Carnegie Mellon
Software Engineering Institute

news @ sei interactive

Volume 4 | Number 2 | Second Quarter 2001

In This Issue

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

news@sei interactive

From the Director

In this issue of news@sei, one of our articles, “Securing Information Assets”, refers to a newly published volume in the Addison-Wesley SEI Series in Software Engineering. The CERT® Guide to System and Network Security Practices, written by Julia Allen, a senior member of the SEI technical staff and published in June, presents more than 50 best practices that address all phases of securing systems and networks. It shows administrators how to protect systems and networks against malicious and inadvertent compromise based on security incidents reported to the CERT Coordination Center. For more information, see the book’s Web page on the Addison-Wesley Web site (http://cseng.aw.com/book/0,,020173723X,00.html).

The Addison-Wesley SEI Series in Software Engineering complements other means that we use to provide information to the software engineering community, such as the SEI Web site (www.sei.cmu.edu), the CERT Web site (www.cert.org), news@sei, news@sei interactive (www.sei.cmu.edu/interactive?si) and the Software Engineering Information Repository (seir.sei.cmu.edu). It provides software engineering practitioners with current, in-depth information that supports their use of mature and continually improving software engineering practices. The works in the series, like all SEI technical work, are intended to improve the state of the practice of software engineering; that is, they are practitioner oriented, not theoretical.

The CERT® Guide to System and Network Security Practices is one of seven new books that will be published in the SEI Series this year. Other planned releases for 2001 are

- CMMI Distilled: A Practical Introduction to Integrated Process Improvement, by Dennis Ahern, Aaron Clouse, and Richard Turner
- Managing Software Acquisition: Open Systems and COTS Products, by B. Craig Meyers and Patricia Oberndorf
- Building Systems from Commercial Components, by Kurt C. Wallnau, Scott A. Hissam, and Robert C. Seacord
- Software Product Lines: Practices and Patterns, by Linda Northrop and Paul Clements
- The People Capability Maturity Model®: Guidelines for Improving the Workforce, by Bill Curtis, William E. Hefley, and Sally A. Miller
- Evaluating Software Architectures: Methods and Case Studies, by Paul Clements, Rick Kazman, and Mark Klein

Most books published in the SEI Series are based on SEI work, but we also publish some books that are based on non-SEI work that complements the SEI technical program and helps to extend the practice of software engineering. I want to express my gratitude and admiration for both SEI and non-SEI authors, who largely donate their own time and energy to writing the books in the SEI Series. For a list of all books currently in the SEI Series, see http://www.sei.cmu.edu/products/publications/sei.series.html?si. For more information about the SEI Series, please contact Bill Pollak, SEI public relations coordinator (wp@sei.cmu.edu), the SEI point of contact for the SEI Series.

Other features in this issue include “The TIDE Program: Strengthening the Defense Manufacturing Base”. In this article, we look at the Technology Insertion, Demonstration, and Evaluation (TIDE) Program, which seeks to improve the profitability and efficiency of small defense and commercial manufacturers by helping them overcome the barriers to technology adoption. We also discuss the Software Engineering Information Repository (SEIR), a Web-based forum for software engineers in the field to exchange lessons learned, pose questions, and submit materials that might help others to adopt process improvement approaches. “Software Architecture Evaluation: A Key to System Success” describes the Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}), which allows an organization to analyze a software or system architecture to determine its quality attributes, such as modifiability, security, and performance. Finally, “Overcoming the Challenges of COTS” describes the Information Technology Solution Evaluation Process (ITSEP), which helps organizations to develop, field, and support commercial off-the-shelf (COTS)-based solutions.

Thanks for reading, and please send your comments and suggestions to news-editor@sei.cmu.edu.

Stephen E. Cross
Director, Software Engineering Institute

Securing Information Assets

Julia Allen

“Thousands of Computer Networks Compromised,” “Customer Credit Card Numbers Posted on the Web,” “Government Web Site Defaced Again,” “Email Attachment Delivers Destructive Payload.” How many times have we read headlines like this in the last year? People who attack computer networks and systems are determined to leave their mark, steal data, or break into your system for a host of other reasons. Many malicious intruders are armed with sophisticated tools and knowledge of the latest computer vulnerabilities. How can we stop them?

The Networked Systems Survivability program at the SEI is dedicated to finding solutions for keeping networked systems secure. Networked systems and the sensitive information they contain can be compromised despite an administrator’s best efforts. Even when administrators know they should be devoting more time to security, they often cannot: keeping systems functioning takes priority over securing those systems. Administrators need a clear and comprehensive set of security practices that are easy to find and follow.

The CERT® system and network security practices represent more than fifty best practices to address all phases of securing systems and networks. CERT security practices are organized into five broad groupings based on the order in which they are performed. They are

- Harden and secure
- Prepare to detect and respond to intrusions
- Detect intrusions
- Respond to intrusions
- Improve

The complete set of practices is available on the CERT Web site at http://www.cert.org/security-improvement/index.html. Also, this summer Addison-Wesley will publish a book entitled The CERT Guide to System and Network Security Practices, written by Julia Allen of the SEI.

Figure 1 illustrates how to secure and protect information assets (such as a network or Web server) using the CERT security practices.

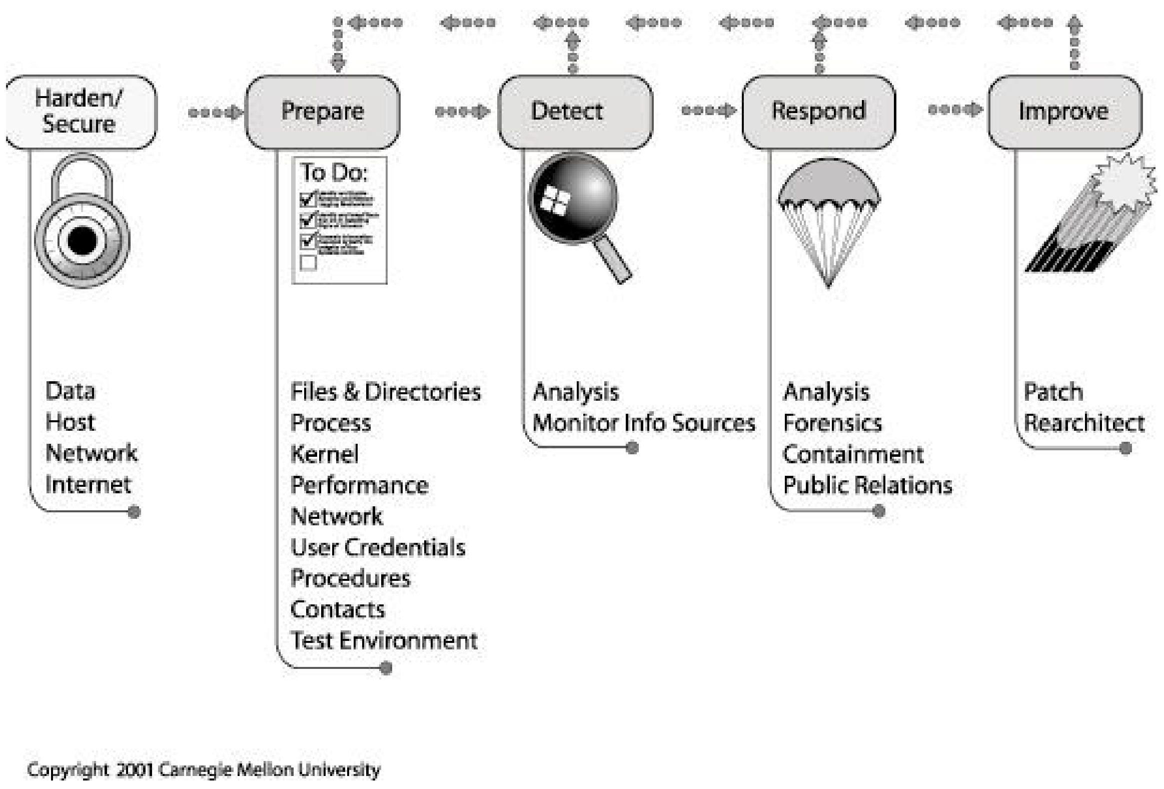

Figure 1: The Structure of CERT Security Practices

The CERT practices were created to address 75 to 80 percent of the problems that are reported to the CERT Coordination Center (CERT/CC) at the SEI. The practices do not make reference to any one operating system or version, so the principles will remain valid despite the rapid advances in technologies. However, many practices are linked to documents called implementations that do discuss specific operating systems and software. Implementations are available from http://www.cert.org/security-improvement/#implementations.

The Five Steps

1. Harden and Secure

   Systems shipped by vendors are very usable but, unfortunately, often contain many security weaknesses. This step yields a hardened (secure) system configuration and an operational environment that protects against known attacks for which there are designated mitigation strategies.

2. Prepare to Detect and Respond to Intrusions

   This step starts with the assumption that there are many vulnerabilities not yet identified. An administrator must be able to recognize when an unknown vulnerability is being exploited. How can an administrator recognize what is not there? The major method to help recognize exploitation is to characterize a system so that an administrator can understand how it works in a production setting. Then, any deviations will provide the clue to noticing unexpected or suspicious activity.

   In addition, the administrator and his or her manager must develop policies and procedures to identify, install, and understand tools for detecting and responding to intrusions well before they need to be invoked.

3. Detect Intrusions

   This step occurs when an administrator is monitoring system or network transactions by, for example, looking at the logs produced by a firewall system or a public Web server. The administrator notices some unusual, unexpected, or suspicious behavior; learns something new about the asset's characteristics; or receives information from an external stimulus (a user report, a call from another organization, a security advisory or bulletin). These indicate either that something needs to be analyzed further or that something on the system has changed or needs to change (a new patch needs to be applied, a new tool version needs to be installed, etc).

4. Respond to Intrusions

   In this step, an administrator further analyzes the damage caused by an intrusion (including the scope and effects of the damage), contains these effects as much as possible, works to eliminate future intruder access, and returns information assets to a known, operational state. It may be possible to do this step while continuing analysis. Other parties that may be affected are notified, and evidence is collected and protected in the event it should be needed for legal proceedings against the intruder.

5. Improve

   Administrators also need to take action to improve their systems following detection or response activities. In addition to practices contained in the step Detect Intrusions, improvement actions may include

   - further communicating with affected parties
   - holding a post mortem meeting to discuss lessons learned
   - updating policies and procedures
   - updating tool configurations and selecting new tools
   - collecting measures of resources required to deal with the intrusion and other security business case information

Even when system administrators know how to secure systems, they often don't have the time to take action. The CERT security practices, organized into five top-level steps, provide administrators with guidance that is easy to access, understand, and implement.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

World Wide Web
http://www.cert.org/security-improvement/index.html

The TIDE Program: Strengthening the Defense Manufacturing Base

Bill Pollak

Like other sectors within the U.S. economy, the defense manufacturing base in the U.S. is evolving. Increasingly, product development is being outsourced to small manufacturing enterprises, and large defense-contractor organizations are becoming integrators of supply chains, as opposed to manufacturers. A supply chain is only as strong as its weakest link. As the defense manufacturing base evolves, these links will be crucial for rapid defense response to future events, especially regional engagements.

Small manufacturers have typically been reluctant to utilize software technology in their design and manufacturing activities. As easier-to-use, less costly tools have been developed to aid the manufacturing process, this reluctance has prevented small manufacturers from participating in the information technology-driven changes that have fueled economic growth and improvement in other sectors. The Technology Insertion, Demonstration, and Evaluation (TIDE) Program, initiated in May 2000, seeks to improve the profitability and efficiency of small defense and commercial manufacturers by helping them overcome the barriers to technology adoption.

To improve the capabilities of smaller manufacturers, the SEI is helping such enterprises adopt state-of-the-art software practices and technology through the TIDE Program. Funded through an appropriation sponsored by Congressman Mike Doyle, the TIDE Program supports the SEI's mission to transition best software engineering practices into widespread use. Advanced software engineering practices—advanced engineering environments, modeling and simulation capabilities, and better performing manufacturing control systems—promise to provide small manufacturers with

- higher quality
- predictable performance
- reduced cycle time
- reduced costs

Demonstration Projects

Currently, two small manufacturers in Southwestern Pennsylvania—Carco Electronics and the Kurt J. Lesker Company—are investing time and engineering personnel to collaborate with the SEI on projects demonstrating the business benefits and process of adopting advanced technology in small manufacturing enterprises.

While solving specific problems for the companies that participate in these demonstration projects, the SEI is documenting lessons learned. The intended outcome of these demonstration projects will be a toolkit that can be used by any smaller manufacturer attempting to establish an enhanced engineering and design capability, to move into new markets, and to provide more value to customers in the form of more technically sophisticated products. Case studies generated from these demonstration projects will provide solid justification to investors as well as to risk-averse owners of smaller businesses that insertion of commercially available software does have substantial benefit. These case studies and other lessons learned from the demonstration projects will then be shared in forums such as workshops, conferences, and curricula so that others in the DoD supply chain can take advantage of this work. Carco Electronics manufactures multi-axis rotational devices for testing inertial navigation and missile-seeker systems. The SEI is working with Carco to evaluate Carco's engineering process and adopt commercial technologies for improvement. The SEI and Carco Electronics are currently pursuing the adoption of new tools to support structural, kinematics, and servo-analysis capability within Carco's engineering process. As the technology adoption plan continues to be implemented, additional results will be collected and documented.

"The TIDE program is helping Carco Electronics to expand our technical analysis capabilities, enabling us to be more competitive in the global aerospace market," says Joe Elm, general manager for Carco Electronics. "This will drive our continued growth."

The Kurt J. Lesker Company manufactures ultrahigh vacuum systems used in the production of a variety of products including semiconductors and flat-panel displays. An assessment of the Lesker engineering process found that it could be improved through the adoption of tools required to efficiently design the more complex products demanded by its customers. The SEI is helping Lesker evaluate and select appropriate commercial software tools that meet the company's engineering requirements. In the process of doing so, the SEI is determining what evaluation processes are suitable for use by small manufacturers.

"We are already doing things smarter," says Kurt Lesker III, president of Kurt J. Lesker Company," and as soon as we get the new tools in, we will be doing them better as well."

Workforce Development

Concurrently with the TIDE demonstration projects, the SEI has initiated an education-and-training outreach program to expand technology adoption throughout the Southwestern Pennsylvania manufacturing community. This program offers scholarship support for small business personnel to attend courses, seminars, and workshops in leading-edge information technology, leading to increased awareness of the value of and return on investment from technology adoption.

“Western Pennsylvania has a tremendous industry base, manufacturing metals, medicine, and a great deal more,” says Congressman Mike Doyle. “In addition to traditional industries, we are developing a new kind of factory—one that manufactures knowledge based on research. The SEI is a prime example of this. It was created by the Department of Defense to transform research results into knowledge-based products that others can use to build better software. This type of knowledge factory is very important. It is my belief that the real benefit of the technological revolution will not be fully realized until our traditional factories reap the benefits produced by our knowledge factories. The TIDE Program is making this happen.”

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

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

SEIR Reaches 10,000 Users

Lauren Heinz

As a systems manager for a major telecommunications company in Mexico, Teodoro Cortés is searching for ways to improve the quality and maturity of his organization's software practices. Quite often, he finds the solutions in the SEI's Software Engineering Information Repository (SEIR).

The SEI developed the SEIR as a forum for software engineers in the field—from government, industry, and academia—to exchange lessons learned, pose questions, and submit materials that might help others to adopt improvement approaches. The SEIR provides a repository of information showing the impact of demonstrated software engineering improvement methods on organizational performance.

For managers such as Cortés, the Web-based SEIR provides the practical information necessary to successfully transition new software processes, and provides it in a global software environment where other engineers with similar interests can be tapped for their expertise. Since its launch in 1998 with a mere 104 users and minimal site content, the SEIR has grown to become one of the most highly visited areas of SEI-operated Web sites. Its members represent nearly 5,000 organizations in 80 countries, and it includes more than 10,300 Web pages and 300 documents. Recently, Cortés confirmed the SEIR's success by putting the site over the 10,000-members mark.

SEIR Contents

The SEIR is organized into six information domains: Capability Maturity Model® (CMM®) for Software (SW-CMM), CMM Integration^{SM}, Measurement and Metrics, People CMM, Personal Software Process^{SM} and Team Software Process^{SM}, and Software Risk Management. Each domain features a general introduction to the topic, a list of frequently asked questions (FAQ), and information on impact implementation and results. The SEIR is of use to software practitioners and managers, Software Engineering Process Groups (SEPGs), Software Process Improvement Network (SPIN) groups, lead appraisers, and other sponsors of improvement efforts.

In addition to CMM-based software improvement information, the SEIR also features an interactive maturity profile. The maturity profile summarizes the aggregate appraisal results submitted to the SEI and shows the status and trends of process maturity for the software community. More than 20,000 copies of the current profile have been downloaded since its release in March 2001. Other active areas of the site include common organizational issues preventing SW-CMM key process area (KPA) satisfaction, information on cultural change in the workplace, and tips for defining and documenting software processes for the SW-CMM.

Involvement is Key

The growth and longevity of the SEIR depends upon the exchange and contribution of information from the software engineering community. “Most people go to a Web site, use the site’s search engine to hastily grab information that might not be right for them, and leave,” says Mike Zuccher, the SEIR’s Webmaster and project manager. “We are a little different in that we ask members to browse and examine the information to find what best suits their needs. The second difference is that we encourage members to share their experiences by contributing information to the site so that they can build a global knowledge base for improved implementation efforts for now and for the future.” Users must register to become members of the SEIR, and membership permissions are granted based on the contribution and exchange of information. Membership is free and open to the public, but a registration process ensures a level of information integrity. Members are rewarded for frequent contributions to the site by receiving special access privileges to additional information and permanent account activation.

From the beginning, Zuccher has paid close attention to the responses he receives from registration forms and feedback. As more and more members started asking for “real world” examples of software improvement, Zuccher realized that if the site could be developed around the exchange of best practices, members of the software community could help themselves. “It was the SEIR members who pointed the site in this direction,” he says. Now the site is populated with implementation data including lessons learned, case studies, return-on-investment information, and tips for encouraging upper management to adopt improvement efforts.

Registered members are asked to submit white papers, presentations, technical reports, executive summaries, or tutorials that demonstrate successful implementations. This information can help others in the software community make informed decisions when considering adopting an improvement approach. Users can also participate by answering other members’ queries in the “Ask the Group” section, a facility where SEIR members can post questions, view responses to questions made by other members, and exchange information.

New Directions

User feedback and submissions will continue to guide the development of the SEIR. Zuccher is currently investigating several interesting Web-based technologies to allow for easier collaboration among members for document exchange and “Ask the Group” communications.

The SEIR is also exploring ways to customize the site for its members. One project the group has discussed is the ability for each SEIR member to create a personal profile that lists their content and technology preferences for the site. Based on their preferences, they could be sent an automated list of documents that would best serve their needs.

Accessing the SEIR

The SEIR is currently free to the public, but you must register to become a member. To register, please visit the SEIR Web site at http://seir.sei.cmu.edu?si.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

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

Software Architecture Evaluation: A Key to System Success

Most complex software systems are required to be modifiable and have good performance. They may also need to be secure, interoperable, portable, and reliable. But for any particular system, what precisely do these quality attributes—modifiability, security, performance, reliability—mean? Can a system be analyzed to determine these desired qualities?

A system's quality attributes are largely permitted or precluded by its architecture. So why should an organization analyze a software or system architecture? Quite simply, because it is a cost-effective way of mitigating the substantial risks associated with this highly important artifact. With its Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}), the SEI is providing a way to perform this task.

About ATAM

A software architecture is a representation of a software system in terms of its components, their externally visible properties, and their relationships. It serves as a blueprint of the system and the basis for communicating, analyzing, and reusing key design decisions. Architectures allow or preclude nearly all of the system's quality attributes such as performance, modifiability, usability, and availability.

The ATAM evaluates architectural decisions in light of quality attributes requirements and reveals how quality attributes interact with each other. It also highlights the relationship between quality attributes and the business drivers (for example, affordability, time to market) that motivate them.

The ATAM elicits sensitivity points, tradeoff points, and risks. Sensitivity points are architectural decisions that dramatically affect a quality attribute. Tradeoff points are architectural decisions that dramatically affect more than one quality attribute. Risks are a subset of sensitivity points that potentially can inhibit the system from achieving its quality attribute goals. The ATAM consolidates risks into risk themes and then examines the impact of risk themes on the organization's business drivers. By identifying sensitivity points, tradeoff points, and risks, software developers can examine the consequences of architectural decisions before committing resources to implementing them.

The ATAM offers a number of additional benefits. It helps software developers and other stakeholders clarify quality attribute requirements, leads to improved architecture documentation, and provides the reasoning for architectural decisions. It also enhances communication among stakeholders.

Experience with ATAM

Over the past several years, SEI technical staff members have performed about 20 architecture evaluations for defense, non-defense government, and commercial organizations using the ATAM. For example, SEI staff members recently conducted a successful ATAM-based evaluation for the Joint National Test Facility (JNTF), Shriever Air Force Base, Colorado. The JNTF provides modeling and simulation support to military and defense industrial customers. The SEI applied the ATAM to Wargame 2000, its centerpiece simulation tool. According to JTNF personnel, the ATAM-based evaluation produced several architecture-related benefits; but more importantly, it empowered the participants. In particular, it enabled the project manager to identify and address a key design tradeoff that would not have surfaced until much later, when its impact would have been significant. In addition, several participants expressed “amazement” about how the process fostered open and frank communication among the government and support contractors.

In a non-defense government application, the SEI is helping to integrate the ATAM into the maintenance and evolution process for a NASA satellite data acquisition and processing system. NASA engineers are using the ATAM to develop a list of quality attribute requirements. Before the ATAM evaluation, the justifications for key architectural decisions were not well documented. Also, stakeholders (other than development staff) viewed all changes as equally plausible. After the ATAM, the architect, development personnel, and operations stakeholders reached consensus on priorities for quality attribute requirements. They identified and documented 50 key architectural decisions and more than 100 associated risks, sensitivity points, and tradeoffs. And they prioritized and compared the technical merits of more than 60 proposed changes.

In the commercial area, the SEI and one of its technology adoption partners, Robert Bosch Gmbh, recently evaluated the architecture of an embedded automotive system. During the evaluation, the ATAM team uncovered 40 risks, 3 sensitivity points, and 5 tradeoff points. Bosch managers responsible for the embedded automotive system were very satisfied with these results. A key manager remarked that “this is the first time that business and technical perspectives were discussed at the same time.” Knowing that a risk may jeopardize a business goal can motivate both managers and engineers. In this instance, the knowledge gave the architecture group a clear and focused agenda and the manager’s statement gave them the authority to act.

Transitioning ATAM

Currently, the SEI is looking for additional technology adoption partners to further the use of the ATAM. Under this arrangement, the SEI will work closely with the organization. SEI technical staff members typically perform an initial evaluation using the ATAM to demonstrate its potential. Next, the organization’s potential ATAM evaluators will take the ATAM evaluator training. The SEI will then perform a joint evaluation with the technology adoption partner. Finally, they will observe the partner independently performing an evaluation and provide additional recommendations, if necessary. The SEI is also planning to license third-party organizations to use the ATAM.

In addition, Paul Clements, Rick Kazman, and Mark Klein have authored Evaluating Software Architectures: Methods and Case Studies, which will be published in the fall of 2001 as part of the Addison-Wesley SEI Series in Software Engineering. The book is designed for anyone who is, or will be, involved in software architecture evaluation. While it is written from the evaluator's perspective, project managers, architects, and other stakeholders can gain valuable insights into the purpose and the process of architecture evaluation.

The ATAM has proven to be an effective means to evaluate the impact of architectural decisions on the achievement of quality attributes requirements. The ATAM has been "road-tested" over the last several years on many systems in different domains and is now ready for widespread adoption.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
Linda Northrop (lmn@sei.cmu.edu)

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

Overcoming the Challenges of COTS

Bob Lang

Few organizations today would consider building a system entirely from scratch. Use of commercial-off-the-shelf (COTS) products offers the promise of faster time to market and an opportunity to take advantage of commercial investments in technology to increase the functionality and capability of the system.

But the promise of COTS products is too often not realized in practice. Many organizations find that COTS-based systems are difficult and costly to build, support, and maintain. An important factor in this lack of success is that organizations building COTS-intensive solutions tend either to assume that COTS can simply be thrown together or they fall back on the traditional development skills and processes that they are familiar with—skills and processes that experience has shown not to work in the development of a COTS-based system.

The SEI is working to help organizations overcome the challenges of acquiring and fielding COTS-based systems through the Information Technology Solutions Evolution Process (ITSEP).

Background

Building systems from COTS products is an accelerating trend. A vibrant market delivers COTS software products that range from software development environments to operating systems, database management systems, middleware, and, increasingly, business and mission applications. Because COTS products are developed and refined in the competitive marketplace, it is assumed that they will have improved capability, reliability, and functionality compared to custom-built components. COTS products are also expected to integrate with one another, work in a wide range of environments, and support extensions and tailoring to local requirements.

However, reality is often very different. Experience has shown that successfully using COTS products requires a new way of doing business: new skills, knowledge, and abilities, changed roles and responsibilities, and different processes. But no known work to date provides organizations the level of detail, the tool sets, or the training needed to support this new way of doing business. “The changes extend beyond the traditional development effort,” says Lisa Brownsword of the SEI. “Implementing a COTS-intensive solution affects the way the organization will conduct business, so all stakeholders need to undergo a culture change, and these changes are just not happening.”

About ITSEP

ITSEP evolved from a need expressed by Col David Bentley, USAF and his MITRE support team for a process designed to meet the challenges of acquiring, developing, and fielding COTS-intensive solutions.

ITSEP is more than a way to select a specific product; rather, it provides a way to develop, field, and support a coherent, harmonious solution set composed of one or more products, any required custom code, and any changes required to end-user processes.

ITSEP relies heavily on ongoing work at the SEI—a framework for developing COTS-based systems—and on the Rational Unified Process (RUP) for its disciplined, risk-based spiral development approach.

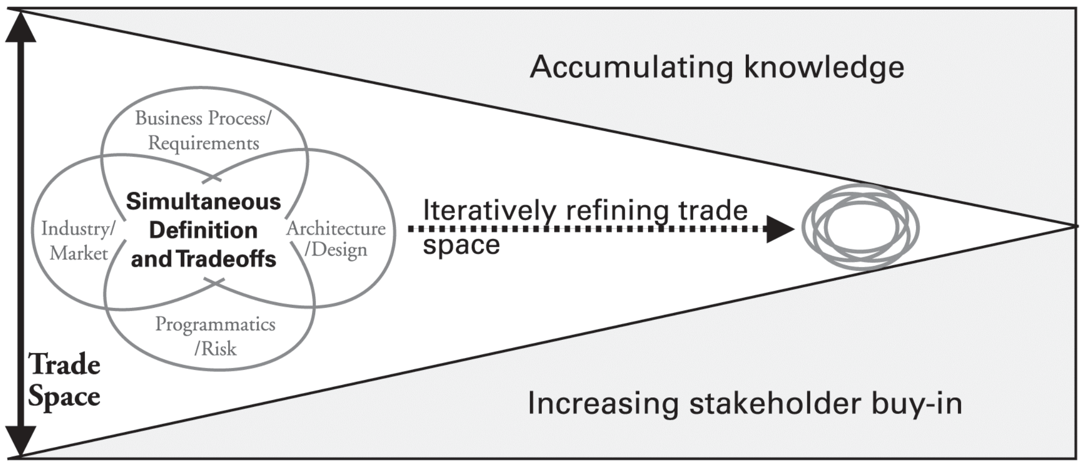
Accumulating knowledge
Business Process/ Requirements
Industry/ Market
Simultaneous Definition and Tradeoffs
Architecture /Design
Programmatics /Risk
Iteratively refining trade space
Trade Space
Increasing stakeholder buy-in

Figure 1: Balance Spheres of Influence

As shown in Figure 1, the key to ITSEP is the simultaneous definition and tradeoff of four spheres of influence; business processes and stakeholder needs, architecture and design, offerings from the commercial marketplace, and programmatics and risk.

This contrasts with the more traditional approach, which consists of defining the requirements, then formulating an architecture to meet those requirements, and then trying to fit products into that architecture. Many projects that have tried to use the traditional approach for COTS-based systems have failed. “So rather than trying to get the stakeholder needs fully documented as requirements up front,” says Brownsword, “we’ve found that to be successful, you need to keep the stakeholder needs as fluid as you can until you know what available COTS products can do.”

Environment to Support COTS-Intensive Solutions

ITSEP creates an environment that supports the iterative definition of the four spheres over time as decisions converge to systematically reduce the trade space. Initially the trade space is large: there is great flexibility for making tradeoffs between the stakeholder needs and business processes, the architecture and design, the offerings of the commercial marketplace, and programmatics and risk. As an improved understanding of the solution is gained across the stakeholder base, the trade space shrinks: the spheres increasingly overlap as fewer decisions remain in any single sphere that can be made without significant impact to the others.

This understanding includes increasingly detailed knowledge of

- the capabilities and limitations of candidate products, their vendors, and the market segment in which they operate
- the stakeholder needs for functional and nonfunctional performance of the system
- the implications of the COTS products on the end user’s business processes as well as the planning necessary to implement any needed business process changes
- the architectural alternatives and integration mechanisms that bind the COTS products together
- the cost, schedule and risk associated with implementing and fielding the solution

In ITSEP, the stakeholders make trades between what COTS products can deliver and how the end users will operate using those products.

Current Status

The ITSEP concepts have been used to implement a number of COTS-based systems. To facilitate institutionalization of these concepts across a broader base, ITSEP has been documented to provide detailed instructions for those who want to implement the process. The process designers are now looking for transition partners and opportunities to pilot ITSEP in both government and commercial settings.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
Ceci Albert (cca@sei.cmu.edu)
Lisa Brownsword (llb@sei.cmu.edu)

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

Using Quality Attribute Workshops to Evaluate Early-Stage Architecture Design Decisions

Mario Barbacci

The architectural decisions made in the beginning of the design process, by and large, determine the software's quality attributes in the end. These architectural design decisions are the hardest to change after the system has been implemented. Therefore, they are the most important to get right. To help architects and other stakeholders evaluate the implications of early-stage design decisions, the Software Engineering Institute has developed the Quality Attribute Workshop (QAW) method. This approach uses test cases to examine an architecture’s ability to achieve desired attributes. It represents a cost-effective and efficient means of exploring the impact of architectural decisions before they are made.

In our March 2000 column we presented our ideas for the QAW method and our early experiences from conducting one workshop. (See (http://interactive.sei.cmu.edu /news@sei/columns/the_architect/2000/spring/the_architect.htm?si) Since then we have modified the method based on our findings from several workshops. This column presents a more refined view of the QAW method.

Quality attributes are interdependent: performance affects modifiability, availability affects safety, security affects performance, and everything affects cost. Design evaluations can help architects explore the impact of decisions on the quality attributes of their software systems. Ideally, these assessments should take place as early in the architecture-development stage as possible. The SEI has developed the Quality Attribute Workshop (QAW) method, which uses test cases to assess an architecture for quality attributes early in the design process. (For a more complete definition of the QAW method, and how it differs from the Architecture Tradeoff Analysis Method, see Sidebar 2: “ATAM and QAW: How They Differ.”)

The QAW process can be applied before the architecture has been completely designed. It can be used on a wide range of architectural representations. It also requires relatively little time and effort on the part of QAW participants. Yet the QAW method can help designers ensure that the resulting architecture will deliver the quality attributes that the system and its stakeholders require.

The QAW Process

The QAW process is composed of four steps. As shown in Figure 1, these steps are: (1) scenario generation, where we generate, prioritize, and refine scenarios; (2) test case development; (3) test case analysis, in which test cases are analyzed against the architecture; and (4) results presentation, in which results are presented to other stakeholders.

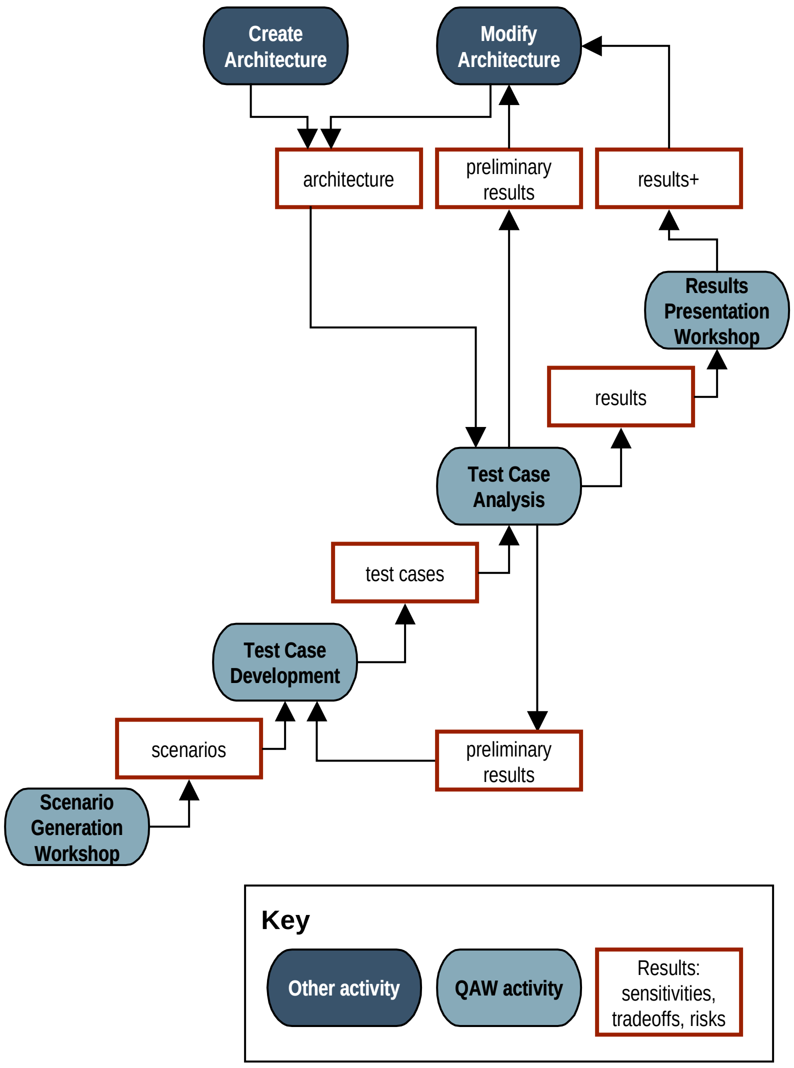

Figure 1: The QAW Process

To understand the power and the potential of the QAW, let's examine each step in detail.

Step 1: Scenario Generation Workshop

At the initial meeting, SEI facilitators and stakeholder representatives conduct a brainstorming session. Participants suggest scenarios or ask questions about the way in which the architecture will respond to various situations. They are encouraged to generate as many scenarios as possible in one to two hours. Usually, participants generate 30 to 40 scenarios.

Because only a small number of scenarios can be refined during a typical one-day workshop, the QAW participants must prioritize their choices by voting on the scenarios. The refining activity elicits details such as the expected operational consequences, the system assets involved, the end-users involved, the potential effects of the scenario on system operation, and the exceptional circumstances that may arise. See Sidebar 1 for an example of a scenario refinement.

Step 2: Test Case Development

Each refined scenario is transformed into a test case consisting of (1) a context section, (2) an issues and questions section, and (3) a utility tree. See Sidebar 1 for an example test case.

The context section describes the mission, the assets involved, the geographical region, the operational setting, and the players. It

SIDEBAR 1
A QAW Test Case

In the following example, a general scenario is turned into a test case. Note that while several quality attributes could be considered, this example only illustrates questions concerning two attributes, performance and availability. For a more detailed example analysis of this test case see http://www.sei.cmu.edu/publications/documents/00.reports/00tn010.html?si

In a recent activity conducted for a government agency, participants generated the scenario "Mars orbital communications-relay satellite fails." The QAW participants refined the scenario by explaining the circumstances associated with it:

- one of three aero-stationary satellites fails
- reports failure to Earth control element
- Mars surface elements and Mars satellites know that it failed
- service assessment done at control center (two days)
- traffic rerouting is to be performed
- network reconfiguration dictated by flight director, perhaps postponed to limit possibility of further failure
- multiple authorities in multiple organizations and control centers
- well defined decision-making process leading to mission director (final authority)
- multiple missions will be running simultaneously; coordination is complex

With the following additional details the scenario can be turned into a test case:

Context. Human and robotic missions are present on the Mars surface when the power amplifier fails on one of three stationary satellites. The primary communications payload is disabled for long-haul functions but the proximity link to other relay satellites and customers in orbit and on the surface is still working. Secondary telemetry and tele-command for spacecraft health is still working for direct-to-Earth with a low data rate. The remaining two satellites are fully functional. Communications with the crew has been interrupted. The crew is not in an emergency situation at the time of the failure, but reconnection is needed "promptly." The crew on the surface is concentrated in one area and the other missions in the Mars vicinity are in normal operations, are working on non-emergencies, or are performing mission-critical events. The communications network is well developed.

Stimulus. Detection of failure: Power amplifier failed, disabling the long-haul functions. The proximity link to other relay satellites and customers in orbit and on the surface is still working.

also describes the operation over some time period. For example, a general scenario may be “system responds to failure of a communication relay device.”

The test case for the failure scenario should describe

- the operation at the time of failure
- what happens when the system reacts to the failure
- degraded operation that occurs while repair is underway
- the process of restoring the system to normal operation
-

The issues section defines various architectural concerns associated with the context and proposes questions that connect these issues to quality attributes. For example, the issue may be: “How is failure detected?” The questions may be: “What

subsystem detects the failure? How long does it take to detect the failure? And, what happens during this interval?” In the failure context, it is relatively straightforward to discuss the following:

- performance issues, such as time to detect failure
- availability issues, such as degraded mode of services provided
- interoperability issues, such as how an alternative service might be introduced
- security issues, such as the impact on data integrity

Finally, the test case incorporates a utility tree. It graphically links quality attributes to specific attribute issues, and then to specific questions as illustrated in the example in Sidebar 1.

Quality Attribute Issues and Questions. The test case lists a number of questions to be answered by the analysis.

1. Issue: Mission safety requires consistent and frequent communications between the crew and Earth (P, A)
   a. Question: How long does it take to detect the failure?
   b. Question: How long does it take to reconfigure the system to minimize the time the crew is without communication?

2. Issue: System operation will be degraded (P, A)
   a. Question: Is there a way for the customer to simplify procedures in order to handle a larger number of missions with less trouble than they now have coordinating two missions?
   b. Question: What redundancy is required?
   c. Question: Is there a way to send information about the degraded satellite back to Earth for analysis?

3. Issue: System recovery (P, A)
   a. Question: Can the crew participate in the repair?
   b. Question: Is there any expectation for a human interface between Mars and Earth (e.g., with the crew in the space station)?
   c. Question: Can the customer participate in the notification (e.g., “Please send a message to the other satellite”)?

Utility Tree

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

Step 3: Test Case Analysis

The analysis should be done at the highest reasonable level of abstraction, but should include components that make sense under the test case circumstances. In the example mentioned above, the architecture team might respond in one of two ways:

1. The team could present a sequence diagram that includes an architectural component labeled “traffic manager” that diverts the message traffic, provides details on the level of degradation of the network traffic during the failure, and also includes a load-shedding component.

2. Alternatively, the team’s sequence diagram could show a “control center” that contains a “traffic manager” and a “load shedder” that will take the appropriate actions.

Obviously the first case includes sufficient detail to evaluate the problem. The second case is much more of a “trust me” statement; one that causes problems in the future. As mentioned earlier, this is an iterative process. The team may refine the architecture or it may continue analyzing scenarios.

Once the team members are satisfied with the results, they should document the answers, then proceed to Step 4.

Step 4: Results Presentation Workshop

This workshop presents the test case analyses. It is an opportunity for the architecture team members to demonstrate that (1) they completely understand the test cases, (2) their architecture is able to handle these cases correctly, and (3) they have the competence to continue analyzing important test cases as part of an architecture development effort.

Experience

The SEI has performed a number of quality attribute workshops. Among the lessons learned were:

- The first QAW workshop (which generates, prioritizes, and refines scenarios) is a useful communications forum. Often stakeholders are unfamiliar with what other stakeholders are doing or are unaware of considerations brought up by those responsible for maintenance, operations, or acquisition.
- The approach depends on getting a number of important stakeholders at the workshop.
- In most cases, the workshops discovered undocumented requirements. They were revealed when workshop scenarios were checked against system requirements.
- The scenarios helped ensure that projected deployment of assets and capabilities matched the scenarios and test cases.

In conclusion, the QAW can help architects explore early-stage architectural design decisions. At this point, SEI facilitators are continuing to hold QAW workshops with additional sponsors, in different application domains, and at different levels of detail. The approach looks promising; the concept of testing out flaws in the architecture should reduce rework in building the system and improve its ability to deliver required quality attributes.

SIDEBAR 2
ATAM and QAW: How They Differ

For the past two years the SEI has been developing the Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}) for evaluating system architectures based on a set of quality attributes, such as modifiability, security, performance, and reliability. The ATAM process typically takes three days and the involvement of 10-20 people, including evaluators, architects, and other system stakeholders. The effectiveness of ATAM depends on having a concrete, well-defined architecture to be analyzed. However, some ATAM benefits can be achieved even if an architecture is not fully defined. Under some circumstances, an organization might wish to identify potential architecture risks while developing a system's architecture.

For this reason, the SEI has developed the Quality Attribute Workshop (QAW) method, in which architects, developers, users, maintainers, and other system stakeholders, such as people involved in installation, deployment, logistics, planning, and acquisition, carry out several ATAM steps, but focus on system requirements and quality attributes, rather than on the architecture. The objective of the workshop is to identify sensitivities, tradeoffs, and risks and use these as early warnings to the architecture developers.

The QAW and the ATAM approach differ in substantial ways. In ATAM, the scenario generation and analysis happen in real-time, during stakeholder meetings, typically one or two days long. In QAW, the scenario generation takes place in a similar venue but the analysis is carried out off-line and is presented to the stakeholders weeks or even months later. In the ATAM case the developers already have an architecture and have made a number of architectural decisions, thus they are able to conduct the analysis of a scenario in a matter of minutes. In the QAW case, there might not be an architecture, the architecture team might still be pondering a number of decisions, or the team members might not be aware that they might have to worry about some risk or another. Thus, in QAW the scenarios, plus any additional refinements obtained from the stakeholders during the scenario-generation meeting, are converted into "test cases" with additional details and specific questions to be answered by the analysts. The process provides the architecture team with time to conduct the analysis, to make changes and try alternative approaches, and to document their decisions and responses to the test case questions without the time pressure of the ATAM process.

About the Author

Mario Barbacci is a senior member of the technical staff at the SEI. 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). Before coming to the SEI, he was a member of the faculty in the School of Computer Science at Carnegie Mellon. Barbacci is a fellow of the Institute of Electrical and Electronic Engineers (IEEE), 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 vice president for technical activities of the IEEE Computer Society and chair of the Joint IEEE Computer Society/ACM Steering Committee for the Establishment of Software Engineering as a Profession. He was the 1996 president of the IEEE Computer Society. He was the 1998-1999 IEEE Division V Director.

Barbacci is the recipient of several IEEE Computer Society Outstanding Contribution Certificates, the ACM Recognition of Service Award, and the IFIP Silver Core Award. Barbacci 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.

Building Systems from Commercial Components Using Model Problems

Robert C. Seacord

Model problems are focused experimental prototypes that reveal technology/product capabilities, benefits, and limitations in well-bounded ways. They are useful for adding rigor to component-based system design as well as reforming the means by which component evaluation and selection is performed.

In 1982 my employer presented me with a new toy, an IBM 5051 computer, which became better known as the "IBM PC." I was also provided with a list of all the software programs available for the platform—on a single sheet of paper.

For some time, developing software for this platform (and others) meant developing custom software. Commercial components were simply not available. While this situation was not particularly conducive to productivity (it seemed the first step of each project in those days was to create a systems services layer), it was a comfortable situation for developers. Once you had mastered the DOS and BIOS interfaces, and had a comfortable understanding of 8086/8088 assembler (or a higher-level programming language), there were not many surprises.

Today, developers do not have this comfort zone when they build systems from commercial components. The dynamics of the component market practically guarantee that engineers are, at least in part, unfamiliar with the capabilities and limitations of individual software components and their interactions when combined into a component ensemble.

Fundamental Ideas

Prototyping is a fundamental technique of software engineering, as it has been incorporated into Boehm's Spiral Model [Boehm 88] and institutionalized in various industrial-strength software processes [Cusumano 95, Jacobson 99]. In spiral process models, a project is conceived as a series of iterations over a prescribed development process. Usually, each iteration (except the last, of course) produces a prototype that is further refined by succeeding iterations.

Prototypes may be developed for the customer of the system or they may be built for the designers of the system. Ultimately, all prototyping is motivated by the desire to reduce risk. For example, the development team might build a prototype to reduce the risk that a customer will be unsatisfied with the interface or planned functionality of a system.

There are three motivations for designers to build prototypes. The first is for the designer to develop a basic level of competence in a component, or in an ensemble of components. Typically, this is best accomplished through unstructured "play" with the component. The second motivation is to answer a specific design question: for example, can I use my Web browser to launch an external application? The use of *model problems*—focused experimental prototypes that reveal technology/product capabilities, benefits, and limitations in well-bounded ways—is well suited to this purpose. The third motivation for building prototypes for the designer is persuasion—after all, good ideas are irrelevant if they are not adopted. Model problems are also an effective mechanism for advocating a particular design approach.

Model Problems

The use of model problems in evaluating technology and product capabilities is fully explored in our new book, *Building Systems from Commercial Components*, published by Addison-Wesley [Wallnau 01]. In the remainder of this article, I provide an introduction to model problems including the terminology and an overview of the process.

A model problem is actually a description of the *design context*. The design context defines the constraints on the implementation. For example, if the software under development must provide a Web-based interface for both Netscape Navigator and Microsoft Internet Explorer, this is part of the design context that constrains the solution space.

A prototype situated in a specific design context is called a *model solution*. A model problem may have any number of model solutions. The number of model solutions developed depends on the severity of the risk inherent in the design context, and the relative success of the model solutions in addressing this risk.

Model problems are normally used by design teams. Optimally, the design team consists of an architect who is the technical lead on the project and makes the principal design decisions, as well as a number of designers/engineers who may be tasked by the architect to execute the model problem.

The overall process consists of the following steps that can be executed in sequence:

1. The architect and engineer identify a *design question*. The design question initiates the model problem. It refers to an unknown that is expressed as a hypothesis.
2. The architect and engineer define the *a priori evaluation criteria*. These criteria describe how the model solution will be shown to support or contradict the hypothesis.
3. The architect and engineer define the *implementation constraints*. The implementation constraints specify the fixed (i.e., inflexible) part of the design context that governs the implementation of the model solution. These constraints might include such things as platform requirements, component versions, and business rules.

4. The engineer produces a *model solution* situated in the design context. The model solution is a minimal spanning application that uses only those features of a component (or components) that are necessary to support or contradict the hypothesis.
5. The engineer identifies *a posteriori evaluation criteria*. A posteriori evaluation criteria include the a priori criteria plus criteria that are discovered as a byproduct of implementing the model solution.
6. Finally, the architect performs an *evaluation* of the model solution against the a posteriori criteria. The evaluation may result in the design solution being rejected or adopted, but often leads to the generation of new design questions that must be resolved in similar fashion.

Summary

Model problems have been successfully applied by the SEI in collaboration with organizations over the past five years. These can be applied with varying degrees of formality, but successful model problems always consist of the steps outlined in this article (that is a design question, *a priori* evaluation criteria, implementation constraints, one or more model solutions, *a posteriori* evaluation criteria, and evaluation).

Model problems are extensively covered in *Building Systems from Commercial Components*, including an extensive case study describing multiple applications of this technique in the Web-based system and security domains. Generally speaking, model problems have proven to be a successful technique for adding rigor to component-based system design as well as reforming the means by which component evaluation and selection is performed.

References

[Boehm 88] Barry Boehm. “A Spiral Model of Software Development and Enhancement.” *Computer*, May 1988: pp. 61-72.

[Cusumano 95] Michael A. Cusumano and Richard W. Selby. *Microsoft Secrets*. New York: The Free Press, 1995.

[Jacobson 99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Reading, MA: Addison-Wesley, 1999.

[Wallnau 01] Kurt Wallnau, Scott Hissam, Robert Seacord. *Building Systems from Commercial Components*. Reading, MA: Addison-Wesley, 2001.

About the Author

Robert C. Seacord is a senior member of the technical staff at the SEI and an eclectic technologist. He is coauthor of the book *Building Systems from Commercial Components* as well as more than 30 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.

CERT® System and Network Security Practices

Julia Allen

Systems, networks, and sensitive information can be compromised by malicious and inadvertent actions despite an administrator's best efforts. Even when administrators know what to do, they often don't have the time to do it; operational day-to-day concerns and keeping systems functioning take priority over securing those systems. Administrators need easy-to-access, easy-to-understand, easy-to-implement security practices. The CERT® system and network security practices are intended to meet those needs.

Editor's Note: This paper was presented at the NCISSE 2001: Fifth National Colloquium for Information Systems Security Education, held at George Mason University in Fairfax, VA, May 22-24, 2001. It will be published in the NCISSE proceedings. A longer version of this article is available online at http://www.cert.org/archive/pdf/NCISSE_practices.pdf.

The Problem as Viewed by Administrators

Administrators choose how to protect assets, but when managers are unable to identify the most critical assets and the nature of the threats against them (as part of a business strategy for managing information security risk), then the protections an administrator offers are likely to be arbitrary at best. Unfortunately, managers often fail to understand that securing assets is an ongoing process and not just a one-time fix. As a result, they do not consider this factor when allocating administrator time and resources. Even if an organization decides to outsource security services, it will probably continue to be responsible for the establishment and maintenance of secure configurations and the secure operations of critical assets.

Most system and network administrators only have their experience and well-meaning advice from peers to guide them in deciding how to protect and secure systems. They do not consult a published set of procedures that serve as de facto standards generally accepted by the administrator community because no such standards exist. Administrators are sorely in need of documented practices that are easy to access, understand, and implement. The practices summarized in this paper are intended to meet this need. They are fully described in my book *The CERT Guide to System and Network Security Practices*, recently published by Addison-Wesley (see http://cseng.aw.com/book/0,,020173723X,00.html).

® CERT and CERT Coordination Center are registered in the U.S. Patent and Trademark Office.

We recognize that it may not be practical to implement all steps within a given practice or even all practices. Business objectives, priorities, and an organization's ability to manage and tolerate risk dictate where information technology (IT) resources are expended and determine the tradeoffs among security and operational capability. However, by adopting these practices, an administrator can protect against today's threats, mitigate future threats, and improve the overall security of an organization's networked systems.

CERT Security Practices Structure

The CERT System and Network Security practices address 75 to 80 percent of the problems that are reported to the CERT/CC.[1] The practices describe the steps necessary to protect systems and networks from malicious and inadvertent compromise. Each practice consists of an introduction and a series of practical steps presented in the order of recommended implementation. There is also a section describing policy considerations (found in the practices on the CERT Web site at http://www.cert.org/security-improvement/index.html?si) that complements the steps and helps ensure that they will be deployed effectively.

All practices assume the existence of

- business objectives and goals from which security requirements derive. These may require periodically conducting an information-security risk analysis and assessment (such as the Organizationally Critical Threat, Asset, and Vulnerability Evaluation[SM]; see http://www.cert.org/octave?si) to help set priorities and formulate protection strategies.
- organization-level and site-level security policies that can be traced to the above business objectives, goals, and security requirements. If these security policies do not currently exist, developing them an essential task. Charles Cresson Wood, among others, has prepared an extensive reference guide describing all elements of a security policy along with sample policy language [Wood 2000].

Practices were written without reference to any one operating system or version. This makes the practice steps specific but still broadly applicable and ensures the practices will be useful and stay relevant longer than the most current version of an operating system. Examples of practice implementations specific to operating systems are available at the CERT Web site (http://www.cert.org/security-improvement).

---

[1] As determined by CERT vulnerability analysis and fourth quarter, 2000 incident analysis. In addition, the CERT security practices are periodically analyzed against top threat lists published by other organizations and consistently provide solutions for at least 80% of such threats.

[SM] Organizationally Critical Threat, Asset, and Vulnerability Evaluation and OCTAVE are service marks of Carnegie Mellon University.

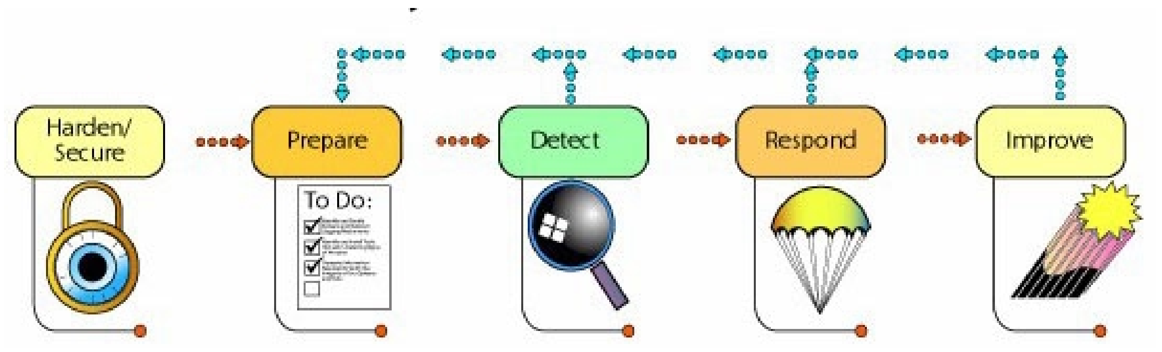

Figure 1: Steps for Securing Information Assets

Figure 1 above serves as a top-level depiction of how to secure and protect information assets. It includes steps to harden/secure, prepare, detect, respond, and improve.

The recommended practices to harden and secure systems form a strong foundation by securely configuring information assets (such as networks, systems, and critical data) and by establishing secure access to these assets. If this is done correctly *and maintained*, many of the common vulnerabilities used by intruders are eliminated. Following these practices can greatly reduce the success of many common, recurring attacks. Prepare, detect, respond, and improve practices assume that harden/secure practices have been implemented and provide further guidance about what to do when something suspicious, unexpected, or unusual happens.

Step One: Harden/Secure

Systems shipped by vendors are very usable but, unfortunately, often contain many weaknesses when viewed from a security perspective.[1] Vendors seek to sell systems that are ready to be installed and used by their customers. The systems come with most, if not all, services enabled by default. Vendors apparently want to minimize telephone calls to their support organizations and generally adopt a “one-size-fits-all” philosophy in relation to the systems they distribute. Therefore, an administrator must first redefine the system configuration to match the organization’s security requirements and policy for that system.

1 Refer to the CERT vulnerability database (http://www.kb.cert.org/vuls), and the Common Vulnerabilities and Exposures (CVE) site (http://cve.mitre.org) for detailed vulnerability information.

This step will yield a hardened (secure) system configuration and an operational environment that protects against known attacks for which there are defined mitigation strategies. To complete this step, follow the instructions below in the order listed:

1. Install only the minimum essential operating system configuration—that is, only those packages containing files and directories that are needed to operate the computer.
2. Install patches to correct known deficiencies and vulnerabilities. Installing patches should be considered an essential part of installing the operating system but is usually conducted as a separate step.
3. Install the most secure and up-to-date versions of system applications. It is essential that all installations be performed before the next step (removing privileges) since any installation performed after privileges are removed can restore undesired access privileges.
4. Remove all privilege and access and then grant (add back in) privilege and access only as needed, following the principle "deny first, then allow."
5. Enable as much system logging as possible to have access to detailed information (needed for in-depth analysis of an intrusion).

Practices for hardening and securing general-purpose network servers (NS) and user workstations (UW) are listed in Table 1 and are fully described on the CERT Web site [Allen 2000a, Simmel 99].

Plan	Address security issues in your computer deployment plan (NS, UW). Address security requirements when selecting servers (NS).
Configure	Keep operating systems and applications software up to date (NS, UW). Stick to essentials on the server host system (NS). Stick to essentials on the workstation host system (UW). Configure network service clients to enhance security (UW). Configure computers for user authentication (NS, UW). Configure operating systems with appropriate object, device, and file access controls (NS, UW). Configure computers for file backups (NS, UW). Use a tested model configuration and a secure replication procedure (UW).
Maintain	Protect computers from viruses and similar programmed threats (NS, UW). Configure computers for secure remote administration (NS, UW). Allow only appropriate physical access to computers (NS, UW).
Improve User Awareness	Develop and roll out an acceptable use policy for workstations (UW).

Table 1: Practices for Hardening and Securing Network Servers (NS=general-purpose network servers; UW=user workstations)

Additional hardening details can be found in the CERT document, *Installing and securing Solaris 2.6 servers.*[1]

Practices addressing more specific details for securing public Web servers (such as Web server placement, security implications of external programs, and using encryption) are listed in Table 2 and are documented on the CERT Web site [Kossakowski 2000].

Configure	Isolate the Web server. Configure the Web server with appropriate object, device, and file access controls. Identify and enable Web-server specific logging mechanisms. Consider security implications for programs, scripts, and plug-ins. Configure the Web server to minimize the functionality of programs, scripts, and plug-ins. Configure the Web server to use authentication and encryption technologies.
Maintain	Maintain the authoritative copy of your Web site content on a secure host.

Table 2: Practices for Securing Web Servers

Practices that provide guidance on deploying firewall systems (such as firewall architecture and design, packet filtering, alert mechanisms, and phasing new firewalls into operation) are listed in Table 3 and are presented on the CERT Web site [Fithen 99]. Public Web server and firewall practices assume that you have first configured a secure general-purpose server and have then built on it.

Prepare	Design the firewall system.
Configure	Acquire firewall hardware and software. Acquire firewall training, documentation, and support. Install firewall hardware and software. Configure IP routing. Configure firewall packet filtering. Configure firewall logging and alert mechanisms.
Test	Test the firewall system.
Deploy	Install the firewall system. Phase the firewall system into operation.

Table 3: Practices for Deploying Firewall Systems

[1] Available at http://www.cert.org/security-improvement under UNIX implementations.

Step Two: Prepare

The philosophy of the preparation step hinges on the recognition that despite steps taken to harden and secure a system, there exist vulnerabilities yet to be identified. Consequently, an administrator must be able to recognize when these vulnerabilities are being exploited. To support such recognition, it is vitally important to characterize a system so that an administrator can understand how it works in a production setting. Through a thorough examination and recording of a known baseline state and expected changes at the network, system (including kernel), process, user, file, directory, and hardware levels, the administrator learns the expected behavior of an information asset. In addition, the administrator and his or her manager must develop policies and procedures to identify, install, and understand tools for detecting and responding to intrusions well before such policies, procedures, and tools need to be invoked.

One way to think about the distinction between the hardening and securing step and the characterization part of preparing is that hardening attempts to solve *known* problems by applying known solutions, whereas characterization helps administrators identify *new* problems and formulate new solutions. In the case of characterization, the problems are identified through anomaly-based detection techniques—that is, departures from normal behavior—so that new solutions can be formulated and applied.

Practices for characterizing information assets, preparing to detect signs of intrusion, and preparing to respond to intrusions are listed in Table 4 and are fully described on the CERT Web site [Allen 2000b, Kossakowski 99].

Define level of preparedness	Establish policies and procedures.
Implement preparation steps	Identify characterization and other data for detecting signs of suspicious behavior. Manage logging and other data collection mechanisms. Select, install, and understand tools for response.

Table 4: Practices for Characterizing Assets and Preparing for Intrusions

Step Three: Detect

This step occurs during the monitoring of transactions performed by some asset (such as looking at the logs produced by a firewall system or a public Web server). The administrator notices some unusual, unexpected, or suspicious behavior, learns something new about the asset's characteristics, or receives information from an external source (a user report, a call from another organization, a security advisory or bulletin). These indicate either that something should be analyzed further or that something on the system has changed or should change (a new patch should be applied, a new tool version should be installed, etc). Analysis includes investigating unexpected or suspicious behavior that may be the result of an intrusion and drawing some initial conclusions, which are further refined during the **Respond** step. Possible changes include a number of improvement actions (see **Improve,** below) such as

- installing a patch (re-hardening)
- updating the configuration of a logging, data collection, or alert mechanism
- updating a characterization baseline to add unexpected but now acceptable behavior or remove no longer acceptable behavior
- installing a new tool

Practices are listed in Table 5 for detecting signs of intrusion in detection tools, networks, systems (including processes and user behavior), network and system performance, files and directories, hardware, and access to physical resources. These practices are fully described on the CERT Web site [Allen 2000b].

Integrity of intrusion detection software	Ensure that the software used to examine systems has not been compromised.
Behavior of networks and systems	Monitor and inspect network activities. Monitor and inspect system activities. Inspect files and directories for unexpected changes.
Physical forms of intrusion	Investigate unauthorized hardware attached to the network. Look for signs of unauthorized access to physical resources.
Follow through	Review reports of suspicious system and network behavior and events. Take appropriate actions.

Table 5: Practices for Detecting Signs of Intrusion

Step Four: Respond

In this step, an administrator further analyzes the damage caused by an intrusion (including the scope and effects of the damage), contains these effects as far as possible, works to eliminate future intruder access, and returns information assets to a known, operational state. It may be possible to do this step while continuing analysis.

Other parties that may be affected are notified, and evidence is collected and protected in case it is needed for legal proceedings against the intruder. Respond practices are listed in Table 6 and are described on the CERT Web site [Kossakowski 2000].

Handle	Analyze all available information. Communicate with relevant parties. Collect and protect information. Contain an intrusion. Eliminate all means of intruder access. Return systems to normal operation.
Improve	Implement lessons learned.

Table 6: Practices for Responding to Intrusions

Step Five: Improve

Improvement actions typically occur following a detection or response activity. In addition to those noted under **detect,** above, improvement actions may include

- further communicating with affected parties
- holding a post-mortem meeting to identify lessons learned
- updating policies and procedures
- updating tool configurations and selecting new tools
- collecting measures of resources required to deal with the intrusion and other security business-case information

Improvement actions may cause you to revisit harden/secure, prepare, and detect practices.

References

[Allen 2000a] Allen, Julia. Kossakowski, Klaus-Peter. *Securing Network Servers* (CMU/SEI-SIM-010). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000. [Online] Available: http://www.cert.org/security-improvement/modules/m10.html.

[Allen 2000b] Allen, Julia. Stoner, Ed. *Detecting Signs of Intrusion* (CMU/SEI-SIM-009). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000. [Online] Available: http://www.cert.org/security-improvement/modules/m09.html.

[Fithen 99] Fithen, William, et al. *Deploying Firewalls*. (CMU/SEI-SIM-008). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999. [Online] Available: http://www.cert.org/security-improvement/modules/m08.html.

[Kossakowski 99]	Kossakowski, Klaus-Peter, et al. *Responding to Intrusions* (CMU/SEI-SIM-006). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999. [Online] Available: http://www.cert.org/security-improvement/modules/m06.html.
[Kossakowski 2000]	Kossakowski, Klaus-Peter. Allen, Julia. *Securing Public Web Servers* (CMU/SEI-SIM-011). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 2000. [Online] Available: http://www.cert.org/security-improvement/modules/m11.html.
[Simmel 99]	Simmel, Derek, et al. *Securing Desktop Workstations* (CMU/SEI-SIM-004). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1999. [Online] Available: http://www.cert.org/security-improvement/modules/m04.html.
[Wood 2000]	Wood, Charles Cresson. *Information Security Policies Made Easy Version 7*. Baseline Software, Inc., 2000.

About the Author

Julia Allen is a senior member of the technical staff within the Networked Systems Survivability Program at the Software Engineering Institute. The CERT® Coordination Center is also a part of this program. Allen is engaged in developing security-improvement practices for network-based systems. Prior to this technical assignment, Allen served as acting director of the SEI for an interim period of six months as well as deputy director/chief operating officer for three years. She started the Industry Customer Sector at the SEI in 1992.

Allen has more than 25 years of managerial and technical experience in software engineering. Before joining the SEI, she held the position of vice president at Science Applications International Corporation (SAIC), where she was responsible for starting a new software division specializing in embedded systems software for government customers. She was at SAIC for eight years. Allen also spent 10 years at TRW in Redondo Beach, CA., where her work included a range of assignments from integration, test, and field site support to managing major software-development programs.

She received a BS in computer science from the University of Michigan, an MS in electrical engineering from the University of Southern California, and an executive business certificate from the University of California at Los Angeles (UCLA). Her professional affiliations include ACM and IEEE Computer Society.

Allen’s publications include Security for Information Technology Service Contracts (CMU/SEI-SIM-003, 1998); Responding to Intrusions (CMU/SEI-SIM-006, 1999), Deploying Firewalls (CMU/SEI-SIM-008, 1999), Detecting Signs of Intrusion (CMU/SEI-SIM-009, 2000), Securing Network Servers (CMU/SEI-SIM-010, 2000), Securing Public Web Servers (CMU/SEI-SIM-011, 2000); six reports within the Security Improvement module series; State of the Practice in Intrusion Detection Technologies (CMU/SEI-99-TR-028); *The CERT Guide to System and Network Security Practices* (Addison-Wesley, June 2001); and various presentations and papers on the CERT security practices, intrusion detection systems, and the SEI's strategic plan and technical program.

The Future of Software Engineering: II

Watts S. Humphrey

This is the second of several columns on the future of software engineering. The first column focused on trends in application programming, particularly related to quality. This column reviews data on programmer staffing and then covers application-programming skills. Future columns deal with trends in systems programming and the implications of these trends for software engineering and software engineers.

In my previous column, I started a discussion of the future of software engineering and reviewed the trends in application programming. In this column, I consider the growing demand for people to write application programs. I also explore the implications of the current trends in application programming. In the next few columns, I will examine the trends in systems programming and comment on the implications of these trends for software engineering and software engineers. While the positions I take and the opinions I express are likely to be controversial, my intent is to stir up debate and hopefully to shed some light on what I believe are important issues. Also, as is true in all of these columns, the opinions are entirely my own.

Some Facts

The demand for software engineers is at an all-time high, and it continues to increase. Based on recent census data, there were 568,000 software professionals in the U.S. in 1996. In 2007, there are projected to be 697,000 [Clark 00]. Since 177,000 are also projected to leave the field during this time, this implies a ten-year need for more than 300,000 new programmers. That is a 50% gross addition to the current programming population.

The Census Bureau estimate of 568,000 programmers seems low to me, and I suspect this is because of the criteria used to determine who was counted as a programmer. Howard Rubin quotes a number of 1.9 million programmers as the current U.S. programming population [Rubin 99]. I have also seen data showing that the number of programmers in the U.S. doubled from 1986 to 1996. While good data are sparse for such an important field, the demand for programmers has clearly increased in the past ten years, and it is likely to continue increasing in the future.

If you consider that most professionals in most fields of engineering and science must now write at least some software to do their jobs, the number of people who write, modify, fix, and support software must be very large. If the growth trends implied by the census data apply to the entire population of casual and full-time programmers, the demand for new programmers in the next ten years is likely to run into the millions.

Future Needs

Judging by past trends, it is clear that just about every industrial organization will need more people with application programming skills and that most programming groups will be seriously understaffed. Since many software groups are already understaffed, and the current university graduation rate of software professionals is only about 35,000 a year, we have a problem [U.S. 00]. In general terms, there are only two ways to address the application-programming problem.

1. Somehow increase the supply of new programmers.
2. Figure out how to write more programs without using more programmers.

Since it takes a long time to increase the graduation rate of software professionals, the principal approach to the first alternative must be to do more of what we are doing today—that is, to move more software offshore and to bring more software-skilled immigrants into the U.S.

While many organizations are establishing software laboratories in other countries, particularly India, this is a limited solution. The principal need is for skilled software professionals who understand the needs of businesses and can translate these needs into working applications. There is no question that the coding and testing work could be sent offshore, but that would require good designs or, at least, clear and precise requirements. Since producing the requirements and design is the bulk of the software job, going offshore can only be a small part of the solution.

Obtaining more software-skilled immigrants is an attractive alternative, particularly because India alone graduates about 100,000 English-speaking software professionals a year. However, the U.S. has tight visa restrictions, and many other groups also have claims on the available slots. Also, since the demand for software skills is increasing rapidly in India, and since many Indian professionals can now find attractive opportunities at home, the available numbers of Indian immigrants will likely be limited in the future.

The Automobile Industry Analogy

To examine the alternative of writing more programs without adding more programmers, consider the automobile industry. Back before Henry Ford, only the wealthy could afford cars. Then Henry Ford made the automobile affordable for ordinary folks. Once the manufacturers started catering to the needs of the masses, the automobile industry changed rapidly.

Many innovations were required before people could feel comfortable driving without a chauffeur. They needed the closed automobile body, automatic starters, heaters, clutches, transmissions, and a host of other progressively more automatic and convenient features. This combination of innovations made operating an automobile simple and easy for almost anyone.

With the aid of these innovations, people could learn to drive without chauffeurs. When all this happened, the chauffeur business went into a tailspin. Soon, as the comfort, convenience, and reliability of cars increased, driving an automobile was no longer a specialty; it became a general skill required of just about everyone. Today, most people learn to drive an automobile before they get out of high school. While there are still professional drivers, the vast majority of driving is now done by the general public.

The Computer Field Today

Today, the computer field is much like the early days of the automobile industry. Many professionals have learned to use computing systems, but few are willing to rely on them for critical work, at least not without expert help and support. In the computer field, the chauffeur equivalents are with us in the guise of the experts who develop applications, install and tailor operating systems, and help us recover from frequent system crashes and failures. Even on the Internet, our systems today often exhibit strange behavior and present us with cryptic messages. While these systems are far easier to use than before, they are not yet usable by the general public.

For computing systems to be widely used, we need systems that work consistently and are problem free. We also need support systems that serve the same functions as automobile starters and automatic transmissions. Then professionals in most fields will be able to automate their own applications without needing skilled programmers to handle the arcane system details.

Another prerequisite to the widespread use of computing systems is that the professionals in most fields be able to produce high-quality application programs with little or no professional help. The real breakthrough will come when it is easier to learn to write good software than it is to learn about most business or scientific applications. Then, instead of requiring that skilled software people learn about each application area, it will be more economical and efficient to have the application experts learn to develop their own software. At that point, software engineering will become a general skill much like driving, mathematics, or writing, and every professional will be able to use computing systems to meet the vast bulk of his or her application needs.

Growing System Size and Complexity

While such a change will be an enormous help, it will not address all aspects of application programming. To see why, consider the trends in the size and complexity of application programs. If history is any guide, future application programs will be vastly larger and more complex than they are today. This means that the development of such systems will change in a number of important ways.

As I wrote in the prior column, the first and possibly most important change is in quality. Those who need software simply will be unable to use programs to conduct their businesses unless they are of substantially higher quality than they are today. The second trend is equally significant: the current cottage-industry approach to developing application programs must give way to a more professional and well-managed discipline. This is not just because of the increasing size of the programs and their more demanding quality specifications, but also because the business of producing such programs will grow beyond the capability of most people to master quickly.

In other words, the day has largely passed when we could hire somebody who was reasonably familiar with the programming language of choice and expect him or her to rapidly become productive at developing application programs. As application programs become larger and more sophisticated, the required application knowledge and experience will increase as well. Soon, the cost and time required to build this application knowledge will be prohibitive. Therefore, a host of new methods must be developed to make application programming more economical and far less time consuming than it is today.

Reuse

My argument to this point has concerned getting more people to write programs. However, there is another alternative: finding ways to produce more applications with fewer people. One proposed solution to this challenge is through reuse. While this seems like an attractive possibility, recent history has not been encouraging. In fact, history indicates that reuse technology will be largely confined to building progressively larger libraries of language and system functions. Unfortunately, this added language complexity will cause other problems. This is not because reuse is unattractive; it is just at too low a level to address the application needs of most users.

The software community has been adding functional capability to programming languages for the 47 years since I wrote my first program. This approach has not solved the programming problems of the past, nor is it likely to solve those of the future. The principal reason is that by adding more microscopic functions to our languages, we merely restate the application development problem in slightly richer terms.

For example, when I wrote my first program we had to control the starting and stopping of the I/O devices and the transfer of each character. Now such functions are handled automatically for us, but we are faced instead with much more sophisticated languages. Instead of a simple language you could summarize on a single sheet of paper, we now need entire textbooks.

Granted, increased language richness reduces the detailed system knowledge required to manage the computer’s functions, but it still leaves us with the overall design problem, as well as the problem of determining what the design is supposed to do for the user. Then, the application programmer has the final challenge of translating the design into a functioning and reliable program.

This leads to the problem that will force us out of the cottage-industry approach to programming. That is the simple impossibility of quickly becoming fluent in all the languages and functions needed to produce the complex application systems of the future. While reuse in traditional terms may be helpful for the professional programming population, it is directly counter to the need to make our technology more accessible to people who are not full-time programming professionals.

Packaged Applications

To handle the volume needs of many users, companies are starting to market packaged applications much like those offered by SAP and Oracle. That is, they produce essentially prepackaged application systems that can be configured in prescribed ways. Rather than custom-designing each application, this industry will increasingly develop families of tailorable application systems. The users will then find the available system that comes closest to meeting their requirements and use its customization capabilities to tailor the system to their business needs.

To make these systems easily tailorable by their customers, companies will design their systems with limited, but generic, capabilities. Then, in addition to tailoring the system, the users must also adjust their business procedures to fit the available facilities of the system. As the experiences of SAP and others have demonstrated, this approach is not trouble free, but it can provide users with large and sophisticated application systems at much lower cost than a full custom-application development.

Judging by the growth of SAP, Oracle, and others, this has been an attractive strategy. Rather than developing applications to meet an unlimited range of possible user needs, users will increasingly adapt their business operations to fit the functions of the available application systems. While this represents a form of reuse, it is at a much higher level than the approaches generally proposed, and it generally requires a thoughtfully architected family of application products or product lines. Just as with the transportation, housing, and clothing industries, for example, once people see the enormous cost of customized products, they usually settle for what they can find on the rack.

Application Categories

Application development work in the future will likely involve three categories of work:

1. developing prepackaged applications that users can tailor to their needs

2. tailoring business systems to use prepackaged application systems
3. developing and supporting unique applications that cannot be created with prepackaged software

The programmers needed for the first category will be professionals much like those needed for developing systems programs, but they will generally have considerable application knowledge. I will write more about this category in later columns.

For category two, we will probably see a substantial growth in the volume of application customization. The people doing this work will be more like business consultants than programmers, and many will not even know how to design and develop programs. These people will be thoroughly trained in the packages that they are customizing and helping to install.

The reason for the third category is that, even though the prepackaged application strategy will likely handle most bread-and-butter applications, it will not handle those applications needed to support new and innovative business activities. Since these applications will not have been used before, nobody will know how to produce prepackaged solutions. As a result, there will be a volume of applications that cannot be solved by prepackaged solutions. Therefore, even with a wide variety of available prepackaged applications, the need for customized application development will not disappear.

Custom Application Programming

Custom application work must be handled by people who know how to write programs and who also understand the application specialty. These people must be experts on a wide variety of specialties, and must be able to write high-quality programs. For these people, we must develop suitable methods and training—to help them develop quality programs on their own. Even though they will not work full time as professional programmers, I believe that this category of programmer will ultimately comprise the vast majority of the people writing programs. Since they will not spend all—or even most—of their time writing programs, we must simplify our languages and develop new languages that are designed for casual use. We must develop tools and support systems that will help these people to produce high-quality programs at reasonable rates and costs. We must also tailor support systems so that writing applications to run on top of a well-designed systems program will not require extensive technical support and hotline consultation.

In sum, what I am proposing is that, instead of having more and more trained programming professionals, we will solve our programming needs by teaching everybody to program. Although these people will not be professional programmers, they will be even more important to the software community because they will be our most demanding customers. They will be operating at the limits of the systems we software professionals provide, and they will be the first to identify important new opportunities. Therefore, they will probably be the source of much of the future innovation in our field.

In the next few columns, I will write about the trends in system programs, what they mean for the programming community, and the implications of these trends for software engineering.

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 David Carrington, Sholom Cohen, Don McAndrews, Julia Mullaney, Bill Peterson, and Marsha Pomeroy-Huff.

In Closing, an Invitation to Readers

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

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

References

[Clark 00] David Clark. “Are too Many Programmers Too Narrowly Trained?” *IEEE Computer*, March 2000: pp. 12-15.

[Rubin 99] Howard Rubin. “Global Software Economics.” *Cutter IT Journal*, March 1999: pp. 6-21.

[U.S. 99] The U.S. Department of Education, National Center for Education Statistics. National Education Survey (HEGIS), “Degrees and Formal Awards Conferred.” “Completions” survey and Integrated Postsecondary Education Data System (IPEDS), June 1999.

About the Author

Watts S. Humphrey founded the Software Process Program at the SEI. He is a fellow of the institute and is a research scientist on its staff. From 1959 to 1986, he was associated with IBM Corporation, where he was director of programming quality and process. His publications include many technical papers and six books. His most recent books are *Managing the Software Process* (1989), *A Discipline for Software Engineering* (1995), *Managing Technical People* (1996), and *Introduction to the Personal Software Process*^{SM} (1997). He holds five U.S. patents. He is a member of the Association for Computing Machinery, a fellow of the Institute for Electrical and Electronics Engineers, and a past member of the Malcolm Baldrige National Quality Award Board of Examiners. He holds a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, and an MBA from the University of Chicago.