

Suppose a scam artist was at your front door. What would you do? Perhaps you might not allow him to enter or buy his product but, at the same time, you might try not to upset him. You'd politely listen to his story and then send him on his way. After you closed the door, you might call the police or the telephone number given in the report that initially brought him to your attention.

With viruses, you often have the chance to react to them when they've been discovered on your home computer. Depending on the specific characteristics of the virus, you might be able to clean the infected file. Or you might be forced to destroy the file and load a new copy from your backups or original distribution media[1]. Your options depend on your choice of anti-virus program and the virus that's been detected.

Viruses can reach your computer in many ways—through floppy disks, CD-ROMs, email[2], web sites, and downloaded[3] files. All need to be checked for viruses each time you use them. In other words, when you insert a disk into the drive, check it for viruses. When you receive email, check it for viruses. When you download a file from the Internet, check it for viruses before using it. Your anti-virus program may let you specify all of these to be checked for viruses each time you operate on them. Your anti-virus program may also do this automatically. All you need to do is to open or run the file to cause it to be checked.

Just as you walk around your living space to see if everything is OK, you also need to "walk" around your home computer to see if there are any viruses lurking about. Most anti-virus programs let you schedule periodic exams of all files on your home computer on a regular basis, daily for example. If you leave your computer turned on overnight, think about scheduling a full-system review during that time.

Some anti-virus programs have more advanced features that extend their recognition capabilities beyond virus signatures. Sometimes a file won't match any of the known signatures, but it may have some of the characteristics of a virus. This is comparable to getting that "there's something not quite right here, so I'm not going to let them in" feeling as you greet someone at your door. These heuristic[4] tests, as they're called, help you to keep up with new viruses that aren't yet defined in your list of virus signatures.

An anti-virus program is frequently an add-on to your home computer, though your newly purchased computer might include a trial version. At some point, say after 60 days, you must purchase it to continue using it. To decide whether to make that purchase or to look elsewhere, use these steps for evaluating anti-virus programs:

1. http://www.cert.org/homeusers/HomeComputerSecurity/glossary.html#media
2. http://www.cert.org/homeusers/HomeComputerSecurity/glossary.html#CD-ROM
3. http://www.cert.org/homeusers/HomeComputerSecurity/glossary.html#download
4. http://www.cert.org/homeusers/HomeComputerSecurity/glossary.html#heuristics

1. The ***D*emand** test: Can you check a file on demand?

2. The ***U*pdate** test: Can you update the virus signatures automatically? Daily is best.

3. The ***R*espond** test: What are all the ways that you can respond to an infected file? Can the virus checker clean a file?

4. The ***C*heck** test: Can you check every file that gets to your home computer, no matter how it gets there, and can those checks be automated?

5. The ***H*euristics** test: Does the virus checker do heuristics tests? How are these defined?

These tests—the **DURCH** tests—help you compare anti-virus programs. Once you've made your selection, install it and use all of its capabilities all of the time.

Intruders are the most successful in attacking all computers—not just home computers—when they use viruses and worms[1]. Installing an anti-virus program and keeping it up to date is among the best defenses for your home computer. If your financial resources are limited, they are better spent purchasing a commercial anti-virus program than anything else.

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.

1. http://www.cert.org/homeusers/HomeComputerSecurity/glossary.html#worm

Software Product Lines: Marathon Man

PAUL CLEMENTS

Product line champions emerge where you find them, but one person who absolutely has to have a clear, coherent vision for the product line is the product line architect. And beyond having the vision, he or she must be unwavering in communicating the vision in the face of reactions that may well range from apathy to hostility. Why hostility? Because for pockets in the organization that are resistant to change, the architect will be seen as the instigator of that change. Management may issue dictums and platitudes about how the product line approach will be good for the enterprise – and who could oppose shorter time to market and a fatter bottom line? – but the product line architecture is often the first sign to the product developers that things have changed.

A sure sign of trouble in an organization striving to produce a product line is that the role of product line architect is unfilled. A second sign of trouble is when that role is filled but its occupant has no nominal authority over the architecture of the product line. And a third sign of trouble is when the role is filled, the occupant has titular authority, but no one listens to him or her.

This is about one organization that went through all three of these phases. The first phase was preceded by a period in which a loose committee was formed to create a product line architecture. The committee had a chairperson who was a gifted designer, but due to circumstances beyond his control lacked the organizational presence to make his vision stick outside the committee (where it was needed to produce products). He, and management, needed to understand that his job didn't end when the architecture was drawn up. What he produced was essentially a re-structuring of the organization's application libraries that product-builders largely weren't using anyway. Management was unenthusiastic about the efforts and disbanded the committee, which it had failed to sufficiently empower in the first place.

This organization still believed in the concept of software product lines and tried again. Its second attempt fared better. Another individual was appointed to the job of designing a minimal product line architecture, which amounted to a set of common platform extensions that all products would be required to use. The goal was simple hardware portability. But unlike his predecessor, this architect grasped the big picture by understanding immediately that the platform extensions alone, while better than nothing, would not achieve the product line vision. He lobbied hard for a single architecture for every product in the product line, whereas at the time of his appointment a product team's only obligation was to use the common extensions in whatever manner it wished. This architect grasped the essential difference between a true product line and products built with reuse. With a common architecture, products could become interoperable (a rich new capability they lacked before) and exploit application-level commonality among themselves.

But first, the extended platform had to be designed. He formed a design committee, consisting of a few of the best designers in the domain plus representatives from several of the key product teams.

Buy in was always in short supply and foremost on the architect's mind. And over the course of several months, they produced a detailed design of that platform *that also included a picture of how a standard product that used the platform would be structured.* The architecture was, in it simplest form, layered; the topmost layer (L1) was product specific; the second layer (L2) was composed of software generic to each of the half-dozen or so classes of products that would populate the product line. That is, a product would incorporate one of several available L2 layers, depending on what kind of product it was. Lower layers formed the common platform. Well-defined interfaces and interaction mechanisms connected all the layers. (This is a standard scheme for a product line's software architecture.) The architect began to describe the architecture to management and to the product teams, always explaining the benefits of the vision. But he was also careful never to let the vision interfere with current work – after all, he would not be able to achieve that vision if he were removed from his position because short-term deadlines were not met.

Buy in came gradually and sometimes not at all. A gain in authority here means a loss of authority over there, and those giving it up did not do so easily. Some members of the design committee, instead of assuming ownership of the architecture and becoming its proponents in their home groups, seemed to view their role as reporting to their home groups "what that wacky group was up to now." Our architect realized that his committee was not the group of proponents he needed them to be. He knew that some members were even telling their home groups that they didn't understand the new product line architecture – even though they had helped to craft it. He was dealing with a severe case of organizational inertia and tried to remedy it. In a series of meetings and e-mail messages, he tried to impress on the committee and management the importance of the unified architectural vision. "If you feel we should not have one architectural model for the overall product line," he wrote, "please contact me." In other words, either get on board or step off the train now. "As a group," he wrote to the committee, "we need to be comfortable and confident with the architecture. I am leading this effort but not dictating it. However I am adamant about having a single architecture (for the whole product line) that we all support and understand. We need to know the architecture like the back of our hands if it is to work. We created it and we will be the ones who understand it, teach it to others, and grow it."

Eventually, the overall architecture was embraced by management in the form of a reorganization whose work units reflected the structure of the architecture: the platform group and the project groups were now joined by a group responsible for turning out the L2 layer. But management omitted an important component when it failed to appoint a single individual to have overall architectural authority over products. Conversely, the L2 group was not clearly chartered, and confusion was the result. The new head of the L2 group wrote our architect saying "Our group is starting to think about how to design this thing called L2. Our group seems to be the best place to start to think about such things. I'd like to make a presentation that shows how this work may fit into the overall architecture work being done by the design committee. I'd like to include a slide of the overall architecture in a presentation. Do you have one I could use?"

Thing? Your group seems to be the best place? *Start* to think? This work *may* fit? Our architect nearly *threw* a fit, and although his reply was composed, the message was clear: “The work you are describing is overall architecture work that is in the purview of the design committee.” And he appended a copy of the design committee’s scope to gently refresh the L2 leader’s memory, and pointedly invited him to the next design committee meeting.

The reorganization should have been the final action in the adoption of the overall product line architecture. In fact, the entire effort nearly failed at about the same time. Delivery pressure from some of the product groups caused management to temporarily embrace an expeditious but purely short-term approach: use as much code as possible from the current product set and get the products out the door. Later products could follow the product line approach. And who better to lead this effort than a man of proven accomplishments? Our architect was told to delay (or wrap up) his work on the architecture and begin porting code.

This was not the first time this organization had faced a crucible like this; schedule pressures had previously been used as an excuse to delay implementation of product line practices. Our architect knew that there would always be schedule pressures, and he knew that a true product line capability could never emerge in fits and starts. Rather than openly rebel against his management, to no good end, he saluted smartly and prepared for his new assignment. In a message to the design committee, product groups, and management, he accepted his new assignment with good grace but made sure the architectural vision did not die.

“As some of you have gathered,” he wrote, “my project scope has narrowed considerably with respect to ownership of the overall product line software architecture definition. To this point in time I have been the lead architect. A meeting was held recently in which I was requested to transfer the responsibility of the overall product line architecture to my replacement. I wish him the best of luck and will implement his vision as it becomes defined. Along with leading the design committee, the following responsibilities accompany the lead architect role and will from this point in time be transferred to him: (a) lead design meetings; (b) develop and maintain the product line architecture and model; (c) communicate the architecture; (d) develop a training class for the architecture; (e) define the technology roadmap of the architecture; and (f) schedule and release dates of architecture. From this point on all questions pertaining to these issues should be addressed to my replacement. To the best of my ability I will help him transition into his new role.”

Later, he explained his gambit to me. “As you can probably tell by the email flying around,” he wrote, “I have been doing my best to keep my company on one architectural path. I have been in extensive meetings with [upper management] and the [middle-level] supervisors. I could write a book alone on what has transpired in the last couple of weeks here! It all comes down to how we are to migrate to a complete product line and the fact that it does take several years to reach Nirvana. But as we migrate we need to spit out applications at various stages along the way. The first of these junctures was just encountered in this new generation. Several non-product-line proposals were flying around such that the architecture would become a splintered, ugly monster that would remain forever. It got to the point where I was completely out of architecture management for about the next nine months. That is why I relinquished control of the design committee. Obviously the people who wanted to split off did not want to take the baggage of architecture management with them. I wanted to make a point that someone is always responsible for the overall architectural vision and the path to that vision. So I handed off the vision! It was at that point that I believe the lights in the managers' heads started turning on as to what they were about to do. I believe we are back on line and I will explain more in detail as we unveil the plan. *The plan gives me technical responsibility to get the architecture done through all the layers for the pilots."*

Our architect had finally brought his organization through the three signs of trouble I described back at the beginning. The victory was not achieved overnight. But through it all, he persevered in keeping his unerring vision for a single architecture – and a single source of architectural authority – for the entire software product line.

I have learned that this architect is a marathon runner in his non-professional life. This did not surprise me. Although personally I find the thought of running 26 miles horrifying, I understand marathon runners to be driven, disciplined, goal-oriented people who can see the future clearly, don't mind some temporary pain, and who actually do their best when they hit what they call "the wall," the point at which no sane person could possibly take another step. Clearly this architect had exactly the right qualities he needed to succeed in his task at this organization.

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.

What's New

The Quality Attitude

WATTS S. HUMPHREY

This is the third in a series of columns on software quality and security. The first column, "Defective Software Works," discussed the software quality problem, why customers don't care about quality, and how bad software quality really is. The conclusion was that even defective software is of higher quality than just about any other humanly produced product. The second column, entitled "Security Changes Everything," described why testing alone cannot produce quality software. It explained that defective software cannot be secure, and it described the need to improve quality by at least 100 times over where we are today.

This column discusses the quality attitude and how software professionals and their managers must change their view of quality if we are to make much headway with software quality and software security. Quality is key to software performance, whether the concern is with function, usability, reliability, security, or anything else. In the next column, I will outline a strategy for solving these problems and describe steps that software professionals and their management can take.

The Testing Attitude

For as long as I have been writing programs, programmers have believed that when they clean up their programs in test, the programs will work. While this is often true, it isn't always true. This difference is the source of many of our software quality and security problems.

Our programs are often used in unanticipated ways and it is impossible to test even fairly small programs in every way that they could possibly be used. To appreciate the testing problem, consider the simple maze in Figure 1. The highlighted corners are possible branches and an example path from corner A to corner B is indicated by the arrows. Considering only the forward-going paths and ignoring loops, there are 252 possible paths from corner A to corner B in this 5 by 5 matrix. This matrix has only 25 branches. For 100 branches, we would need a 10 by 10 matrix, which would have 184,756 possible forward-going paths.

Most useful programs have many more than 100 branch instructions. For example, one of my C++ programs has 996 lines of code (LOC) and 104 branches. This is about one branch for every ten instructions. Even if large programs only had ten branches per thousand lines of code (KLOC), a 1,000,000 LOC program would have 10,000 branches. For a 100 by 100 square matrix with 10,000 branches, there are an astronomical $2.28*10^{58}$ possible forward-going paths from corner A to corner B. Few developers are even willing to run 252 tests, let alone $2.28*10^{58}$ tests.

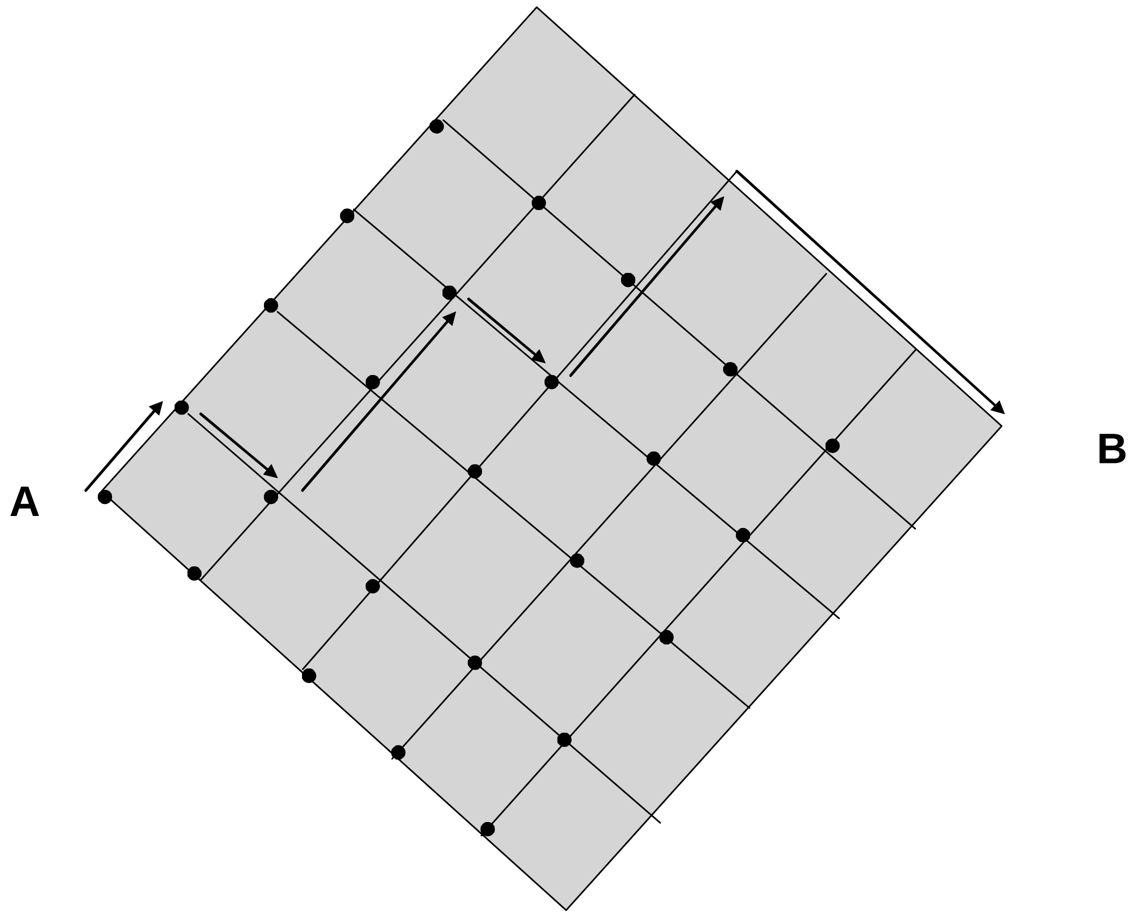


Figure 1: Possible Paths Through a Matrix

A brief analysis shows that testing this number of paths is impossible. For example, if you simultaneously ran a million tests a second for 24 hours a day on a million computers, it would take longer than the lifetime of the universe to run all of these tests. Even then, you would only have tested one set of data values.

Since it is impossible to exhaustively test large programs, we face the next question: “Is this much testing really necessary?” To answer this question, we must consider how many defects there are in typical large software systems. We now know how many defects experienced software developers inject. On average, they inject a defect about every ten lines of code. An analysis of data on more than 8,000 programs written by 810 industrial software developers is shown in Table 1.

The average injection rate for these developers is 120 defects per KLOC, or one defect in every eight lines of code. Even the top 10% of the developers injected 29 defects/KLOC and the top 1% injected 11 defects/KLOC. Even at the injection rate for the top 1% of software developers, a 1,000,000 LOC system would enter compiling and testing with 11,000 defects. More typical developers would have 120,000 defects in their products.

Table 1:Defect Injection Rages for 810 Experienced Software Developers

Group	**Average Defects per KLOC**
All	120.8
Upper Quartile	61.9
Upper 10%	28.9
Upper 1%	11.2

That is why most large programs, even after compiling and testing, have from 10 to 20 defects per KLOC when they enter system testing. For a 1,000,000 LOC system, that would be between 10,000 and 20,000 defects. With current practices, large software systems are riddled with defects, and many of these defects cannot be found even by the most extensive testing. Clearly, while testing is essential, it alone cannot provide the quality we need to have secure systems.

So, the first required attitude change for software professionals and their managers is to accept the fact that testing alone will not produce quality software systems. Also, since defective software cannot be secure, testing alone won't produce secure systems either.

The Defeatist Attitude

The next attitude that we must change concerns the feasibility of producing secure and high-quality software. Many software experts will tell you that there is no such thing as defect free software. Their obvious conclusion is that we must learn to live with poor-quality and insecure software products. While the previous discussion of testing may have convinced you that this attitude is correct, it is not. What these professionals are really saying is that there is no way to prove that a software system is defect free.

Unfortunately, it is true that there is no way to prove that a software system is defect free. But that is also true for every other kind of product. There is no way to prove that an automobile or a jet airplane or any other product is defect free, but we don't accept that as an excuse for poor quality automobiles or jet aircraft. The question is not to prove that a software product is defect free, but rather to prove that we have taken all practical steps and used all available methods to make the product as close to defect free as possible. There are many ways to do this, and they have been proven highly effective in many other technical fields. While the methods are simple, they are not easy. They require measurement, analysis, a lot of personal attention, and, above all, a conviction that quality is absolutely essential.

The current common view that it is impossible to produce defect free software is an excuse for not making the effort to do quality work. A more subtle form of this attitude is relying exclusively on tools to identify errors. How many times has some programmer told you that a new environment, language, debugger, or analyzer is the answer to the quality or security problem? But then, even after using all of these fancy new gadgets, the resulting products still have defects and are still insecure. The problem is attitude, and no one who counts on some tool or gadget to handle quality will ever produce large, secure software products.

The Quality Attitude

Having the right attitude can make an enormous difference. There is growing evidence that defect free software is possible. Some development groups are now producing reasonably large-scale software products that have had no defects found by their users. While these products actually may have latent defects, for all practical purposes, they are defect free. Today, a few development teams can consistently produce such software.

The Challenge Ahead

We need to learn from and extend these proven quality methods and practices to our highest priority software needs. Today's critical need is to apply these proven quality practices to the really complex and interconnected systems that support the nation's critical infrastructure.

What should be clear from this discussion is that we have not tried hard enough to produce defect free software, so we really cannot know if such high-quality products are possible or not. The next column will discuss the quality methods that people are using today to produce exceptionally high-quality software and the steps required to extend these practices to the truly massive systems that will be common in the future. These systems must be secure and, to be secure, they must be essentially defect free. Learning how to consistently produce such systems is our challenge for the future.

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 Tim Morrow, Julia Mullaney, Bill Peterson, Marsha Pomeroy-Huff, Alan Willett, and Carol Woody.

In closing, an invitation to readers

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

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

About the Author

Watts S. Humphrey founded the Software Process Program at the SEI. He is a fellow of the institute and is a research scientist on its staff. From 1959 to 1986, he was associated with IBM Corporation, where he was director of programming quality and process. His publications include many technical papers and several books. His most recent books are *Introduction to the Team Software Process* (2000) and *Winning With Software: An Executive Strategy* (2002). He holds five U.S. patents. He is a member of the Association for Computing Machinery, a fellow of the Institute for Electrical and Electronics Engineers, and a past member of the Malcolm Baldrige National Quality Award Board of Examiners. He holds a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, and an MBA from the University of Chicago.

CMMI for Small Businesses: Initial Results for the Pilot Study

LAUREN HEINZ

A frequent misconception about adopting Capability Maturity Model® Integration (CMMI®) is that it works only for large organizations—its cost and complexity appear to make it impractical for smaller organizations to implement.

Jim Hendrix, a business systems engineer at Analytical Services Inc. (ASI), doesn't believe that's the case. His organization recently implemented three process areas (PAs) of CMMI as part of a pilot study with the Software Engineering Institute (SEI). He said CMMI might even be more beneficial to smaller businesses because it allows them to grow more consistently and to make changes when they are less costly, that is, "before growth demands them."

ASI and Cirrus Technology Inc. are two Huntsville, AL, companies that participated in a recent study to develop a business case and technical guidance for small- to medium-sized enterprises—defined by the study as companies with 25 to 250 employees—that wish to adopt CMMI. Initial results from the pilot look promising: both organizations described significant benefits from using CMMI, especially in the areas of project management and change management, and both are in the process of documenting and disseminating their findings so that others can learn from their experiences.

The pilot, launched in July 2003, is part of a joint project between the SEI and the U.S. Army Aviation and Missile Research and Development Center (AMRDEC) Software Engineering Directorate in Huntsville. SuZ Garcia, a member of the piloting team from the SEI, said that the pilots help support a business case for deploying CMMI in smaller companies. "We now are much more confident that CMMI is technically feasible for these kinds of companies; the process areas helped them to solve current business problems, and the generic practices helped them to institutionalize the new practices," she said. "The challenges lie in making CMMI adoption for these smaller companies and projects affordable."

Pilot Process

Although the focus of the pilots was technical feasibility, the team chose an approach that, for the most part, simulated the level of resources that a small company might be able to expend for outside consulting for CMMI. Each organization received two days of training and business analysis, two days for an initial on-site appraisal, and one day each month for on-site consultation during implementation.

At ASI, a management and technical services provider with ISO 9001:2000 certification, the team voted on which areas needed the most work, settling on Requirements Management (REQM), Project Planning (PP), and Measurement and Analysis (MA). The next step was a gap analysis to determine where ASI's current practices mapped to these areas of CMMI and to develop an action plan for bridging the gaps.

The team then developed project-specific process descriptions based on this analysis. One project the team worked on involved a small project to upgrade a software system. Once the gap analysis was complete, the ASI team formulated an action plan, updated its existing process descriptions to conform to CMMI expectations, executed the pilot projects, collected metrics, gathered lessons learned (including benefits) and process improvements, and updated its processes to reflect process improvements. In the final phase of the pilot, the team set up a process for monthly status checks and coaching sessions with the SEI and AMRDEC consultants.

Pilot Results: ASI

Hendrix said that even though the team members worked hard to learn and implement the PAs, they were astounded by their appraisal results: capability level 2 in REQM, PP, and MA and capability level 3 in Organizational Training and Organizational Process Focus—two additional areas ASI decided to appraise to understand the capability of these existing processes, which were built on ASI's ISO 9001 implementation.

"We now have confidence that CMMI will help our services-based company, and we have renewed initiative for improvements," Hendrix said. "The pilot members and others at ASI have seen the value of CMMI and the potential it offers."

Pilot Results: Cirrus

Cirrus is a technical services and manufacturing enterprise based in Huntsville with several satellite offices around the country. The Cirrus piloting team followed the same general process as ASI, selecting PP, REQM, and Project Monitoring and Control (PMC) as its three focus areas for improvement. The team applied CMMI guidance to a two-person project to research and catalog information from the Web as well as to one of its manufacturing projects. At the end of the pilot, Cirrus was appraised at capability level 1 in each PA, which was its initial target, since it had only recently purchased the manufacturing sector of the company and knew that the organization wasn't ready to attack the institutionalization aspects of the generic practices until it had settled some of the basic operations of the new business unit.

"As the pilot proceeded, our emphasis changed from an original desire to 'get certified' to a focus of improving in smaller 'chunks' in areas identified by business analysis," said Bill Clemons, Cirrus project lead. "We realize now that we can use CMMI in the areas that naturally add value to our organization and quality to our end products by improving activities where we need them the most."

Next Steps

Garcia and her team are currently compiling guidance—in the form of documents, methods, and tools—that the pilot organizations used to make CMMI adoption easier. This guidance will be published in a series of reports throughout the next year and shared at several software conferences, including Software Engineering Process Group (SEPG) Australia Conference 2004 in September and the CMMI Technology Conference and User Group in November. The SEI and AMRDEC are also discussing potential follow–up work in the Huntsville region.

For more information, contact—
Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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

An Interview with Paul Nielsen, New Director of the SEI

JANET REX

U.S. Air Force Major General Paul D. Nielsen became the Software Engineering Institute's new director in August. He retired from the Air Force after 32 years of distinguished service. Most recently, he managed the Air Force's science and technology budget of more than $3 billion annually and a staff of approximately 8,700 people in the laboratory's component technology directorates and the Air Force Office of Scientific Research. He also was the Air Force's technology executive officer and determined the investment strategy for the full spectrum of Air Force science and technology activities. *news@sei* caught up with him soon after his arrival at the SEI.

What trends do you see in federal funding for science and technology?

In the 1990s, the biggest increases were in health, medical research, genomics; the budget for the National Institutes of Health quadrupled in ten years. Research in medical fields is important to us all, but as a country, we have to invest in the physical sciences, too. Other federal funding for research and technology decreased across the board in the 1990s but has been increasing since fiscal year [FY] 2000 and is likely to increase again in the FY 2006 budget. The Department of Defense, the Air Force, the Army and Navy all have seen sizable increases in science and technology funding since 2000.

Is this increased spending a characteristic of the post 9/11 world?

No, this trend started earlier. Of course 9/11 has focused attention on certain issues that might not otherwise have been as important. But as a country, we were under-investing in the physical sciences. A lot of great industries, and associated job growth, happened as a result of investments in the physical sciences. Look at the development of computers and then the integrated circuit revolution afterward: there was no business of this kind 60 years ago, but now it's a huge part of the economy.

So do you see the SEI as an engine of growth?

I do. We're in an interesting position as a quasi-governmental organization, an FFRDC [federally funded research and development center]. We do have a DoD focus, because they are our sponsors, but the work that we do trickles to the whole country, really the whole world. Software drives economic growth and competitiveness. Today most technological advances have a great deal of software content. The SEI's Watts Humphrey has pointed out that as software grows in complexity, as some programs begin to approach a billion lines of code, you can't have the same percentage of defects that you had with smaller, simpler programs, or you'd be driven crazy trying to find all those defects in test. The military services used to have their own programs to improve software quality, but since the cutbacks of the 1990s, the DoD has relied on the SEI to do this work on software process improvement.

Talk about your experiences with techniques for transition, such as applied technology councils in the Air Force and other kinds of partnerships. How might the SEI strengthen and develop its transition activities?

For every research organization, an engineer’s work is not done until it transitions, until it makes people’s lives better. Sometimes this is forgotten, and sometimes it’s hard to pull off. A big portion of the job has to be developing new technologies, but transition, making sure it gets out there, is critical.

We found that it’s important to stay close to customers. In the Air Force, applied technology councils met with customers at the senior level, so that we understood the needs from our customers’ most senior people, and they knew what we were doing for them, and we could establish schedules and commitments. These same principles apply for the SEI.

One way the SEI transitions technology is through training and education. The SEI is offering a growing number of courses and has seen a healthy growth in attendance over the past few years. These courses represent a tremendous opportunity to get the word out. And while we can’t train everyone on our own, we can rely on our partners to help us supply this training.

In the Air Force, there was a lot of discussion about a crisis in systems engineering: where had all the system engineers gone? Perhaps they all retired, or maybe they left the aerospace industry, landing in telecoms and dotcoms. I think the work of the SEI goes beyond its original charter and is contributing to a revitalized systems engineering discipline in the country as well as software engineering.

The Air Force is actively involved in attracting and nurturing scientists and engineers. How can the SEI help with building the software engineering workforce?

At times, people will hit some of the wrong buttons when they try to recruit scientists and engineers. You have to have reasonable salaries, but more importantly, engineers want to have interesting work. You need to take away obstacles and provide them with resources to do their jobs well and give them educational opportunities, which may not always be traditional academic programs. It may just be helping employees think at the next strategic level, and the SEI can help with that through its courses. Engineers want to make a difference—and they want to expand their skills.

What drew you to the SEI?

I’m a technophile; I started my career as a physicist. As I completed my Air Force career, I wanted to stay in high tech, I wanted to do something important, to be at or near a major university, and I wanted a leadership role at a great organization. I’m not a turnaround artist; I want to take good organizations and make them better. I had been aware of the SEI since it started in 1984, first with its research on methodology and then with CERT work in network security.

The SEI is in an ideal position to collaborate with industry, to impact the corporate world along with the military services. We can’t give advice to everyone, but we can educate key men and women so they know the right questions to ask, the right sensitivities to have. We need to reach out to international partners as well; a lot of systems today have software and other components that were developed outside of the United States. We want this software to be solid and secure no matter where it comes from. The SEI is big enough to matter but small enough to be nimble, and I want to cultivate the right partnerships to extend our influence.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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

Benchmarking for Improvement in Army Acquisition

STEPHEN BLANCHETTE, JR., KRISTI L. KEELER

In August of 2002, the U.S. Army's acquisition executive, Claude Bolton, asked the SEI to help him with an ambitious, multi-year program to improve the way the Army acquires software-intensive systems. The resultant Army Strategic Software Improvement Program (ASSIP) put into place the necessary infrastructure and initiatives to meet the requirements of section 804 of the FY03 National Defense Authorization Act, which requires defense agencies to develop plans for improving their software acquisitions.

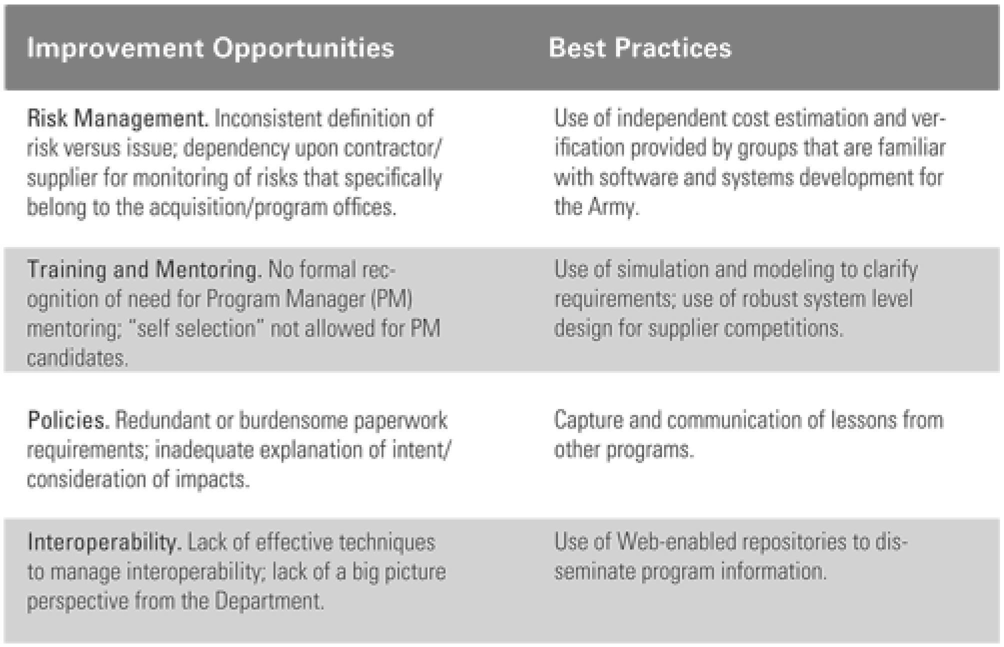
To determine how to improve acquisition processes, the SEI first set about determining the current state of the practice within Army acquisition. Developed specifically for ASSIP, Benchmarking for Improvement, or BFI, is the technique employed. The SEI uses BFI results, along with other data collected for ASSIP, to identify improvement needs across Army acquisition. This article discusses BFI and some of the emerging results from its application on Army programs to date.

Benchmarking for Improvement

The purpose of BFI is to understand the practices used in acquisition programs. Using a model-based question set, a team elicits practices that have been successful in an individual program office as candidate benchmarks for broader Army application. The team also identifies where a program needs practices to help overcome some difficulty (possibly leading to a benchmark for the program). BFI also helps determine where Army policies present barriers to program progress or where absent policies cause ambiguity and increased risk, so that the Army can set its own high-level benchmark targets.

Although BFI applies appraisal techniques, the focus is on discovering what a program office does rather than on measuring its maturity against a model. BFI is not an appraisal, and it does not result in a rating. It is a quick look to identify potential best practices or problems that may be relevant to the program or to the Army in general. However, BFI does retain some familiar appraisal elements: interviews are confidential, and findings require more than one source (human or document) for substantiation. Program managers own the results; to ensure program anonymity, the SEI never publishes attributable information.

To ensure broad system-level coverage, BFI has evolved to include topic areas from the new CMMI Acquisition Module (CMMI-AM). In fact, several BFIs are part of the pilots that will help the Office of the Secretary of Defense refine the CMMI-AM, which is becoming the preferred guide for defense acquisition office improvement.



Improvement Opportunities	Best Practices
Risk Management. Inconsistent definition of risk versus issue; dependency upon contractor/ supplier for monitoring of risks that specifically belong to the acquisition/program offices.	Use of independent cost estimation and verification provided by groups that are familiar with software and systems development for the Army.
Training and Mentoring. No formal recognition of need for Program Manager (PM) mentoring; "self selection" not allowed for PM candidates.	Use of simulation and modeling to clarify requirements; use of robust system level design for supplier competitions.
Policies. Redundant or burdensome paperwork requirements; inadequate explanation of intent/ consideration of impacts.	Capture and communication of lessons from other programs.
Interoperability. Lack of effective techniques to manage interoperability; lack of a big picture perspective from the Department.	Use of Web-enabled repositories to disseminate program information.

ties in
ıot
ıst 14
of the
) and the
3FI themes

guide
s that need
sponses by

Value to the Programs

For their investment, programs receive immediate and confidential feedback about their current practices, and can leverage that information to develop action plans based on their needs. For example, one benchmarked program used its results to validate its ongoing process improvement initiatives. Another program used its results as the basis for beginning improvement activities. Even some program executive offices (the organization level above the programs) have used BFI results to guide the start of their own improvement work.

Programs also have an opportunity to anonymously critique and influence higher-level policies, with the SEI acting as their advocate. Lastly, each program has a chance to maintain an ongoing relationship with the SEI to provide feedback and gain additional knowledge from other Army programs. Collaboration remains the best method for sharing the findings of the BFIs and other Army work facilitated by the SEI.

The Future of BFIs

As of this writing, the SEI plans to conduct six more BFIs through the end of 2004. Going forward, programs may request BFIs, or they may perform their own self-assessments against the CMMI-AM. As program offices become more experienced with process improvement, the SEI will continue to follow up to determine the successes and shortfalls of the programs' efforts in order to continually gauge the direction of Army-wide improvement.

For more information, contact—
Cecilia Albert

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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

Making the Best Use of the DoDAF Easier for DoD Organizations

PENNIE WALTERS

The Department of Defense Architecture Framework[1] (DoDAF), which is mandated by the DoD for large-scale systems, describes how the architecture for a system or system of systems (SoS) should be documented. The framework breaks that documentation into three major views: operational (OV), system (SV), and technical (TV), and one associated view, the all view (AV). Each view contains one or more graphic, tabular, and descriptive representations of the system. Because using any framework for the first time can be difficult, the SEI set out to discover how DoD organizations and their contractors can make the process easier.

Researchers at the SEI did this in two ways. First, they held an invitation-only workshop with researchers from government, industrial, and academic institutions. Second, they conducted eight interviews with architects, designers, architecture documentation reviewers, and program managers who are associated with acquisition category I (ACAT I) programs. Each program team was developing large-scale software-intensive SoSs using the DoDAF.[2] The architectures they were creating consisted of many OV, SV, and TV representations and served as the bases for the SoSs' acquisition planning, development, and deployment.

What the Workshop and Interviews Revealed

1. The DoDAF and current software architecture approaches were developed separately, by different organizations, for different purposes, and with little overlap. Hence, the views they use to document systems are significantly different, making compatibility and consistency among those views problematic.

   While the DoDAF is used to document system architectures, it does not address software architectures. For this reason, software views are often needed to supplement the DoDAF representations so that architects can better understand how well the systems will operate. Architects must determine which views are needed for a particular system. IEEE Std 1471-

1. DoD Architecture Framework Working Group. *DoD Architecture Framework Version 1.0*. Washington, DC: Department of Defense, 2003. http://www.teao.saic.com/jfcom/ier/documents/DOD_architecture_framework_volume1.doc and http://www.teao.saic.com/jfcom/ier/documents/DOD_architecture_framework_volume2.doc

2. The actual framework inquired about was an earlier version of the DoDAF—the Command, Control, Communications, Computer, Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework. At the time of the interviews, the DoDAF had not yet been officially released.

2000[1] and the SEI's Views and Beyond[2] approach (which leads to 1471-compliant documentation) both provide guidance on selecting views.

2. Program teams must tailor the DoDAF representations they use. Without this tailoring, reviewers will be inundated with information that they have neither the time nor the expertise to comment on. Tailoring may include annotating representations, providing more or less detail within specific representations, or adding guidelines for understanding them.

   Program teams may also consider adding other views such as those that provide additional information about software. Team members should ask only for the details that can be reviewed effectively by the available reviewers within the specified time frame.

3. None of the views conveniently represents multi-stage transitions from stovepiped legacy systems to interoperable SoSs, even though that is a common concern among those developing mission-critical systems. Some additional views, such as a master evolution plan, are needed to show when new systems and capabilities will be introduced and old systems retired.

4. Currently, each program office and its contractor must struggle with the differences between the DoDAF and other software architecture approaches when developing their own problem-solving approach. They must then train their staff members to follow that approach, using available tool sets as much as possible. Organizations also need to develop a process for using and building the DoDAF representations and any additional representations. That process will control the flow of building and reviewing the architecture, and relating the architecture to the system. Cross-service workshops, discussion forums, and documentation on lessons learned can promote a shared understanding of the architectural issues involved.

5. Tool usage requires specific rules for successful application within a program. Both commercial off-the-shelf (COTS) and custom tools are insufficient to meet the needs of architects or reviewers. Without training, architects will not use tools—even tailored ones—consistently across a program, and different contractors will apply the tools in different ways. Organizations may consider the use of different tool sets by contractors, provided the latter are given adequate guidance in defining the information that must be provided. Government organizations also need guidance on using the tools, so they can understand and review the representations.

   Crosscutting attributes such as performance, availability, modifiability, and security are at least equal to functionality in determining the architecture. Architecture evaluation in the form

1. Institute of Electrical and Electronics Engineers. IEEE Std 1471-2000. Piscataway, NJ: IEEE Computer Press, 2000.

2. Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison-Wesley, 2002.

of the SEI Architecture Tradeoff Analysis Method (ATAM) or architectural requirements elicitation in the form of an SEI quality attribute workshop (QAW) can help explore these attributes.

Learn More About the SEI's Research on Using the DoDAF

Because the DoDAF is now mandatory for DoD organizations, the SEI is striving to help them make the best use of it. To learn more about the SEI's findings on using the DoDAF, see the technical notes located at http://www.sei.cmu.edu/publications/documents/03.reports/03tn027.html?si
and http://www.sei.cmu.edu/publications/documents/03.reports/03tn006.html?si.