Carnegie Mellon
Software Engineering Institute

# news @ sei interactive

**Volume 5 | Number 4 | Fourth Quarter 2002**

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

news @ sei interactive

# From the Director

Since the late 1980s, the SEI℠ has been in the forefront of efforts to improve the quality of processes for organizations that develop and maintain software products and services. The SEI first met this objective through the Capability Maturity Model® (CMM®) for Software (SW-CMM). Because the SEI believes that the Capability Maturity Model Integration[1] (CMMI®) models represent the evolution of several great models that preceded them, including the SW-CMM, the SEI intends to remain steadfast in its commitment to CMMI while encouraging and supporting research in process improvements with the academic units at Carnegie Mellon.[2]

The CMMI models build on learning and experiences from the 1990s and are compatible with many other quality standards and techniques. CMMI best practices improve on SW-CMM and other best practices by enabling organizations to do the following:

- more explicitly link management and engineering activities to business objectives
- expand the scope of and visibility into the product life cycle and engineering activities to ensure that the product or service meets customer expectations
- incorporate lessons learned from additional areas of best practice (e.g., measurement, risk management, and supplier management)
- implement more robust high-maturity practices
- address additional organizational functions critical to their products and services
- more fully comply with relevant ISO standards

The SEI and its transition partners are currently working with numerous organizations, both nationally and internationally, in government, academia, and commercial industry, that are successfully adopting the CMMI Product Suite, which includes the CMMI models and the Standard CMMI Appraisal Method for Process Improvement (SCAMPI℠). The positive responses from these organizations have substantiated the SEI's commitment to the CMMI Product Suite. We are encouraged that more than half of the adopters to date are from small organizations (fewer than 100 employees), and many of them are software developers.

---

1 http://www.sei.cmu.edu/cmmi/cmmi.html?si

2 http://www.sei.cmu.edu/cmmi/adoption/kamlet-cmmi.html?si
http://www.sei.cmu.edu/cmmi/adoption/sunset-faq.html?si
http://www.sei.cmu.edu/cmmi/adoption/commitment-announce.html?si

In August, the SEI released the CMMI for Software (CMMI-SW) model and announced plans to develop CMMI interpretive guidance for software-only organizations. Many software-only and IT organizations, including many smaller commercial software companies, have been using CMMI to improve business processes for over a year. The SEI has begun to collect interpretation issues and lessons learned from these early adopters for the benefit of many other software-only and IT organizations that are only now beginning to adopt CMMI. SEI development of CMMI interpretive guidance for software-only and IT organizations, currently underway, will address the special needs of the software community when using CMMI products. The SEI will be holding sessions at various community events to collect information to aid development of this guidance. If you would like to help the SEI in this development effort, please contact <cmmi-software@sei.cmu.edu>.

To realize the benefits of software process improvement as quickly as possible, I also urge you to investigate the Team Software Process℠ (TSP℠) and Personal Software Process℠ (PSP℠). Experience has shown that TSP and PSP enable organizations to realize Maturity Level 4 and 5 capabilities within months as opposed to years (see http://www.sei.cmu.edu/tsp/results.html?si). TSP and PSP[1] can be used with any Capability Maturity Model, and are but one recent example of SEI and Carnegie Mellon intellectual leadership in helping organizations define, use, and improve their software-development processes.

**Stephen E. Cross**
SEI Director and CEO

[1] http://www.sei.cmu.edu/tsp?si

The Architect

# Enterprise Integration

Dennis Smith, Liam O'Brien, Kostas Kontogiannis, Mario Barbacci

For several years, the SEI has been conducting architecture evaluations with a variety of companies and DoD organizations,[1] helping them to identify system and software architecture concerns and risks. We have noticed particular problems from an activity referred to as "enterprise integration." The goal of enterprise integration (EI) is to provide timely and accurate exchange of consistent information between business functions to support strategic and tactical business goals in a manner that appears to be seamless. However, problems often emerge from overly ambitious or imprecise requirements resulting from inadequate plans for integrating different systems (legacy or otherwise). In this column, we describe the challenges associated with EI and provide a road map toward a solution.

## Reasons for Enterprise Integration

Depending on the organization, there are different legal and business reasons for integration. For example, U.S. federal law, under the Clinger-Cohen Act[2], mandates that each federal agency develop a plan for EI (for example, the Veterans Administration, the Internal Revenue Service, and the DoD Office of the Secretary of Defense have all instituted EI efforts). U.S. and international corporations regularly engage in major integration efforts (the telecommunications industry consortium, KLM Airlines, EURONEXT, and the Australian Stock Exchange, for example). Effective integration is often a prerequisite for e-business success (for example, successful enterprise-wide software integration was a critical factor for the rapid success of Dell online sales).

## Enterprise Integration Challenges

Integration of information systems is expensive and time consuming. Between 20% and 40% of labor costs can be traced to the storage and reconciliation of data. In addition, 70% of code in corporate software systems is dedicated to moving data from system to system.

Efforts at EI should transcend a single system or a single department, since optimization at a broader level helps to resolve the problems of redundancy and inconsistency. However, the term "enterprise" can be ambiguous; it has been referred to variously at the level of a division, a strategic business unit, a profit center, a public sector agency, and a corporation. While the scope of the term "enterprise" can legitimately vary depending on the needs of an organization, it is

---

1 See previous "The Architect" columns in the news@sei interactive archives.

2 http://www.dau.mil/jdam/contents/clinger_cohen.htm

important to define the scope for the purpose of an integration project and to stick with that definition.

Despite the large investments that are made, many integration efforts fail for a number of technical, organizational and management, and migration planning reasons. The technical issues include the following:

1. Legacy systems originated from unplanned, stovepipe development or were developed as batch, single tier.
2. Data is specific to the applications and was not designed for sharing.
3. The legacy systems were not initially designed for new quality-attribute requirements and are being affected by needs for interoperability, performance, security, and usability.
4. The scope for the integration effort is not adequately defined.
5. Decisions are made without performing adequate analyses (sometimes referred to as “management by magazine”)

The organizational and management issues include the need to obtain and maintain sponsorship and financial commitment at all levels of the organization for the duration of a potentially lengthy project; the need to break a project up into subprojects so that consistent, intermittent milestones can be measured; and the need to communicate effectively.

Finally, issues involved in migration planning are often poorly understood. These include a consideration of migrating databases and data, understanding user issues, providing training, and developing both temporary and long-term bridges between legacy systems and the newer applications.

## Critical Success Factors for Enterprise Integration

EI requires developing an overall plan to determine the scope of a set of projects and then implementing this plan with a sound technical approach. Both of these activities are crucial for the success of EI. We next briefly describe the overall planning activity and the activities associated with implementing the plan.

### Planning for Enterprise Integration

One way to plan for enterprise integration is to develop an enterprise architecture that represents how major information systems across the enterprise fit together. The architecture identifies the

scope of individual systems and the boundaries between them. Thus, an enterprise architecture is essentially a planning activity, rather than a development activity.

In practice, however, this distinction between planning and development is often ignored. Organizations focus on the wrong sets of issues in developing an enterprise architecture, and as a result get little value from it. Here are two basic problems that often occur in practice:

1. overscoping the enterprise architecture, resulting in an open-ended effort that is too ambitious to ever successfully implement
2. driving the enterprise architecture down to low levels of detail
    a. this results in losing the focus of the enterprise architecture as well as modeling low levels of detail that will only need to be repeated when actual systems are developed.
    b. it also results in focusing on the functionality of individual systems rather than the interconnections between them.

Examples of enterprise architectures are shown in sidebar 1 and sidebar 2.

An enterprise architecture has a very different scope from a software architecture. The software architecture is the structure or structures of a system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. Thus, a software architecture should begin where the enterprise architecture ends.

## Implementing the Plan

Implementing the plan for enterprise integration requires a set of projects that rely on a systematic approach for representing business, information, and technology perspectives; a systematic method to plan for specific tactical solutions, including budgets and project plans; and an understanding of infrastructure issues, such as processors and storage, networking, data management, and management and staffing. The management and organizational issues are critical for enterprise integration. In this column, we will focus primarily on technical issues.

To understand the relationship between the technical issues, we developed an integration model, which is shown in Figure 1.

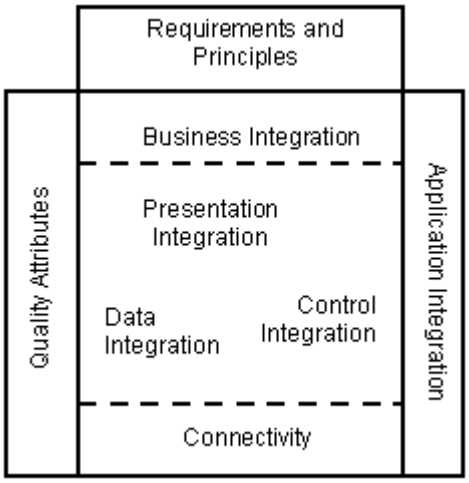



*Figure 1: Integration Model*

## The Technical Issues

*Requirements and principles* deal with determining the business drivers and guiding principles that help in the development of the enterprise architecture. Each functional and non-functional requirement should be traceable to one or more business drivers. Organizations are beginning to become more aware of the need for capturing and managing requirements. Use-case modeling is one of the techniques that is used for doing this.

*Business integration* concerns the design and modeling of business processes. Processes represent how the user sees the system. They are often highlighted when companies merge. Business process reengineering (BPR) decisions are often made in the context of enterprise integration.

*Presentation integration* involves with integration of corporate knowledge and business processes to enable an enterprise to share applications, services, and core competencies among the enterprise's main constituents: employees, customers, partners, and suppliers.

*Data integration* deals with aspects of integrating data within an enterprise. It deals with how data is modeled and the meaning of the data. It deals with normalization, validation, and integrity of data and what translations need to be applied to the data for exchange between applications within the enterprise or between the enterprise and outside systems.

*Control integration* deals with different types of messaging between applications and how these messages are handled based on different modes of communication and protocols.

*Connectivity* relates to workflow, data, and service connectivity and is handled by application bridges and gateways, message-handling services, and basic communication protocols.

*Quality attributes* involve architectural decisions that influence quality characteristics such as performance, dependability, and security. Quality attributes span several aspects of the integration model. When the software architecture is specified, designers need to determine the extent to which architecture decisions influence quality attributes, the extent to which techniques used for one quality attribute support or conflict with those of another attribute, and the extent to which multiple quality-attribute requirements can be satisfied simultaneously.

Quality-attribute issues do not get enough attention in enterprise integration. Quality-attribute requirements are usually defined too late in the development process. One problem is to determine how business requirements and drivers translate into quality-attribute requirements.

*Application integration* deals with getting different applications to work together using mechanisms such as automatic event notification, flow control, and content routing. Transactional integrity and application-context transformation across applications are important aspects of application integration. Application integration spans several layers of the integration model. Clear techniques are not available for making decisions about which applications to integrate and determining the nature of the problem. Applications on the same server may be easier to integrate, as cross-network integration can be complicated by different operating system versions, network delays, and different protocols.

## Example Requirements and Solutions

The following cases illustrate optimal examples of successful control, data, and presentation integration:

- Delivering information to the point of care: A physician accesses information about a patient, notifies appropriate specialists, and secures resources. (On average, hospital physicians need access to information they don't have three to five times a day.)
- Business-to-business transactions: Corporations implement their transactions and enact their business-process models using intelligent agents to access back-end corporate services.
- Corporations build customized services by fusing data content and services originally available as HTML web pages, programmatic scripts, and back-end applications so they can be accessed, fused, processed, and presented in various platforms.

Figure 2 shows a publish/subscribe configuration that can be used as a solution for integrating diverse applications. Figure 3 illustrates the component interactions. In the figures, *publisher* is the provider of the information, *subscriber* is the consumer of the information, and *broker* is the

service that controls and routes all the messages from the publishers to the subscribers. *Topic* is the subject of the information that is contained in the message. A publisher provides information on a specific topic, while a subscriber receives information for only the topics that it is interested in.

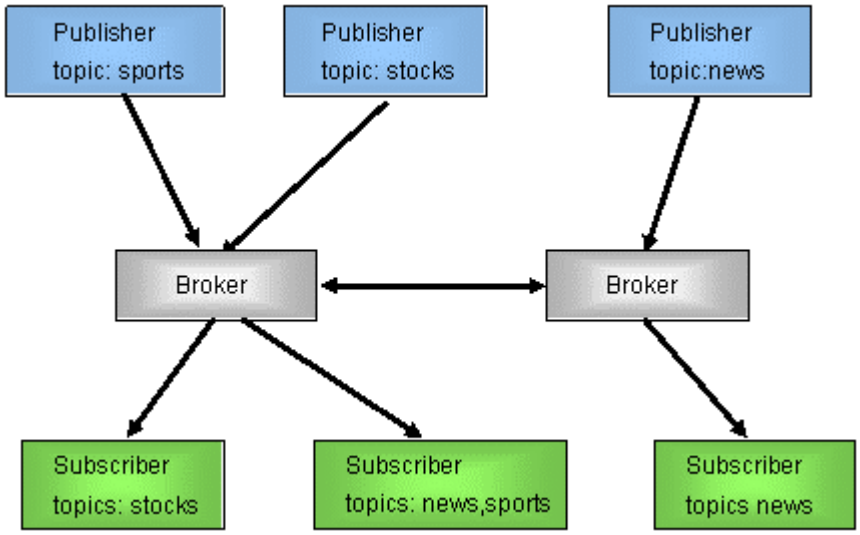



Figure 2: Publish/Subscribe Configuration

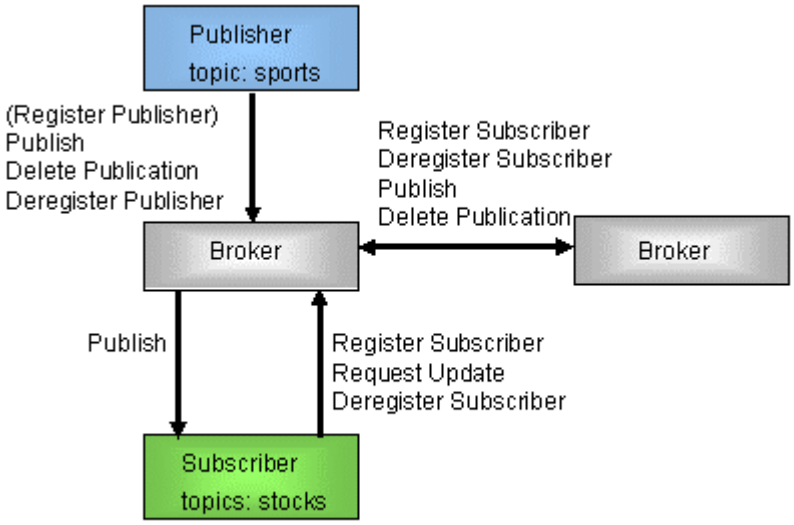



Figure 3: Publish/Subscribe Component Interactions

The solution depends on integration of heterogeneous services (operating systems, programming language, network protocol, data representation) and Internet accessibility of services (transactional services, security, messaging, naming services). Markup languages provide the ability to achieve these requirements more easily. One common approach to enterprise integration problems is the use of Enterprise Resource Planning (ERP). See sidebar 3 for details.

## A Roadmap to Future Progress

As we have seen, the demand for enterprise integration is broad. This affects all organizations, and addressing enterprise integration can often be a key to meeting the strategic goals of the organization.

A number of recent trends provide a foundation that can lead to future success. Middleware technologies have advanced rapidly over the past 10 years, and these enable more options for integration than had been previously available. The maturing of markup languages such as XML enable more effective integration, particularly between structured data, such as data from databases, and non-structured data, such as email. The Web can serve as a common front end for integrating a variety of applications, and it can enable effective presentation integration. The emergence of enterprise portals over the past several years demonstrates the strong interest and need for effective presentation integration. In addition, ERP applications, while still displaying significant problems, have become more mature, and their interfaces are better able to share data with other applications.

Despite this progress, the overall status of the field is immature. The issues of planning and scoping continue to hamper many efforts. There is significant confusion between the roles of different types of architectures. As a result, organizations sometimes go into too low a level of detail in developing an enterprise architecture, leading to a failure to adequately specify the scope, as well as a duplication of the effort that will later be applied to defining the software architecture. Although middleware technologies have matured significantly, clear guidelines are not yet available about when to use different types of technologies. As a result, some organizations have made decisions in a haphazard manner and have sometimes focused on vendor-driven approaches without adequate analysis. Some ERP products are heavily promoted as a total solution to enterprise integration. However, substantial modifications are often required to the ERP product for it to work in a specific organization. In addition, interfaces with legacy systems and other ERP products can be problematic.

The organizational issues that we mentioned earlier can be significant. An enterprise-integration effort affects an entire organization, and it is necessary to have long-term management support and financial commitment, realistic plans, and systematic migration planning.

To address these issues, a community-wide enterprise integration agenda needs to be developed. Such an agenda should focus, as a starting point, on the following issues:

- effective techniques for developing an appropriate scope for enterprise integration, including understanding business goals and how these goals are affected by current systems
- effective mapping between enterprise architectures and software architectures
- understanding the role and influence of organizational issues on enterprise integration
- understanding the role (and limitations) of middleware technologies and ERP solutions: when and how to select and use them
- understanding the role of migration planning

## For Further Information

A set of white papers about enterprise integration is available at: <http://www.sei.cmu.edu/plp/EI_IRAD/?si>.

A roadmap on enterprise integration is being written and will be available on the SEI Web site: <http://www.sei.cmu.edu/plp/EI_IRAD?si>.

## References


**Bass 2003** Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice* (Second Edition). Reading, MA: Addison-Wesley Longman, 2003 (to be published).

**C4ISR** <http://www.c3i.osd.mil/org/cio/i3/AWG_Digital_Library/index.htm>

**Clinger-Cohen Act** Information Technology Management Reform Act (ITMRA) <http://www.gsa.gov/attachments/GSA_PUBLICATIONS /extpub/Clinger_CohenAct1996_4.doc>.

**SEI Enterprise Integration White Papers** <http://www.sei.cmu.edu/plp/EI_IRAD?si>

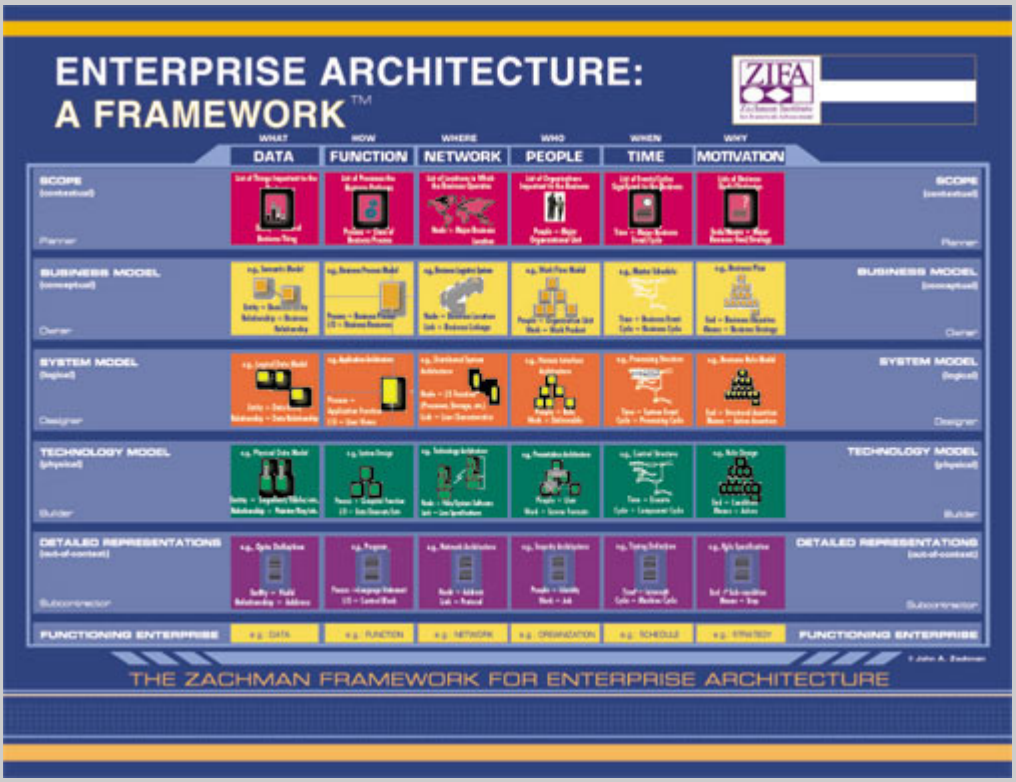
**SEI Interactive Archives** <http://interactive.sei.cmu.edu/news@sei/columns /columns-archives.htm>

**Zachman Framework** <http://www.zifa.com/>

## Examples of Enterprise Architectures and Frameworks

Zachman Framework

The Zachman framework is a classification system for design artifacts. It enables concentration on aspects of an object as well as the overall context of the object.






(Note: A PDF version of the Zachman framework can be downloaded from http://www.zifa.com/framework.html and enlarged for easier reading.)

Zachman distinguished between roles and product abstractions. Roles include planner, owner, designer, builder, and subcontractor. Product abstractions include data, function, network, people, time, and motivation.

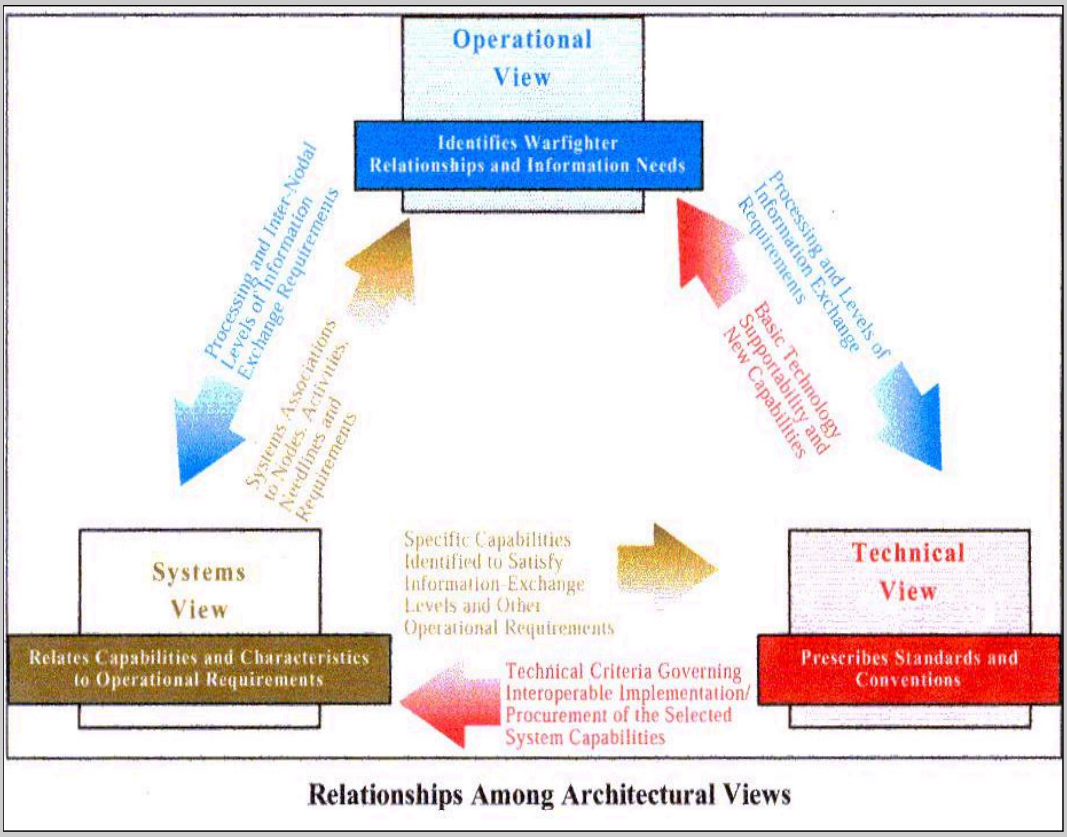
The Zachman framework has been used extensively as the starting point for enterprise architectures. The framework was originally proposed in 1978, using terms appropriate for structured analysis techniques. The conceptual approach needs to be updated to account for object-oriented language and client-server and Web-centric approaches.

While most analysts recognize the top two levels of the framework as being the appropriate role of an enterprise architecture, in reality they tend to delve into a much lower level of detail.

## C4ISR Architecture

The U.S. Department of Defense requires an architectural approach called C4ISR for systems within the domain of command, control, and intelligence. C4ISR has three components:

1. operational architecture, to describe tasks, operational elements, and information flows
2. technical architecture, to define a minimal set of rules (or set of standards) governing the arrangement, interaction, and interdependence of components
3. systems architecture, to describe systems and interconnections necessary to carry out tasks defined in the operational architecture




**Relationships Among Architectural Views**

In practice, the C4ISR goes into substantial levels of detail. There is not a clean separation between the planning and development stages and, as a result, there tends to be substantial duplication of effort between development and planning. Many practitioners don't have a clear understanding of the purpose and scope of the C4ISR.

### Enterprise Resource Planning Solutions

Enterprise resource planning (ERPs) solutions are integrated, technology-based solutions that provide support for standard enterprise needs in such areas as finance, human resources, and logistics. A number of vendors provide ERP solutions, including PeopleSoft, SAP, Baan, and Oracle. ERPs are popular because they offer the promise of enabling an organization to leverage the research and development efforts of the ERP vendors. Functional areas such as taxes, purchasing, and human resource management have a significant amount of commonality between organizations, and there are strong arguments for purchasing a ready solution rather than developing an application from scratch.

ERPs can make good sense for an organization. However, the benefits are far from automatic. In fact, a number of disasters in ERP implementations have occurred. A decision to implement an ERP requires careful analysis of the following factors:

- an understanding of the gap between the underlying object, data, or functional models of the ERP solution and those currently supported by an organization's legacy systems (often substantial effort is required to customize the ERP or change the organizational processes to match those of the ERP)
- an understanding of specific ways in which the ERP will interface with legacy systems, other ERPs, and future development efforts
- an understanding of migration issues, such as user training, data migration, phasing in of the ERPs, and phasing out of the legacy systems
- development of realistic cost and schedule estimates reflecting realistic expectations

More details on ERPs are available under Areas of Investigation in <http://www.sei.cmu.edu/plp/EI_IRAD/?si>

## About the Authors

Dennis B. Smith is a senior member of the technical staff in the Product Line Systems Program at the Software Engineering Institute. He is the technical lead in the effort for migrating legacy systems to product lines. In this role he has integrated a number of techniques for modernizing legacy systems from both a technical and business perspective.

Dr. Smith has been the lead in a variety of engagements with external clients. He led a widely publicized audit of the FAA's troubled ISSS system. This report produced a set of recommendations for change, resulting in major changes to the development process, and the development of an eventual successful follow-on system.

Earlier, Dr. Smith was project leader for the CASE environments project. This project examined the underlying issues of CASE integration, process support for environments, and the adoption of technology.

Dr. Smith is a co-author of the book, *Principles of CASE Tool Integration*, Oxford University Press, 1994. He has published a wide variety of articles and technical reports, and has given talks and keynotes at a number of conferences and workshops. He is also a co-editor of the IEEE and ISO recommended practice on CASE Adoption. He has been general chair of two international conferences, IWPC'99 and STEP'99. Dr. Smith has an M.A. and PhD from Princeton University, and a B.A from Columbia University.

Liam O'Brien is a Senior Member of the Technical Staff at the SEI. His primary research interests are architecture reconstruction, reengineering, migration planning and enterprise integration. He is the author of over 20 papers and book chapters.

Kostas Kontogiannis is an Associate Professor at the Department of Electrical and Computer Engineering, University of Waterloo. Kostas obtained a B.Sc in Applied Mathematics from the University of Patras, Greece, a M.Sc in Computer Science from Katholieke Universiteit Leuven, Belgium, and a Ph.D degree in Computer Science from McGill University, Canada. Kostas is leading the Software Reengineering group at the Department of Electrical & Computer Engineering. He is actively involved in various projects with IBM Canada, Center for Advanced Studies, the Network of Centers of Excellence and the Consortium for Software Engineering Research. He is the recipient of two IBM University Partnership Program Awards, and recipient of the Canada Foundation for Innovation Award. His research is focusing on the design and implementation of tools and techniques that allow for legacy system reengineering and for enterprise application integration.

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

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

Barbacci is a Fellow of the Institute of Electrical and Electronic Engineers (IEEE) and the IEEE Computer Society, a member of the Association for Computing Machinery (ACM), and a member of Sigma Xi. He was the founding chairman of the International Federation for Information Processing (IFIP) Working Group 10.2 (Computer Descriptions and Tools) and has served as chair of the Joint IEEE Computer Society/ACM Steering Committee for the Establishment of Software Engineering as a Profession (1993-1995), President of the IEEE Computer Society (1996), and IEEE Division V Director (1998-1999).

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

CMMI in Focus

# Messages from the Field

Mike Phillips

Because we get a lot of email, visit a lot of conferences, and teach CMMI®[1] courses, we get a lot of "messages from the field." In this quarter's column, we summarize and comment on some of the messages we have received.

## It Is "Process Improvement," Not CMM® vs. CMMI

Increasingly we are seeing both the legacy SW-CMM[2] and the CMMI upgrade being mentioned together in documents such as abstracts for SEPG.[3] This mention of both CMM-based products together makes sense. The motivation behind upgrading SW-CMM Version 2, draft C to CMMI was not that there were major flaws in the trusted SW-CMM, but that more had been learned about software engineering best practices during its use. Continuous improvement has been the goal for using the CMMs, and they in turn have needed updating as our knowledge grew. One might view this element of change to be "vertical," as it strengthens the understanding of key concepts such as measurement and analysis and risk management. So in this dimension,, CMMI offers modernization and greater clarity. But there was also the growing recognition that the power of process discipline was enhanced by involving more of the organization. This led to attention in the "horizontal" direction. Here, CMMI offers expanded breadth of coverage, since the rest of the organization can no longer say, "Oh, that's just for the software group." For those who need better coverage across the organization, the "I" in CMMI stands for "Integration," but for those who have brought the organization into healthy examination of its process capability and maturity with the SW-CMM, the "I" may also be defined as "Improved."

## New Uses

Our initial thought was that CMMI would be used in the same fashion as the SW-CMM. But some innovative uses of CMMI have come to our attention. For example, a large, global, multiproduct development organization has used CMMI to examine its approach to life-cycle management. This use of CMMI was not done to establish a benchmark, but as a checklist to ensure adequate coverage of life-cycle processes at the corporate level. Best practices critical to achieving corporate objectives for communication and product development could be identified and institutionalized across all corporate units using CMMI. Another example involved an

---

1 Capability Maturity Model Integration

2 Capability Maturity Model for Software

3 The Software Engineering Process Group Conference <http://www.sei.cmu.edu/products/events/sepg/>.

organization that chose to appraise its developmental risks. A risk taxonomy based on CMMI was used to probe the areas that posed particular difficulties for the project investigated.

## Government Acceptance

A joint government and industry team examined ways to reduce the impact of repeated government evaluations supporting source selections. A recent SCAMPI[SM1] appraisal provided an opportunity to test an idea that the team had considered: having government participation on an industry appraisal, rather than performing an additional independent evaluation. Two government evaluators, trained in both CMMI and SCAMPI, joined an industry appraisal team for the industry site SCAMPI. At the end of the SCAMPI, the government appraisers provided a letter to the SEI indicating that they had participated in the SCAMPI and agreed with the findings. Government participation in scheduled improvement appraisals offers a way to address the need for ensuring a level playing field of contractors for government procurement of software-intensive systems, while giving the confidence associated with government involvement. If this practice is encouraged within the DoD, a proposing contractor will be able to provide evidence that the government had participated in a recent appraisal. This could be considered sufficient to avoid the need for Software Capability Evaluations (SCEs) or Software Development Capability Evaluations (SDCEs) for those contractors.

## "Hardware Engineering"

We frequently hear requests for coverage of "hardware engineering." Most often that request refers to the hardware directly associated with software-intensive systems. At various presentations, attendees have noted the emphasis on systems engineering and software engineering. Some have asked about "the rest of the system." Many specify that they mean "hardware engineering." These questions usually come from computer-systems companies. The engineering discipline most often described is electrical engineering for the hardware associated most closely with the software. Pending a further update to CMMI models, likely a few years away, organizations are encouraged to use the flexibility of the Microsoft® Word versions[2] of CMMI models to insert typical work products or sub-practices that cover elements in the organization that have not seen their material included in the informative components of CMMI models. At the SEI, we also encourage the production of supplemental reports that can help organizations pursue process improvement in new areas without increasing the size of the model documents. If your organization has best practices that can aid us in expanding our coverage, we would welcome submissions to <cmmi-comments@sei.cmu.edu>.

---

1 Standard CMMI Appraisal Method for Process Improvement

2 http://www.sei.cmu.edu/cmmi/models/model-components-word.html

## "Errata"

Some readers have pointed out errors in the text of the CMMI documents. We regularly update an errata sheet and post it on the CMMI Web site. This practice allows us to avoid republication, while enabling us to catch errors that eluded us before publication of V1.1. In addition, readers have noted that some concepts may have been captured in one Process Area (PA), but are less clearly covered in another related PA. An example of this is coverage of technical performance measures. These measures are specifically mentioned in the Requirements Development PA, but not in the closely related Technical Solution PA. The CMMI model authors were often torn between the desire for absolute consistency across the PAs and the desire to avoid repetition. Care was taken to ensure that key concepts were addressed at each level of increased capability or maturity, but repetition across closely related PAs was not enforced. Another reader noted that test plans, explicitly discussed in the SW-CMM, were not emphasized across CMMI PAs. They are mentioned in the Requirements Management and Configuration Management PAs, but not, ironically, in the Verification PA. This was another example of an idea fully endorsed by CMMI authors, but not called out with the frequency desired by experts in the test discipline. Cases like this are sometimes better addressed with updates to the Frequently Asked Questions[1] (FAQs), but change requests will be held for the next CMMI model update, which will happen no earlier than 2005.

## "Technical Notes du Jour"

One of our commitments at the SEI has been to add informative elements around the basic building blocks of the product suite—the models, appraisal method, and training program. Three new technical notes are now available from the CMMI Web site[2]. A technical note by Brian Gallagher[3] describes how CMMI might be interpreted for use in operational organizations; one by Larry Jones[4] describes how CMMI supports organizations that have recognized the power of product line strategies for their development efforts; and a third by Paul Solomon[5] describes the synergy between CMMI and earned value management. Other technical notes currently being prepared cover how CMMI applies in service and sustainment organizations, how modern knowledge management practices can assist the employment of CMMI, and how areas such as security and safety engineering can be addressed using CMMI.

---

1 http://www.sei.cmu.edu/cmmi/adoption/cmmi-faq.html

2 http://www.sei.cmu.edu/cmmi/adoption/

3 http://www.sei.cmu.edu/publications/documents/02.reports/02tn006.html

4 http://www.sei.cmu.edu/publications/documents/02.reports/02tn012.html

5 http://www.sei.cmu.edu/publications/documents/02.reports/02tn016.html

## Ease of Upgrade

At the recent annual CMMI Technology Conference, several companies and lead appraisers described their move to CMMI. One described the long-term value of having gathered the Process Improvement Indicators, as those will help to guide future improvement. Another noted the relatively small additional effort (about 22 staff-days) to provide this enhanced picture of the process improvement effort. Another noted the enhanced business case analysis for CMMI over the SW-CMM, because applying process discipline across the rest of the development organization multiplied the favorable impacts on test time and defect-reduction activities.

As time goes on, the character of the CMMI Product Suite will build and develop based on what you, the community, find useful about it. The messages you send us will shape how CMMI can meet organizations' needs and influence the practice of software engineering throughout the world.

## About the Author

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

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

The COTS Spot

# Modernizing Legacy Systems

Robert C. Seacord

## Introduction

The knowledge embodied in legacy systems represents a significant corporate asset. Managing long-term software evolution is critical, because systems cannot be easily replaced.

The concept of software evolution has existed for some time, as the following transcript of a talk by J.I. Schwartz at the second NATO Software Engineering Conference held in 1969 indicates:

> *Managers must be willing to adapt as situations arise requiring changes in scope or technique. The basic system must, through such means as data description divorced from procedures, centralized data maintenance, and data-independent programming languages, be flexible enough to permit change and evolution without reprogramming. People must be flexible enough to tolerate rather drastic changes in direction [Buxton 70].*

Developing evolvable systems adds costs to the initial development cycle [Sommerville 00]. Managers, already constrained by unrealistic budgets and schedules, are forced to cut time and budget by de-emphasizing evolvability. Re-engineering existing systems to improve evolvability uses time and resources without repairing specific defects or deploying new functionality, making it a hard idea to sell to management. The cumulative consequences of near-term management decisions have had a debilitating effect on multinational companies, all levels of government organizations, and even a sizable part of smaller organizations [Brodie 95].

For large enterprise systems, a strategy of design for evolvability is a necessity. This approach does not distinguish between development and maintenance; maintenance is simply continued product development. A common practice has been to engage a large development team for several years, followed by years of maintenance by an inadequately small team, followed by a major effort to replace the system. An alternative and potentially more practical approach is to keep an adequately sized development team in place to constantly evolve and maintain the software in a sustainable manner.

Before systems can be evolved, they must be evolvable. Transforming legacy systems to the point where evolvable software development again makes sense is accomplished through legacy system modernization.

## COTS and Legacy Systems

Systems are made up of many different types of components:

1. commercial components: commercial off-the-shelf (COTS) products
2. open-source components: community-developed and maintained software

3. reuse libraries: corporate assets developed by other groups
4. legacy components: recovered, reusable assets
5. custom code: components developed for the current system

Similar to commercial components, legacy components may be treated as "black-box" components because the source code is not available or because of the inscrutability of the legacy source code. Black-box modernization involves examining the inputs and outputs of a legacy system, within an operating context, to understand the system interfaces.

Black-box modernization is often based on wrapping—surrounding the legacy system with a software layer that hides the unwanted complexity of the old system and exports a modern interface. Wrapping removes mismatches between the interface exported by a software artifact and the interfaces required by current integration practices [Wallnau 97, Shaw 95]. Ideally, wrapping is a black-box reengineering task, because only the legacy interface is analyzed and the legacy system internals are ignored. Unfortunately, this solution is not always practical, and often requires understanding the software module's internals using white-box techniques [Plakosh 99].

Software modernization efforts may also replace existing legacy code with COTS components. While replacing code that you own with code that you must license may seem counter-intuitive, there are numerous reasons to attempt it. Replacing legacy code with commercial components typically reduces the amount of source code that must be maintained. When building a business case for this type of modernization effort, it is important to consider not only the savings of not having to maintain code, but also the cost of extracting the legacy code, the amount of glue code that must be developed (and maintained), and any additional licensing and training costs.

The COTS-Based Systems initiative at the SEI has long followed a spiral model of customer engagements, where we work with customers to solve problems related to their COTS products. From these engagements, in which we apply known techniques, we discover new techniques and practices and expand and perfect existing ones. Recently, customer engagements have been formalized in the SEI's strategic approach to achieving its mission to "create, apply, and amplify" (described at <http://www.sei.cmu.edu/about/#strategy?si>).

Many, if not all, CBS initiative engagements over the past several years have been with customers that are modernizing or replacing legacy systems. Through these customer engagements we have developed and refined a risk-managed modernization (RMM) approach to modernizing legacy systems. This approach is documented in a new book in the SEI Series in Software Engineering, *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices* [Seacord 03].

## Risk-Managed Modernization

Risk management is a software engineering practice that continuously assesses what can go wrong (risks), determines what risks are important, and implements strategies to deal with those risks. Risk management is not a new idea; it is an integral element of the spiral development process developed by Barry Boehm [Boehm 88] and is documented in the SEI's Continuous Risk Management Guidebook[1] and a large body of related work [Boehm 91, Karolak 96, Higuera 96].

The objective of the RMM approach is effective risk management and mitigation leading to the development of a modernization plan that minimizes the risk of the modernization effort. The creation of this plan requires the application of a wide range of software engineering methods and techniques, as well as detailed understanding of legacy and modern technologies.

Figure 2 illustrates a UML™ activity diagram of the RMM approach. Ovals in the diagram represent activities. Arrows represent transition between activities. The horizontal *synchronization* bars require completion of the previous activities before starting new ones. The process starts with a project for modernization that has been selected using portfolio analysis. The end state for the process is an integrated modernization plan.

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

™ Unified Modeling Language is a trademark of Rational Software Corporation.

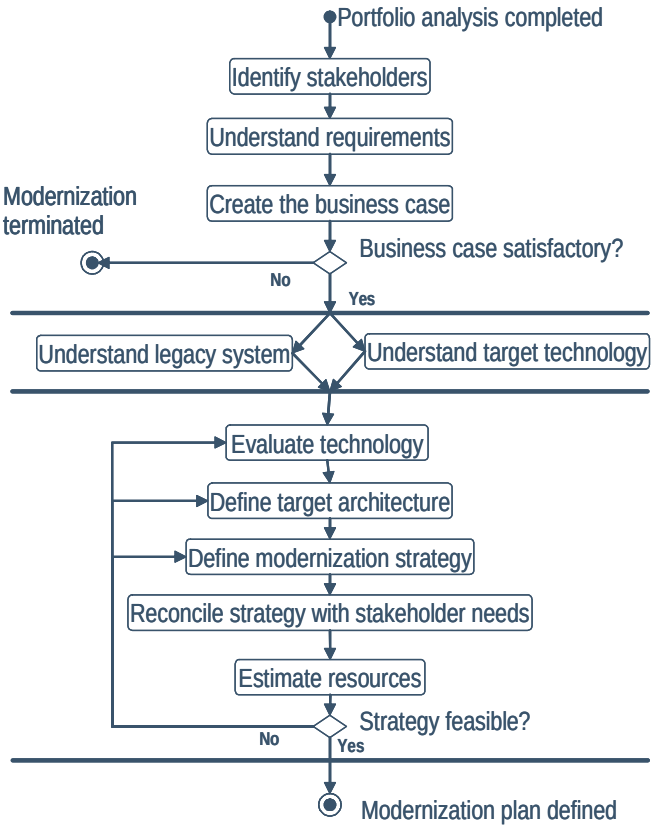



*Figure 2 Risk-managed modernization approach*

Each step in the RMM approach is described in more detail at <http://www.sei.cmu.edu/activities/cbs/mls/?si> and is fully described in *Modernizing Legacy Systems*.

## Summary

Software modernization requires carefully analyzing candidate modernization options and strategies that affect the interests of many stakeholders. As a result, software modernization requires making non-trivial and non-obvious tradeoffs. These tradeoffs are multi-faceted and include technical, programmatic, and organizational considerations that may strain an organization's decision-making abilities.

## References


**[Boehm 88]** Boehm, B. "A Spiral Model of Software Development and Enhancement." Computer (May 1988): 61-72.

**[Boehm 91]** Boehm, B. W. "Software Risk Management: Principles and Practices." IEEE Software 8, 1 (January 1991): 32-41.

**[Brodie 95]** Brodie, M. L. and Stonebraker, M. Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach (Morgan Kaufmann Series in Data Management Systems) (ISBN: 1558603301). Morgan Kaufmann Publishers, March 1995.

**[Higuera 96]** Higuera, Ronald P. and Haimes, Yacov Y. Software Risk Management (CMU/SEI-96-TR-012). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, June 1996.

**[Karolak 96]** Karolak, D. W. Software Engineering Risk Management. Los Alamitos, CA: IEEE Computer Society Press, 1996.

**[Plakosh 99]** Plakosh, Daniel; Hissam, Scott; and Wallnau, Kurt. Into the Black Box: A Case Study in Obtaining Visibility into Commercial Software (CMU/SEI-99-TN-010). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.

**[Seacord 03]** Seacord, Robert; Plakosh, Daniel; and Lewis, Grace. Modernizing Legacy Systems: Software Technologies, Engineering Processes and Business Practices.
New York, NY: Addison-Wesley, 2003.

**[Shaw 95]** Shaw, Mary. "Architecture Issues in Software Reuse: It's Not Just the Functionality, It's the Packaging." Proceedings of the IEEE Symposium on Software Reusability, April 1995.

**[Sommerville 00]** Sommerville, I. Software Engineering, 6th Edition. 2000.

**[Wallnau 97]** Wallnau, Kurt; Morris, Edwin; Feiler, Peter; Earl, Anthony; and Litvak, Emile. "Engineering Component-Based Systems with Distributed Object Technology" (Lecture Notes in Computer Science). Proceedings of the International Conference on Worldwide Computing and its Applications (WWCA'97). Tsukuba, Japan. March 1997.

## About the Author

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

Security Matters

# Installing and Using a Firewall Program

Larry Rogers

Your home computer is a popular target for intruders. They look for credit card numbers and bank account information so they can use your money to buy themselves goods and services. Intruders also want your computer's resources—your hard disk space, your fast processor, and your Internet connection. They use these resources to attack other computers on the Internet. In fact, the more computers an intruder uses, the harder it is for law enforcement to figure out where the attack is really coming from. If intruders can't be found, they can't be stopped, and they can't be prosecuted.

Why are intruders paying attention to home computers? Home computers are typically not very secure and are easy to break into. When combined with high-speed Internet connections that are always turned on, intruders can quickly find and then attack home computers. While intruders also attack home computers connected to the Internet through dial-in connections, high-speed connections (cable modems and DSL modems) are a favorite target.

One tool you can use to help defend your home computer against intruder attack is a firewall. This column describes a firewall, its importance to your home computer strategy, and some tips about creating firewall rules. You can read about other tools and guidelines in *Home Computer Security*[1].

## The Security Guard

Have you ever visited a business where you first stopped at the reception desk to interact with a security guard? That guard's job is to assess everybody who wishes to enter or leave the building to decide if they should continue on or be stopped. The guard keeps the unwanted out and permits only appropriate people and objects to enter and leave the business's premises.

Let's dig deeper into this analogy. When someone enters a building, the security guard usually greets them. If they have an appropriate identification badge, they show it to the guard or swipe it through a reader. If all is okay, they pass through the guard's checkpoint. However, if something's wrong or if they are a visitor, they must first stop at the guard desk.

[1] http://www.cert.org/homeusers/HomeComputerSecurity/

The guard asks whom they wish to see. The guard may also ask for identification such as a driver's license or their company ID. The guard reviews the list of expected guests to see if this person is approved to visit the party in question. If the guard decides everything is all right, the visitor may pass. The visitor usually signs a logbook with their name, the company they represent, whom they are seeing, and the time of day.

On a computer, the firewall acts much like a guard when it looks at network traffic destined for or received from another computer. The firewall determines if that traffic should continue on to its destination or be stopped. The firewall "guard" is important because it keeps the unwanted out and permits only appropriate traffic to enter and leave the computer.

To do this job, the firewall has to look at every piece of information—every packet—that tries to enter or leave a computer. Each packet is labeled with where it came from and where it wants to go. Some packets are allowed to go anywhere (the employee with the ID badge) while others can only go to specific places (visitors for a specific person). If the firewall allows the packet to proceed (being acceptable according to the rules), it moves the packet on its way to the destination. In most cases, the firewall records where the packet came from, where it's going, and when it was seen. For people entering a building, this is similar to the ID card system keeping track of who enters or the visitor signing the visitor's log.

The building's guard may do a few more tasks before deciding that the person can pass. If the person is a visitor and is not on the visitors list, the guard calls the employee being visited to announce the visitor's arrival and to ask if they may pass. If the employee accepts the visitor, they may proceed. The guard may also give the visitor a badge that identifies them as a visitor. That badge may limit where in the building they can go and indicate if they need to be escorted. Finally, no matter whether the person is a visitor or an employee, the guard may inspect their briefcase or computer case before they pass.

The firewall can also check whether a given packet should pass, allowing the computer's user to respond to unanticipated network traffic (just as the guard does with the unexpected visitor). Individual packets can be allowed to pass, or the firewall can be changed to allow all future packets of the same type to pass. Some firewalls have advanced capabilities that make it possible to direct packets to a different destination and perhaps even have their contents concealed inside other packets (similar to the visitor being escorted). Finally, firewalls can filter packets based not only on their point of origin or destination, but also on their content (inspecting the briefcase or computer case before being allowed to pass).

Back to the office building, when employees leave the building, they may also have to swipe their ID card to show that they've left. A visitor signs out and returns their temporary badge. Both may be subject to having their possessions inspected before being allowed to leave.

Firewalls can also recognize and record when a computer-to-computer connection ends. If the connection was temporary (like a visitor), the firewall rules can change to deny future similar connections until the system's user authorizes them (just as visitors must re-identify themselves and be re-approved by an employee). Finally, outgoing connections can also be filtered according to content (again, similar to inspecting possessions at the exit).

What does this all mean? It means that with a firewall, you can control which packets are allowed to enter your home computer and which are allowed to leave. That's the easy part.

The hard part is deciding the details about the packets that are allowed to enter and exit your home computer. If your firewall supports content filtering, you also need to learn which content to allow and which not to allow. To help you get a handle on this harder task, let's return to our security guard analogy.

Imagine that you are that security guard and it's your first day on the job. You have to decide who's allowed in, who's allowed out, and what people can bring into and take out of the building. How do you do this?

One strategy is to be very conservative: let no one in or out and let no possessions in or out. This is very simple, very easy to achieve, but not particularly helpful to the business if none of its employees or visitors can get in or out. Nor is it helpful if they can't bring anything with them. With this type of strategy, your tenure as a security guard may be short-lived.

If you try this, you quickly learn that you need to change your strategy to allow people in and out only if they have acceptable identification and possessions using some agreed-to criteria. Add the requirement that if you don't meet the precise criteria for admittance, you don't get in.

With most firewalls, you can do the same thing. You can program your firewall to let nothing in and nothing out. Period. This is a deny-all firewall strategy and it does work, though it effectively disconnects you from the Internet. It is impractical for most home computers.

You can do what the security guard did: review each packet (employee or visitor) to see where it's coming from and where it's going. Some firewall products let you easily review each packet so that you can decide what to do with it. When you are shopping for a firewall, look for this review feature because it can be quite helpful. Practically speaking, it isn't easy to decide which traffic is all right and which is not all right. Any feature that makes this job easier helps you achieve your goal of securing your home computer.

Just like the security guard who learns that anybody with a company photo ID is allowed to pass, you too can create firewall rules that allow traffic to pass without reviewing each packet each time. For example, you may choose to allow your Internet browsers to visit any web site. This rule would define the source of that traffic to be your browsers (Netscape Navigator and

Microsoft Internet Explorer, for example) and the destination location to be any web server. This means that anybody using your home computer could visit any Internet web site, as long as that web server used the well-known standard locations.

## Building Firewall Rules

Now that you have an idea of what your firewall security guard is trying to do, you need a method for gathering information and programming your firewall. Here is a set of steps to use to do just that:

1. The Program test: What's the program that wants to make a connection to the Internet? Although many programs may need to make the same type of connection to the same Internet destination, you need to know the name of each. Avoid general rules that allow all programs to make a connection. This often results in unwanted and unchecked behavior.

2. The Location test: What's the Internet location of the computer system to which your computer wants to connect? Locations consist of an address and a port number. Sometimes a program is allowed to connect to any Internet location, such as a web browser connecting to any web server. Again, you want to limit programs so that they only connect to specific locations where possible.

3. The Allowed test: Is this connection allowed or denied? Your firewall rules will contain some of each.

4. The Temporary test: Is this connection temporary or permanent? For example, if you're going to connect to this specific location more than five times each time you use the computer, you probably want to make the connection permanent. This means that you ought to add a rule to your firewall rules. If you aren't going to make this connection often, you should define it as temporary.

With each connection, apply the PLAT tests to get the information you need to build a firewall rule. (A worksheet[1] in PDF format is available to help you with this.) The answer to the PLAT tests tells you if you need to include a new firewall rule for this new connection. For most firewall programs, you can temporarily allow a connection but avoid making it permanent by not including it in your rules. Where possible, allow only temporary connections.

As you run each program on your home computer, you'll learn how it uses the Internet. Slowly you'll begin to build the set of rules that define what traffic is allowed into and out of your computer. By only letting in and out what you approve and denying all else, you will strike a practical balance between allowing everything and allowing nothing in or out.

---

[1] http://www.cert.org/homeusers/HomeComputerSecurity/checklists/checklist4.pdf

Along the way, you may come across exceptions to your rules. For example, you might decide that anybody who uses your home computer can visit any web site except a chosen few web sites. This is analogous to the security guard letting every employee pass except a few who need more attention first.

To do this with firewall rules, the exception rules must be listed before the general rules. For example, this means that the web sites whose connections are not allowed must be listed before the rules that allow all connections to any web site.

Why? Most firewall programs search their rules starting from the first through the last. When the firewall finds a rule that matches the packet being examined, the firewall honors it, does what the rule says, and looks no further. For example, if the firewall finds the general rule allowing any web site connections first, it honors this rule and doesn't look further for rules that might deny such a connection. So, the order of firewall rules is important.

Many firewalls can be programmed to require a password before changing the rules. This extra level of protection safeguards against unwanted changes no matter their source, that is, you, an intruder, or another user. Follow the guidance in "Use Strong Passwords[1]" in *Home Computer Security* when assigning a password to your firewall.

Finally, make a backup of your firewall rules. You've probably taken a lot of time to build and tune them to match how your home computer is used. These rules are important to your computer's security, so back them up using the guidance in "Make Backups of Important Files and Folders[2]" in *Home Computer Security*.

## Types of Firewalls

Firewalls come in two general types: hardware and software (programs). The software versions also come in two types: free versions and commercial versions (ones that you purchase). At a minimum, you should use one of the free versions on your home computer. This is especially important if you have a laptop that you connect to your home network as well as a network at a hotel, a conference, or your office.

If you can afford a hardware firewall, you should install one of these too. The same issues apply to the hardware versions that apply to the software versions. Many can also be password protected against unwanted changes. Search the Internet with your browser to see what's available and what they cost. The price of hardware firewalls is coming down as the demand grows.

---

1 http://www.cert.org/homeusers/HomeComputerSecurity/#6

2 http://www.cert.org/homeusers/HomeComputerSecurity/#5

## Summary

A firewall is your security guard that stands between your home computer and the Internet. It lets you control which traffic your computer accepts. It also controls which of your programs can connect to the Internet. With a firewall, you define which connections between your computer and other computers on the Internet are allowed and which are denied. There are free firewall products that provide the capabilities you need to secure your home computer. Commercial versions have even more features that can further protect your computer.

Firewalls are an important part of your home computer's security defenses.

Example: Operating a Firewall Program (contains several graphics), at <http://www.cert.org/homeusers/HomeComputerSecurity/examples.html#firewall>.

## 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.

Software Product Lines

# E Pluribus Unum

Paul C. Clements

The title of this column is Latin for "Out of many, one." It is the motto of the United States and expresses the vision of its founders that out of a group of loosely related states would emerge a unified nation. This motto is part of our culture. Culture is what binds a group of people together and gives them a common identity. Culture is the name we give to our habits, our values, our history, our likes and dislikes. Culture is a lens through which we see the world.

Product line organizations have a culture, also. After years of pulling back the covers and peeking inside companies who do this for a living, some aspects of that culture have become startlingly evident. Successful product line organizations employ people who are very comfortable working under the auspices of strong, documented processes. They have long and deep expertise in their application area. They have an identifiable champion who lobbied tirelessly to turn the organization to product lines.

And it turns out that the best ones have an unofficial motto, which might fairly be phrased as "*e pluribus unum.*" The best product line organizations view their business, their mission, as building a product line—singular. Other companies, including those still climbing the product line mountain, tend to view their business as turning out individual products—plural.

How does this manifest itself operationally? Quite often, a good barometer is the behavior of the developers. In an organization that has not yet reached the spiritual plateau of *e pluribus unum*, a developer will be likely to change a core asset unilaterally if it's not quite what he needs and a product deadline is looming. He may feel badly about it, and he may intend to send the change to the core asset team later, but he'll do it. After all, where would our company be if we didn't sell any products?

In a mature product line organization, covertly circumventing established processes would simply never occur to a developer. First of all, the processes in place would accommodate core assets that need tweaking under short deadlines; there would be no need to shortcut the process because the process would be robust enough to work in that case. But more to the point, the developer's first instinct would be to ask "What should I do that is best for the long-term health of the entire product line?"

Each of these hypothetical developers is doing what he believes is in the best interest of his organization. And each is probably operating under the reward and incentive structures that are in place. The first developer believes that the product line approach is a luxury that can only be afforded after the company's real business is taken care of. The second developer knows that the product line *is* the company's real business. How did they each come to these conclusions? What messages has management sent about what really counts when the pressure's on?

Out of many, one: the Zen of product lines if there ever was one. We first observed it when writing the CelsiusTech case study [1]:

> *Externally, CelsiusTech builds ship systems. Internally, it develops and grows a common asset base that provides the capability to turn out ship systems. This mentality--which is what it is--might sound subtle, but it manifests itself in the configuration-control policies, the organization of the enterprise, and the way that new products are marketed.*

It emerges over and over. From a 1999 case study of the software product line of an avionics manufacturer whom we visited (the case study remains unpublished to preserve confidentiality):

> *We shared with this group [of senior technical managers] that we hypothesized that one difference between a mature product line organization and an immature one is that the mature organization sees their business as maintaining and nurturing the product line, where as the immature one sees their business as turning out products via reuse. "I can tell you," said one of them immediately, "that is not just a hypothesis." (He meant it was true.) It is exactly how they view themselves -- their primary responsibility is to the product line.*

And from a report on an engine control software product line at Cummins Engine Company, the world's largest manufacturer of large diesel engines [2]:

> *The software product line program slashed development costs and cycle time across these highly varying products and launched many successful products over the past four years… Cummins now has a group dedicated to the study and improvement of software architecture. No longer is Cummins concerned with the separate delivery of software products, but has turned its focus to the delivery of integrated product lines and taking advantage of all the benefits that come with a product line engineering concept.*

The difference was put to us in stark relief recently when we urged a company struggling to adopt the product line approach to devote more resources to their core asset team. The senior manager, in a moment of keen frustration as a result of too much business and too few people, lost his composure. "You don't seem to understand," he erupted. "We have products to get out!"

We do understand. But we also understand the difference between a short-term and a long-term view. We suspect this company will continue to get products out, because that's what their management holds to be important. And so their developers will continue to take shortcuts (and be rewarded for it), and their product line efforts will continue to falter. Because by "getting products out" they are not making the investment to fully exploit what those products have in

common. As they contract to build more and more products, they are in more and more need of the product line paradigm, and are less and less likely to achieve it.

I spoke recently with someone who just finished a market study of twelve organizations who are producing e-commerce software. “Mass customization” is the order of the day; they produce many versions of their products, tailored to individual customers. Or at least, they would like to. The market study asked about their ability to manage the different versions effectively. Seven of the organizations responded that they had already “hit the wall”—that the complexity of keeping track of the deployed versions, and the assets that went into the building of each, was already consuming vast amounts of time and intellectual capital, and they were on the verge of losing control completely. Four of the organizations avowed that they weren’t yet to that point, but they could all see that wall fast approaching. And one organization was so intimidated by the problem of tracking multiple versions that they actually refused to deploy more than one version of their product. If a customer wanted something different, well, sorry, that’s not possible. My contact asked each of them if they thought they could use technology that would let them manage their software as a set of variations off of a single product, rather than so many unrelated products. “It was amazing,” he said. “You could see the light go on above their head. It was a way of thinking about their business that had never occurred to them before.” It was an *e pluribus unum* epiphany.

Out of many, one. For the mature product line organizations we have observed who have this motto, it also describes their position relative to their competition.

## References


[1] Brownsword, L. & Clements, P. *A Case Study in Successful Product Line Development* (CMU/SEI-96-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996 <http://www.sei.cmu.edu/publications /documents/96.reports/96.tr.016.html?si>.

[2] Dager, J. “Cummins’s Experience in Developing a Software Product Line Architecture for Real-time Embedded Diesel Engine Controls.” Donohoe, P., ed., Software Product Lines: Experience and Research Directions. Boston, MA: Kluwer Academic Publishers, 2000: 23–45.

## About the Author

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

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

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

Watts New

# Learning from Hardware: Design and Quality

Watts S. Humphrey

In the September column, I discussed what we could learn from hardware engineering, particularly about planning. Hardware engineers have developed a family of planning practices that they have used with great success. The prior column reviewed the pros and cons of using these planning methods for software work and discussed how these methods could help us meet our businesses' needs.

It is no fun to be late, to have unhappy customers, and to be unable to predict when you will finish a job. By following sound engineering planning methods, software work can be more productive, more predictable, and more enjoyable for the engineers themselves. In this column, I continue this same discussion with a focus on how engineering quality practices and design principles could be adapted to software and what we might gain from doing so.

## Hardware Costs

Hardware development strategies are dominated by cost considerations. The principal costs are those the factory incurs in producing the products, as well as those the service organization expends in handling product warranty and repair work. Even though we don't need a factory to produce volumes of software products, we do have large and growing service costs and we can learn a great deal from the ways in which hardware engineers have addressed quality and design problems.

## The Design Release

One of my first jobs when I joined IBM some years ago was to manage the development and release-to-manufacturing of a hardware product. We had built a working model and had complete parts lists, assembly drawings, and component specifications. While I thought we had a complete story, the manufacturing and service groups put us through the ringer for two exhausting days.

It took me a while to realize why they were being so difficult. They would not accept the release until we convinced them that our design provided the information they needed to meet cost, schedule, quality, and production volume commitments. Once they accepted the design release, these manufacturing and service groups would be committed to producing, warranting, and repairing these products on a defined schedule, with specified product volumes, and for the estimated costs. Their ability to do this would determine whether or not IBM made money on the product.

Since manufacturing and service were the two largest direct cost items for IBM's hardware products, these groups had learned how to manage costs and they were not about to accept a

release that had potential cost problems. While the manufacturing and service people were hard to convince, they imposed a valuable discipline on the development engineers. By making us produce complete, precise, and clear designs, they motivated us to think about manufacturing and service quality during design. This release discipline provided a solid foundation for all of the subsequent hardware quality improvement programs.

## The Need for Precise and Detailed Designs

Most hardware engineers quickly learn the importance of a precise and detailed design. On their very first jobs, they learn the difference between designing a laboratory prototype and releasing a design to the factory. Manufacturing groups will not accept a design release unless it provides the information they need to define the manufacturing processes, estimate the costs of production units, predict cost as a function of production volume, calculate warranty and service costs, order and fabricate all of the parts, and assemble and test the system. Many people need the design information and it is essential that they all get precisely the same story. Design documentation is also essential to enable the inevitable design changes and to track and control these changes.

## The Need for Documented Software Designs

The need for precise and documented designs in software is both similar to and different from hardware. There are five principal reasons to document a software design:

- to discipline the design work
- to facilitate design reviews
- to manage change
- to preserve and communicate the design to others
- to enable a quality and cost-effective implementation

Some people can hold very complex designs in their heads. However, regardless of how gifted you are, there is some upper limit beyond which you will no longer be able to do this. When you hit this limit, your design process will fail and the failure will not be graceful. Even very complex designs are not beyond our mental capacities when we follow sound and thoroughly documented design practices. Then we will not face the design crises that often result in complete project failures.

By documenting your designs, you also facilitate design reviews. This will both improve the quality of your designs and improve your personal productivity. The connection between quality and productivity is easy to see in hardware because a major redesign generally results in a lot of scrapped hardware. For software, however, these scrap and rework costs are equally significant, although not as visible. In the simplest terms, it is always more productive to do a design job correctly the first time than it is to do and redo the design several times.

Furthermore, a precise and documented design facilitates design changes. Anyone who has worked on even moderate-sized systems knows that, to control the inevitable changes, they must have precise records of the design before and after the change. Also, many programs will still be used long after their designers are no longer available, and many of these programs must be modified and enhanced. Without reasonably clear and complete design documentation, it is expensive to maintain or enhance almost any product. A well-documented design will add significantly to the economic life and value of the programs you produce. What is even more important, by increasing the economic value of your products, you also increase your personal value.

## Separate Implementation

Another reason to thoroughly document your designs is to facilitate the growing practice of subcontracting software implementation and test. You cannot efficiently use people in lower-cost countries to do this work unless you have a thorough and well-documented design. Otherwise, these off-shore groups would have to complete the designs themselves. This would waste much of the time and money the subcontract was supposed to save.

With a complete and well-documented design, the designers can move on to newer jobs while the implementing groups build and test the products. Without a complete and well-documented design, the designers will be needed throughout the implementation and test work. One way to ensure that the software designs are complete and implementable would be to require that the implementing groups review and sign off on the design before they accept a design release. While the software designers might initially object to this practice, it would impose the discipline needed to truly capitalize on the implementation and test talent potentially available in developing countries.

## New and Innovative Products

One argument against producing complete and well-documented designs is that software requirements are often imprecise and rapidly changing. When this is the case, the requirements, design, and implementation work must all be evolved together. This permits the users to test early product versions and to provide feedback on their improving understanding of the requirements. If these development increments are small enough and are done quickly enough, there will be fewer requirements changes to address and development can proceed rapidly and efficiently.

These requirements problems are most severe with new product development. However, in most established software groups, new product development is the exception. Only a small percentage of development time is generally spent on building new products. Most of the development work in most software organizations is devoted to repairing and enhancing existing products. Since the original designers are rarely available for this work, a documented design is needed to allow other groups to modify and enhance the original designs.

## Software Service Costs

Software service costs are largely a function of product quality. While the largest proportion of user service calls are usually for what are called no-trouble-founds (NTF), we once did a study and found that over 75% of these unreproducible NTF calls were attributable to latent product defects. NTF problems are enormously expensive. They waste the users' time and they require multiple service actions before they can be found and fixed. To minimize service costs and to reduce testing time and cost, early attention to quality is essential.

## Measure and Manage Quality

Quality products are not produced by accident. While most software professionals claim to value quality, they take no specific steps to manage it. In every other technical field, professionals have learned that quality management is essential to get consistently high quality products on competitive schedules. They have also learned that quality management is impossible without quality measures and quality data. As long as software people try to improve quality without measuring and managing quality, they will make little or no progress.

The lessons from hardware quality practices are instructive. Hardware quality problems have the same categories as software. They include requirements mistakes and oversights, design problems, and implementation defects. In addition, hardware groups must also worry about raw materials defects. Because of their rigorous design and design release procedures, most hardware manufacturing organizations find that their quality problems are generally due to manufacturing problems and not to design or requirements issues. This is why manufacturing quality programs concentrate almost exclusively on raw materials quality and on the quality of the manufacturing processes. The quality of the design is managed by the product developers and verified during the design release to manufacturing.

These manufacturing quality control practices are based on two principles. First, that the quality of the product is determined by the quality of the process that produced it. Second, that the quality of the process can be managed by measuring the work. The manufacturing engineers then use these measures to control and improve the manufacturing processes.

## Quality and Fix Time

One way to think about quality is to consider how the process would change as a function of defect fix times. For example, programmers generally think that it takes only a few minutes to fix defects in test. They base this on their experience with most of the defects they find in unit testing. In system test, however, the time to find and fix defects typically extends to many hours or even days. While most of these defects are fixed rather quickly, some take much longer. The average time to find and fix each defect is generally 10 to 20 or more hours.

Suppose, however, that the fix times in test were much longer, how would that affect the software process? The lessons from the hardware community are instructive. Some years ago, my laboratory had a small semiconductor facility for making special-purpose chips. The turn-around time for producing a custom chip from a completed design was six months. As a result, correcting any design or fabrication errors required at least six months. Since our products were for a highly competitive marketplace, the number of chip-fabrication turn-arounds was critical. The engineering objective was to release products with only one turn-around. With a little practice, their designs were of such high quality that they were generally able to meet that goal.

In the software business, the time to fix defects in final test is increasing and in some cases it can run into months. For example, for imbedded products like television sets and appliances, the general practice is to use more expensive technologies for the initial models so that they can quickly make any software corrections. Then, when the change rate drops sufficiently, they switch to a cheaper technology. As technology continues to get more complex and as competitive forces continue to increase, we will soon have to produce defect-free software before system test. At least we will for high-volume imbedded products. While testing will always be required, the software quality objective should be a one-cycle system test.

## Engineered Software

While the quality management methods for the software process are necessarily very different from those used in hardware manufacture, the same principles apply. In summary, these principles are: product quality is determined by process quality; produce and document clear, complete, and precise designs; and measure and manage quality from the beginning of the job. By following these principles, many software groups are now delivering defect-free products more predictably and faster than they ever delivered products before [Humphrey].


## Acknowledgements

In writing papers and columns, I make a practice of asking associates to review early drafts. For this column, I particularly appreciate the helpful comments and suggestions of Dan Burton, Julia Mullaney, and Bill Peterson.


## In closing, an invitation to readers

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

Thanks for your attention and please stay tuned in.


**Watts S. Humphrey**
watts@sei.cmu.edu

## Reference

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

# OCTAVE Developers Reach Out to Smaller Organizations with OCTAVE-S

Erin Harper

Small organizations can face big information security challenges. For example, a small doctor's office has the same responsibility for safeguarding patient information as a large chain of hospitals, but is not as likely to have adequate information technology resources at its disposal. Yet even though small businesses represent more than 99% of all employers and employ 51% of private-sector workers,[1] most approaches for evaluating information security risks focus on the needs of large organizations.

To meet the information security needs of small organizations, the SEI has developed a derivative of its Operationally Critical Threat, Asset, and Vulnerability Evaluation℠ (OCTAVE℠) Method: OCTAVE-S. "When we were developing OCTAVE, we met with people from many types of organizations to understand their requirements as potential users of the method," says SEI staff member Chris Alberts. "People who worked in small organizations typically liked the approach, but they needed some modifications to accommodate their staff compositions, schedules, and budgets."

The development of OCTAVE-S was sponsored by the SEI's Technology Insertion, Demonstration, and Evaluation (TIDE) program, which was created to help small manufacturing enterprises adopt state-of-the-practice technologies. While remaining consistent with OCTAVE's principles, OCTAVE-S provides smaller organizations with an efficient, inexpensive approach to identifying and managing information security risks.

## Using OCTAVE-S

To date, pilots of OCTAVE-S have been completed at two organizations, with positive results. The remainder of this article highlights the experiences of a small non-profit organization that participated in the first pilot.

With a staff of 80 people, the organization provides special services for member organizations and also collects "census" information about them, including yearly revenues. The chief information officer (CIO) of the organization became concerned about the protection of the sensitive information they were collecting and decided to conduct a security evaluation using OCTAVE-S.

---

1 U.S. Small Business Administration, May 2002 <http://www.sba.gov/advo/stats/sbfaq.pdf>.

## Preparation Activities

The CIO was able to get the support of the management team, an important first step whether the organization conducting the evaluation is large or small. He then chose employees to participate on the team that would conduct the analysis, ensuring that the team had both breadth and depth of knowledge about the organization. The CIO, chief financial officer, a system administrator, and a network administrator were chosen to participate.

## Phase 1: Build Asset-Based Threat Profiles

In this phase, team members determine what is important to the organization (assets) and how well those assets are protected. This phase can be considerably shorter in OCTAVE-S than in OCTAVE because analysis team members are likely to have insight into most or all areas of the organization, and formal knowledge-elicitation workshops are not necessary to gather information from disparate groups. The analysis team members identified about 40 information-related assets, and they determined that the following were the two most critical:

- customer relationship management system—contains sensitive membership data, including dues receipts, advertising receipts, and attendance lists for events
- accounting management system—used to manage cash flow throughout the organization

## Phase 2: Identify Infrastructure Vulnerabilities

The purpose of phase 2 is to examine an organization's computing infrastructure for technological weaknesses. However, no one at this organization had the experience or expertise to conduct such an evaluation, and there were no funds available to outsource the activity. They chose to acknowledge a gap in the organization's skill set and carried into phase 3 a recommendation that the organization develop an approach for conducting periodic vulnerability evaluations of the computing infrastructure.

## Phase 3: Develop Security Strategy and Plans

After looking at the information gathered throughout the evaluation, the team identified a broad range of risks to each critical asset. For example, they determined that staff members or people external to the organization (attackers) could exploit technological weaknesses to view sensitive customer data or interrupt access to systems. This could irrevocably destroy the organization's reputation, resulting in a reduced number of member organizations and the loss of revenue. Staff work hours could increase by 50% for more than five days to bring an attacked system back up and to complete tasks that could not be addressed while it was unavailable.

After reviewing the risks to critical assets and discussing how the exploitation of those risks could affect the organization's business processes, the analysis team identified the top three areas in which the organization should improve:

1. Vulnerability management: Internal or external people might be able to exploit technological weaknesses in the computing infrastructure, enabling them to view or interrupt access to sensitive customer data.

2. Contingency planning: If any of the major risks affecting business operations were to occur, the organization's down time would likely be prolonged because it had no defined plans for continuity of operations.

3. Physical access control: Physical access to the organization's confidential files and its computer systems was poorly controlled, if at all.

## After the Evaluation

The team was able to begin making security improvements while the evaluation was still underway. For example, the organization purchased a backup server for its customer relationship management system and developed a sign-in procedure for people entering the building. Some of the remaining improvements were larger and would require more time and money. The evaluation helped the management team understand the relationship between the security threats identified and their impact on the organization's mission and business objectives, so they allocated funds for each of the three recommended improvement areas and increased the overall budget for the IT department. "The CIO said he had been trying to get increased funding for several years, but had not been able to convince management that security was important enough to invest in," says Alberts. "One thing that OCTAVE does best is relate security issues to an organization's business objectives."

An implementation guide for OCTAVE-S is in development. For more information about OCTAVE-S and licensing opportunities, visit the OCTAVE Web pages at <http://www.cert.org/octave/>.

For more information, contact—

Bob Rosenstein
**Phone**
412-268-8468

**Email**
br@sei.cmu.edu

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

# SEI Hosts Software Product Line Conference

Pamela Curtis

The SEI sponsored the second Software Product Line Conference (SPLC2) in San Diego, California, August 19–22, 2002. SPLC2 brought together researchers and practitioners in the software community to push the frontiers of software product line technology and to discuss topics such as understanding and managing variability in product lines, organizational and business issues for product lines, and product-line life cycles.

There were 157 attendees, roughly two thirds from the United States and the remaining third from 16 other countries spanning North America, Europe, Asia, and Africa. Most of the participants were from commercial organizations, but academia and government (especially through government contractors) were well represented. Most of the software product line leaders participated, including Hewlett Packard, Nokia, Philips, Robert Bosch GmBh, Avaya, Motorola, Cummins, Siemens, Ericsson, Thales, and General Motors.

Anders Heie, a software specialist from Nokia Mobile Phones, presented an inspiring keynote talk, "Global Software Product Lines and Infinite Diversity." Heie described the software architecture and processes that Nokia has created to support the demands of a wide range of new products each year, with requirements coming in from around the world. Other events featured in the conference were seven tutorials; seven workshops, including one that was focused on DoD product line practice; two panels; 24 technical paper presentations; four demonstrations of software product line engineering tools; and several birds-of-a-feather sessions. Heie's keynote and other presentations are available on the conference Web site (Web addresses are listed at the end of the article), along with information about ordering the conference proceedings.

Continuing a practice established at SPLC1, several exemplary product lines were voted into the Software Product Line Hall of Fame. The purpose of the Hall of Fame is to improve software product line practice by identifying product lines that represent the highest achievement in the field. Nominations submitted by participants in the last session of the conference generated discussions about what constitutes excellence and success in product lines. Participants then voted on the nominees based on these election criteria:

- The family that constitutes the product line is clearly identified, that is, there is a way to tell whether or not a software system is a member of the product line, either by applying a known rule or a known enumeration.
- The family that constitutes the product line is explicitly defined and designed as a product line, that is, the commonalities and variabilities that characterize the members of the product line are known and there is an underlying design for the product line that takes advantage of them.
- The product line has strongly influenced others who desire to build and evolve product lines, and has gained recognition as a model of what a product line should be and how it should be built. Others have borrowed from it in creating their product lines or in expounding ideas and practices for creating product lines.

- The product line has been commercially successful.
- There is sufficient documentation about the product line so that one can understand its definition, design, and implementation without resorting solely to hearsay.

SPLC2 inductees included Cummins's diesel engine software product line, Philips's telecommunications switching system, Lucent Technologies' 5ESSTM telecommunications switch, Boeing's Bold Stroke avionics software family, and Market Maker Software AG's Merger software product line. More information about these product lines, as well as descriptions of previous inductees, is available on the SPLC2 Hall of Fame site.

The conference evaluations were overwhelmingly positive. The following were among the comments received:

> *I very much enjoyed the conference and liked most of the presentations I attended, a much higher percentage than many other conferences. Also the variety of the attendees (many industrials, many Europeans).*
>
> *The conference was a great success. I enjoyed it very much both from the professional and social viewpoint.*
>
> *The tutorials were exceptional.*

The content and the associated discussions marked a decidedly more mature software product line community than was evidenced at SPLC1—one not without challenges, but one with success stories and approaches to share.

SPLC3 will be held in the United States in autumn 2004. An SPLC Steering Committee that includes SEI and international product line leaders has been formed to direct the conference in coming years.

For more information about the SEI's work in product line practice, visit the Product Line Systems Web site, and read the Software Product Lines column in *news@sei interactive*.

## On the Web

**SPLC2**
http://www.sei.cmu.edu/SPLC2/?si

**SPLC2 Hall of Fame**
http://www.sei.cmu.edu/SPLC2/SPLC2_hof.html?si

**Product Line Systems**
http://www.sei.cmu.edu/programs/pls/pl_program.html?si

For more information, contact—

Customer Relations
**Phone**
412-268-5800

**Email**
customer-relations@sei.cmu.edu

**World Wide Web**
http://www.sei.cmu.edu/programs/pls/pl_program.html?si

# Managing Risks in Modernizing Legacy Systems

Claire Dixon

Software must evolve to remain useful. According to the first law of M.M. Lehman, a renowned software evolution researcher, "A large program that is used undergoes continuing change or becomes progressively less useful."[1] Lehman states in his second law that "As a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it."[2] Years of accumulated code changes lead to less maintainable code. Many organizations that depend on legacy systems are struggling with how to modernize those systems, but are unable to adequately address the potential risks. A new book on risk-managed modernization provides the needed guidance. *Modernizing Legacy Systems: Software Technology, Engineering Processes and Business Practices*, to be published in February by Addison-Wesley, was written by Robert Seacord, Dan Plakosh, and Grace Lewis, members of the SEI technical staff. The book introduces a legacy-system modernization approach that accounts for both software technology and business objectives. The book describes engineering processes and business efforts that can be used in planning, justifying, and executing the modernization effort.

## The Legacy Crisis

The book meets the need for an effective modernization approach. The Standish Group reported that 23% of software and information-technology projects were cancelled before completion, while only 28% finished on time and budget with expected functionality.[3] Because of the huge growth in computers for business process support, an estimated 250 billion lines of source code are being maintained, and that number is increasing.[4] The relative cost for maintaining software and managing its evolution now represents more than 90% of its total cost. If trends continue, eventually no resources will be left to develop new systems. Seacord and his co-authors refer to this as the legacy crisis.

Legacy systems evolve through maintenance, replacement, or modernization. Maintenance can effect only limited change. Modernization can be a cost-effective option to replacement. However, modernizing software is daunting in that it requires careful analysis of options, stakeholder interests, and multi-faceted tradeoffs, including technical, programmatic, and organizational considerations.

---

1 Lehman, M.M. & Belady, L. *Program Evolution: Processes of Software Change*. London: Academic Press, 1985.

2 ibid.

3 The Standish Group. *CHAOS Chronicles II*. West Yarmouth, MA: The Standish Group International, Inc., 2001.

4 Sommerville, I. *Software Engineering, 6th Edition*. New York: Addison-Wesley, 2000.

## Risk-Managed Modernization

Typically, development teams approach large modernization efforts either with trepidation and misgivings or with unwarranted self-confidence. “The former group is usually the one with the most experience,” says Seacord. “Fortunately, there is a middle road, which is to approach the modernization with respect for the magnitude of the effort and a plan.” This plan must incorporate an understanding of the legacy-system technologies, engineering processes, and business requirements. Finally, this plan must manage risk throughout the modernization effort.

This approach is introduced in the book as risk-managed modernization (RMM). RMM integrates software engineering concepts with an organized understanding of the information- systems technologies that define the range of possible solutions. The approach integrates risk management at every step—continuously assessing what can go wrong, determining what risks are important, and implementing strategies to deal with those risks.

The book shows how legacy systems can be incrementally modernized. It uses and extends the methods and techniques described in the book *Building Systems from Commercial Components*[1] to draw upon engineering expertise early in the conceptual phase to ensure realistic and comprehensive planning. The book features an extensive case study involving a major modernization effort. The legacy system in the case study consists of nearly 2 million lines of COBOL code developed over 30 years. The system is being replaced with a modern system based on the Java 2 Enterprise Edition (J2EE) architecture. Additional challenges include a requirement to incrementally develop and deploy the system.

## Finding a Better Way

The authors also emphasize the importance of creativity and innovation in addition to process. It is crucial, they write, to find a better way of doing things rather than trying to improve the way things have been done in the past. Many modernization efforts simply replace old legacy code with new legacy code. While this approach extends the useful lifetime of the legacy systems, it often results in re-codifying existing, outdated business practices. When deciding how aggressively to approach a modernization effort, consider when the next opportunity to modernize will come along, the authors suggest. On the other hand, being too aggressive in modernization plans means increasing risks that could lead to project failure. Therefore, plans must be well considered and plausible. “In our experience,” the authors state, “the best ideas actually reduce overall risk while providing needed capabilities.” Finding a “better way” may mean ignoring the existing system and processes and considering the underlying principles instead. This may provide a clearer vision of goals and allow streamlining of system and processes to this vision.

---

1 Wallnau, K.; Hissam, S.; & Seacord, R. *Building Systems from Commercial Components*. Boston, MA: Addison-Wesley, 2001.

In his review of *Modernizing Legacy Systems*, M. M. Lehman asserted that the book provides needed guidance where very little exists. “I recommend it as a must for people directly involved in legacy system modernization, whether as customers and resource providers or as implementers. . . [It] should also be of significant interest to those who seek a better understanding of the evolution and legacy phenomena, and given the current. . . reliance on computers, that means all of us.”

For more information, contact—

Robert Seacord
**Phone**
412-268-7608

**Email**
rcs@sei.cmu.edu

**World Wide Web**
http://www.sei.cmu.edu/activities/cbs/mls/?si

# CMMI Myths and Realities

Lauren Heinz

*CMMI is too big and complex.*

*A CMMI appraisal takes longer and costs more than an appraisal for SW-CMM.*

*CMMI is only for large organizations.*

These and other myths are often heard in discussions about upgrading from the SW-CMM to a CMMI model. But those who have been using the new set of models and training materials contend that making the switch to the CMMI Product Suite is not only easier than it looks, but well worth it.

"For organizations already operating at a high Maturity Level, the process of achieving CMMI is very straightforward," says Sarah Bengzon, an associate partner at Accenture, a leading management consulting and technology services organization. "People think that with CMMI, everything is new and that the process is too complex to undertake. But at Accenture, we have always been doing things this way. If anything, CMMI validates the best practices we already had in place."

In fact, Accenture's USA Government Operating Unit, which is an early adopter of the CMMI Product Suite, attained Maturity Level 3 just eight months after upgrading from the SW-CMM model. "CMMI enforces tying project objectives to organizational objectives, which is not only a good thing to do, but a bad thing not to do," Bengzon says. "CMMI shows you exactly what you should be doing to improve your quality processes."

Accenture's group is just one of hundreds making the switch to CMMI Version 1.1 worldwide. Since its release in December 2001, nearly 4,000 people have attended the Introduction to CMMI course offered by the SEI and its transition partners, 85 instructors have been trained to teach the introductory course, and more than 120 individuals have become authorized SCAMPI Lead Appraisers℠. "Initial acceptance of CMMI seems to be much faster than it was for SW-CMM," says Bill Peterson, director of the Software Engineering Process Management Program at the SEI.

While some myths from the earlier development and piloting days of the CMMI models are still circulating, Bengzon and others are proving that these misconceptions are easy to clear up with a little guidance from the experts.

## "CMMI is too big and complex"

For more than 10 years, the SW-CMM model has been the global standard for appraising and improving software processes. As organizations came to know and experience the value of the SW-CMM model and other CMM models, these organizations sought to expand the use of the CMM concept beyond its initially defined scope. This evolution of the CMM concept naturally grew into the development of the CMMI Product Suite. Its purpose is to provide guidance for improving an organization's processes and its ability to manage the development, acquisition, and maintenance of products and services. The CMMI Product Suite places proven practices into a structure that helps an organization appraise its organizational maturity and process capability, establish priorities for improvement, and guide the implementation of these improvements. But at 700+ pages each, the CMMI models can seem a bit daunting.

Roger Bate, principal architect of the CMMI Product Suite, says the models are so lengthy because they provide comprehensive guidance and details. "It's similar to a dictionary," he says. "There are a lot of words in there that you'll never need to look up, but they're there so they can be available to everyone when and if they need them."

The most obvious additions to the models are related to integrated product and process development (IPPD), which now includes two additional goals and two new process areas (PAs) called Integrated Teaming and Organizational Environment for Integration. SW-CMM's single Software Product Engineering key process area was expanded into five, more comprehensive PAs in the CMMI Product Suite. A Measurement and Analysis PA at Maturity Level 2 and a Decision Analysis and Resolution PA at Level 3 were also added to the models.

"But don't let the page count throw you," Bate says. He recommends three ways that an organization can reduce a model's size and complexity:

1. Select the right model. There are several CMMI models to choose from, including CMMI for Software, CMMI for Systems and Software Engineering (SE/SW), CMMI SE/SW with IPPD, and CMMI SE/SW with IPPD and Supplier Sourcing (SS). Once you select a model, tailor it to fit your organization's needs.

2. Don't try to implement the whole model at once. "Select those parts that are most applicable and will have the biggest payoff at the first stage of process improvement," Bate says. "Get at those things which are most important: improving quality, predicting costs and schedules, and reducing time to market. Develop a base from which you can move forward."

3. Follow the practices that make the most sense for your organization. "You can pick and choose or substitute your own processes as long as they meet the overall goals. Every subpractice doesn't need to be implemented. They are informational guides, not requirements," he says.

## "A CMMI appraisal takes longer and costs more than one for SW-CMM."

"While it is understandable that these kinds of sweeping statements will be made," says David Kitson, manager of the SEI's appraisal program, "the reality is that the two appraisal methods have some subtle but significant differences that make such direct comparisons less than useful."

For example, the SCAMPI method, which is used to appraise an organization's use of CMMI best practices, is designed as an Appraisal Requirements for CMMI (ARC) class A appraisal method. It is intended for use where the highest confidence and accuracy is desired on the part of the appraisal sponsor. In contrast, the CMM-Based Assessment for Internal Process Improvement (CBA IPI) method was designed as a replacement for the Software Process Assessment method. The primary design goal with CBA IPI was to minimize unexplainable differences in appraisal results produced by CBA IPI and the Software Capability Evaluation (SCE) methods. It was not designed primarily for benchmarking, so its requirements and features differ from those of SCAMPI.

With that understood, however, it can be said that a lot of thought and design effort was directed toward reducing the effort required to conduct SCAMPI appraisals. After hearing two years ago from early adopters that the SCAMPI method was often taking 150 or more hours for a Maturity Level 3 appraisal, the CMMI Product Team adopted a stretch goal: Maturity Level 3 appraisals would take no longer than 100 hours on site.

Kitson worked with a team of appraisal experts from industry, government, and the SEI who redesigned the SCAMPI method. Since the first round of SCAMPI Version 1.1 lead appraiser training in April, Kitson says he and his staff have seen a number of SCAMPI appraisals performed, and they are very satisfied with the results. One defense contractor has reported conducting its Level 3 SCAMPI appraisal in just 60 hours.

"We are seeing in practice the realization of the benefits we expected SCAMPI V1.1 would provide," Kitson says. "The organizations that are reaping the maximum benefits that SCAMPI offers are the ones that are taking the time to make genuine improvements in their processes and to treat process improvement just as they would any other project they undertake."

"The bottom line is that direct comparisons of appraisal costs-especially in the absence of consideration of other relevant factors-is not a very useful exercise," Kitson says. "Ultimately, it all goes back to why process improvement is being undertaken in the first place. If the goal is world-class performance, the focus of attention should be on more substantive issues, such as ensuring that the SCAMPI method is applied in a way that minimizes the risk that any process-related issues escape the notice of the appraisal team."

## "CMMI is only for large organizations."

Not true, Bate says. Although the CMMI models were developed in part to help larger organizations tackle complex issues across multiple disciplines, they can be tailored to meet the needs of smaller companies, especially if the companies apply the models purely for process improvement. "Many smaller organizations aren't interested in using CMMI to benchmark themselves against other organizations. They don't need to commit to all the process areas and practices. CMMI is designed so they can pick and choose among the PAs that have the most immediate applicability to them," he says.

The CMMI Product Suite is currently being applied across a range of organization sizes, from large to small.

"Planning, measurement and analysis, monitoring and control, and requirements management are just solid, basic parts of what everyone should do, no matter what business they're in or what they're building," Bate says. "It just makes sense to be doing those things, regardless of how large or small the organization might be."

## Ongoing Guidance for Software Organizations

The SEI is working with software-only organizations that may need additional guidance when interpreting the CMMI models for software-specific applications. Sessions at community events are being offered to collect user experiences and address questions from software and information technology organizations, the first of which took place at the recent CMMI Technology Conference and User Group in Denver, Colorado. "The community's involvement is key to this effort as the SEI begins to understand and address the issues that software organizations face when migrating to or using the CMMI Product Suite," says Mary Beth Chrissis, the team lead for this effort at the SEI.

Organizations are encouraged to contact the SEI if they are interested in receiving additional guidance on the CMMI Product Suite or if they would like to share their CMMI implementation experiences.

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