Carnegie Mellon
Software Engineering Institute

news @ sei
interactive

Volume 5 | Number 1 | First Quarter 2002

In This Issue

columns

features

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

news @ sei interactive

2002 by Carnegie Mellon University

The Software Engineering Institute (SEI) is a federally funded research and development center sponsored by the U.S. Department of Defense and operated by Carnegie Mellon University.

® Capability Maturity Model, Capability Maturity Modeling, Carnegie Mellon, CERT, CERT Coordination Center, and CMM are registered in the U.S. Patent and Trademark Office.

SM Architecture Tradeoff Analysis Method; ATAM; CMMI; CMM Integration; CURE; IDEAL; Interim Profile; OCTAVE; Operationally Critical Threat, Asset, and Vulnerability Evaluation; Personal Software Process; PSP; SCAMPI; SCAMPI Lead Assessor; SCE; Team Software Process; and TSP are service marks of Carnegie Mellon University.

TM Simplex is a trademark of Carnegie Mellon University.

From the Director

In past years, the Software Engineering Symposium has provided us with an opportunity to present our technical program to key stakeholders within the DoD. We had high hopes for a very successful symposium in Washington, D.C. on October 15-18, 2001. However, as a result of the catastrophic events of September 11, we ultimately concluded that we had to cancel this event.

One of our goals for the symposium this year had been to engage with the DoD acquisition community. In light of the cancellation of our symposium, the SEI hosted a DoD Software Collaborators Workshop for the DoD acquisition community in our

Arlington, Va. facility on January 22-24, 2002. This event included presentations about the SEI technical program along with planning sessions with the DoD Software Collaborators (http://dodsis.rome.ittssc.com) and other DoD personnel. The event provided us the opportunity to meet with and understand our DoD stakeholders' needs and to discuss transition and adoption strategies, success criteria, and outcome metrics with these stakeholders. We intend to continue to build on the many positive collaborations that were fostered at this workshop.

The success of this workshop, along with the continuing growth and success of other targeted conferences and workshops that we sponsor, has led us to consider whether there might be more effective venues than the Software Engineering Symposium for disseminating information about SEI work. Past symposia have been organized in multiple tracks, with each track representing a focus area within the overall SEI technical program. However, as the software engineering community has grown, and as the SEI's impact has increased, we have found it increasingly difficult to cover our focus areas in sufficient depth in a single conference.

Over the past few years, the SEI has provided the stimulus for the emergence of focused communities of interest/excellence in specific areas of software engineering, and has supported these emerging communities through conferences and workshops. Examples include

- The annual Software Engineering Process Group (SEPG) Conference (http://www.sei.cmu.edu/sepg)
- The International Conference on COTS-Based Software Systems (http://wwwsel.iit.nrc.ca/iccbss), to be held for the first time in 2002
- The annual Software Product Line Conference (http://www.sei.cmu.edu/SPLC2)
- The annual Information Survivability Workshop (http://www.cert.org/research/isw/isw2001)

For the reasons I have cited here, we have decided to discontinue our annual Software Engineering Symposium so that we can focus more of our efforts on these conferences and

workshops, which serve specific communities of interest. We believe that this decision will result in events that more effectively transition best practices within the process, COTS-based systems, product line/software architecture, and survivable systems communities.

We plan to continue to report on the full body of SEI work through such media as the SEI Web site (http://www.sei.cmu.edu), news@sei and news@sei interactive (http://interactive.sei.cmu.edu), and the CERT Coordination Center Web site (http://www.cert.org). This year for the first time, we are also publishing an annual report for fiscal year 2001. I also encourage you to get involved in and support your local Software Process Improvement Network (SPIN). I or my associates at the SEI are always willing to come and present at a SPIN meeting, as a way of supporting the community in learning more details about the SEI. If your SPIN would like to schedule a visit from an SEI speaker, please contact SEI Customer Relations at customer-relations@sei.cmu.edu.

Stephen E. Cross
SEI Director and CEO

Security Matters

The Internet—Friend or Foe?

Lawrence R. Rogers

The answer to almost any question is probably on the Internet, but there can be risks in going after it. By posting a query to a newsgroup, for example, you can provide enough information to an intruder to give him or her strong leads for mounting an attack on your system. Having an awareness of the kinds of data that intruders can capture and use in perpetrating attacks will help you stay on friendly terms with the Net.

When I was growing up back in the late 1950s and early 1960s, our local grocery store sold Funk & Wagnalls encyclopedias, and they gave discounts based on the amount of groceries that you purchased. It took our family a long time, but we eventually became the proud owners of our own set of green Funk and Wagnalls. In those days, having your own encyclopedias was one of the few ways to acquire information, especially the kind we needed to write reports for school. Plus, they were convenient. If you had them at home, you didn't have to bother your parents to take you to the library to do your research.

"It's On the Internet"

Now the world is vastly different. No matter what the question is, it seems that the answer is "It's on the Internet." Long gone are the Funk & Wagnalls, or anything similar for that matter, from the grocery store. (In fact, *Funk & Wagnalls New Encyclopedia i*s now available on CD-ROM for your computer.) As the Internet spreads to more and more households, information that you were once able to buy or receive in the mail—store catalogues come to mind—will be available on the Internet, perhaps only on the Internet. If you are on the Internet from your home, you have instant access, especially if you have a cable modem or a DSL connection. The world is at your fingertips!

People are also instantaneously accessible, as are archives of discussions on various topics. You can send a message to anyone with an email address; and if that message goes to a discussion list, more than likely it will be archived and indexed so that others can benefit. Again, no matter what the question, the answer is probably on the Internet somewhere. Even finding those answers is less of a challenge than it was a few years ago. There are many indexing engines that sweep the Internet and capture what is needed to allow you to search the myriad of sites that are connected. The information is out there for the taking, and it is becoming easy to find.

Potholes on the Information Superhighway

But risks go along with this convenience—for home and commercial users alike. Imagine, then, that you are a systems administrator and you are having some trouble with a piece of technology, say the integration of a shopping cart application with your Web server under an operating system. The Internet to the rescue. You peruse the related vendor support Web pages and, failing to find just what you are looking for, you begin to search the appropriate news groups and archives of email discussion lists. You find some items that are close to your problem and that match your configuration, but not exactly.

To make sure you have the right set of circumstances and problem solution, you decide to post the following to a news group:

```
From: Joe Sys-Admin joeSA@FledglingEcommerceStartup.com
Date: Mon, 2 Apr 2001 10:08:48 -0600
Subject: Grelnob's Shopping Cart App on MacroHard's SSI Server

Dear Fellow Systems Administrators:

I'm trying to install Grelnob's Shopping Cart Application, Version
The.One.With.Bugs, under MacroHard's SSI Server, Version
The.One.With.Bugs, on a FarmerInThe platform with 2 processors,
256Mb of memory, and 20Gb of disk. The error I am getting is:

               Cannot find application library

But I know that I have it installed in the same location as the SSI
Server. Anybody else have this problem? Please drop me a line or
give me a call at 1-800-555-1212. TIA!
```

This certainly seems harmless, doesn't it?

Consider this: you are an intruder and you have selected Fledgling Ecommerce Startup as your next target. Normally, you need to do an amount of reconnaissance of your target before you attempt a break-in. This message from Joe is a gold mine of information, saving you potentially several weeks of work. Let's see what could be learned from this message alone:

- In the domain named FledglingEcommerceStartup.com, there is an account named joeSA. Now all you need is a password, and you may be able to login to one of their machines. You are half-way home.
- The machine used to send this mail is in the Central time zone (-0600 or 6 hours west of Greenwich Mean Time), so now you have an idea of the working hours of the staff–when people are likely to be in and out of the office.

- The software configuration and version of two key components of Fledgling Ecommerce Startup's business, namely Grelnob's Shopping Cart application and MacroHard's SSI Web server.

- The hardware configuration of one of their servers.

- The telephone exchange that you could use in a war dialer (automated dialing) attack.

Wow! That's a lot of information "leaked" to the world at large, and all in the name of solving a simple problem. And, there's probably more information encapsulated in the Received and Message-ID headers that are not shown in this example. A gold mine indeed! To learn even more about your target, you could search the archives of various news groups and discussion lists to see if old Joe or anyone else from FledglingEcommerceStartup has posted questions. This may give more clues about hardware and software configurations and other accounts that may be available to you when trying to gain access. You could even build on what you already know by sending Joe a response to his question. You'll get more information from his inevitable response. Reconnaissance comes in many flavors.

What should Joe have done? The key point is connecting the configuration information with an email address and, therefore, with a specific site. By breaking this relationship, Joe could have asked these same questions and still gotten the information he needed to solve his problem without leaking extraneous information. One way to achieve this is by using another email site like hotmail.com or lycos.com, for example, and not Joe's production site. Unfortunately, the telephone number is a bad idea no matter what the source of the email. Sorry, Joe, but the Information Superhighway is littered with potholes.

The Internet is indeed your friend and can significantly speed the flow of information that you need to solve problems when building cost-effective and secure configurations. However, there is a cost, and frequently that cost is difficult to recognize, let alone quantify. That's what makes it your foe.

The message here is that virtually every time you access another computer on the Internet, whether from work or home, you are leaking information. Be aware of what is happening and seek ways to minimize the information that you provide. You never know who's watching. Now, where's my old Funk and Wagnalls?

About the Author

Lawrence R. Rogers is a senior member of the technical staff in the Networked Systems Survivability Program at the Software Engineering Institute (SEI). The CERT® Coordination Center is a part of this program. Rogers's primary focus is analyzing system and network vulnerabilities and helping to transition security technology into production use. His professional interests are in the areas of the administering systems in a secure fashion and software tools and techniques for creating new systems being deployed on the Internet. Rogers also works as a trainer of system administrators, authoring and delivering courseware. Before joining the SEI, Rogers worked for 10 years at Princeton University. Rogers co-authored the *Advanced Programmer's Guide to UNIX Systems V* with Rebecca Thomas and Jean Yates. He received a BS in systems analysis from Miami University in 1976 and an MA in computer engineering in 1978 from Case Western Reserve University.

This and other columns by Larry Rogers, along with extensive information about computer and network security, can be found at <http://www.cert.org>.

The Architect

Architectures for Adaptive Mobile Systems

Rick Kazman

The architectural paradigms that have been used to build non-mobile systems do not fit the needs of mobile systems particularly well. Mobile systems require effective methods to manage and control their resources, particularly in the face of changing environmental conditions and user needs. SEI staff members are examining the application of *adaptive architectures* to maximize the user-defined utility of mobile devices within the limits of their available resources.

Introduction

The computational environment that the average person is exposed to is changing rapidly these days. In particular, we are seeing the widespread commercial presence of mobile systems: cell phones and personal digital assistants that combine voice and data communication and other applications on a single device. And this trend towards mobility appears to be steadily increasing. But the architectural paradigms that we have been using to build non-mobile systems in the past do not fit the needs of mobile systems particularly well. For one thing, mobile systems are, and always will be, more constrained with regard to resources than their stationary counterparts. As a result, the architecture of mobile systems has to be carefully balanced between the physical size of the system and its computation, communication, and energy capacity. And resources need to be allocated differently on mobile systems. Currently, resource-allocation decisions on mobile systems are almost always fixed at the time of system creation. This is partly due to the very limited resources available on most mobile systems and partly because typical mobile systems are used for a small set of operations that are well known in advance (e.g., phone calls, web browsing, short message service, etc.).

Adaptive Architectures

This situation is changing, however, and modern mobile systems are becoming more powerful in terms of the computation and communication resources that they can call on. At the same time, as they become ubiquitous, the demands being placed on them are increasing. For this reason, such systems must have effective methods to manage and control their resources, particularly in the face of changing environmental conditions and user needs. At the SEI, we are examining the application of *adaptive architectures* to maximize the user-defined utility of mobile devices within the limits of their available resources. The novelty of this approach is that we are making resource-allocation decisions based upon user-defined notions of utility (essentially a subjective notion of goodness), as compared with traditional approaches to resource scheduling, which focused on purely internal or technical notions of goodness, such as throughput, latency, or resource utilization.

Mobile systems must be adaptable for a number of reasons: they have limited resources; user needs are constantly changing as the user's environment changes; and resource availability constantly changes as the user moves around, as environmental conditions change, and as time passes. For example, as users move around, the signal strength that they are receiving will change. And as time passes, the amount of available energy (battery) normally decreases.

A Utility-Based Approach

We believe that the best way to make adaptive mobile systems is to use a user-defined utility-based approach to resource allocation. By choosing "utility" as the central concept on which adaptation rests (rather than, say, more traditional notions such as maximization of CPU utilization or throughput), we can create systems that adapt in ways that more closely mirror a user's needs, which may be changing dynamically. For example, at one moment a user might want to surf for some files on the Internet, but if an important call comes in from a client, he or she might want to allocate more resources to that call and less (or even none) to downloading a file from the Web.

Most common scheduling and resource-allocation approaches rely on relative priorities of computational elements, such as threads, processes, and programs. However, on a personal mobile device this approach must be extended to a utility-based resource-allocation mechanism, where utility is purely user defined. In this view of the world, application services that have a high utility to the user get a preferential allocation of resources.

A mobile personal-communication device serves multiple purposes. That implies change in user preferences. User mobility implies inevitable changes in environment, affecting connectivity to communication and sources of energy. Mobility also leads to change in the dominant use of the device. Utility-based resource allocation must be dynamically adaptable to these changes. A direct consequence of this is that mobile applications are expected to be able to run at different fidelity levels providing a different level of utility to the user and having different resource requirements. For example, if a user is browsing the Web, different fidelity levels might take the form of text only, text with minimal graphics, and text with full graphics. Voice communications might have different levels of fidelity corresponding to different CODECs that run at different data rates. Video communications might have different resolutions, color depths, and numbers of frames per minute. And so forth. In each case, the resource requirements for these applications are different, and the resulting utility that they provide to the user is different. In addition, a user's subjective view of the utility of these different levels of fidelity might change depending on the situation. For example, the user might demand high voice quality for an important business call with a client, but might be perfectly content with a low voice quality when

checking sports scores or receiving unsolicited calls. Even within a single run of a single application, the fidelity demanded by a user might change dynamically.

Clearly the design space in creating such systems is enormous. There may be different configurations of the hardware and software components, and different quality attributes that the user may want to optimize (e.g., communication speed, battery life, and performance). This enormous design space poses a problem for designers—how can they go about comparing architectural alternatives for such systems without having to go to the expense and risk of building the systems?

Exploring the Design Space

To be able to conduct research in determining appropriate architectural alternatives for dealing with all of these complexities, we need to create a simulation test bed. The research questions that we propose to answer from this simulation test bed can be organized into two broad categories:

> **Implementation**: Is it possible to build a system that incorporates user preferences and dynamically optimizes the assignment of resources to maximize user satisfaction? What are the constraints on building this type of system? What are the resource-consumption characteristics of various processes (including the optimization process) and how do they interact?
>
> **Architecture**: What are the various architectural structures that can be used to provide dynamic allocation of resources based on user preference? Is it possible to prescribe architectural designs for different types of mobile devices, based on the desired characteristics of those systems?

To aid in our research we are building a test-bed simulator that simulates the various components of a mobile device, the user, and the environment in which the mobile system is operating. The problem of creating such architectures is challenging because the utility that the user gets from the system can change with respect to the changing environment. At one part of the day a user might desperately want to receive the latest download of sports scores or Wall Street numbers from the web, and at another point this same user might have an important voice call to make and would like all resources to be diverted from other tasks to be concentrated on this one high-priority task. For this reason the device has to adapt to the changing needs of the user. For this reason, it seems crucial to have a simulator in which different architectural choices can be compared before building the system itself. In the final mobile system, such changing needs would either require a change in the scheduling algorithm or even a change in the control and data flow across components; in the simulator they require only a change of an initialization file.

The purpose of the test bed is manifold:

- to provide a tool through which one can explore methods for incorporating user preferences on mobile devices to maximize the utility of services provided
- to explore different scheduling strategies
- to do performance modeling
- to explore the impacts of architectural alternatives and their relationship to user preferences and profiles

As shown below in Figure 1, the test bed allows a variety of resources to be specified (in this case CPU, memory, network bandwidth, and battery). The test bed further allows a user to specify an architectural configuration of these resources, along with the tasks that use them. In this way, changing the architecture of a simulated system is as easy as changing a simple specification file.

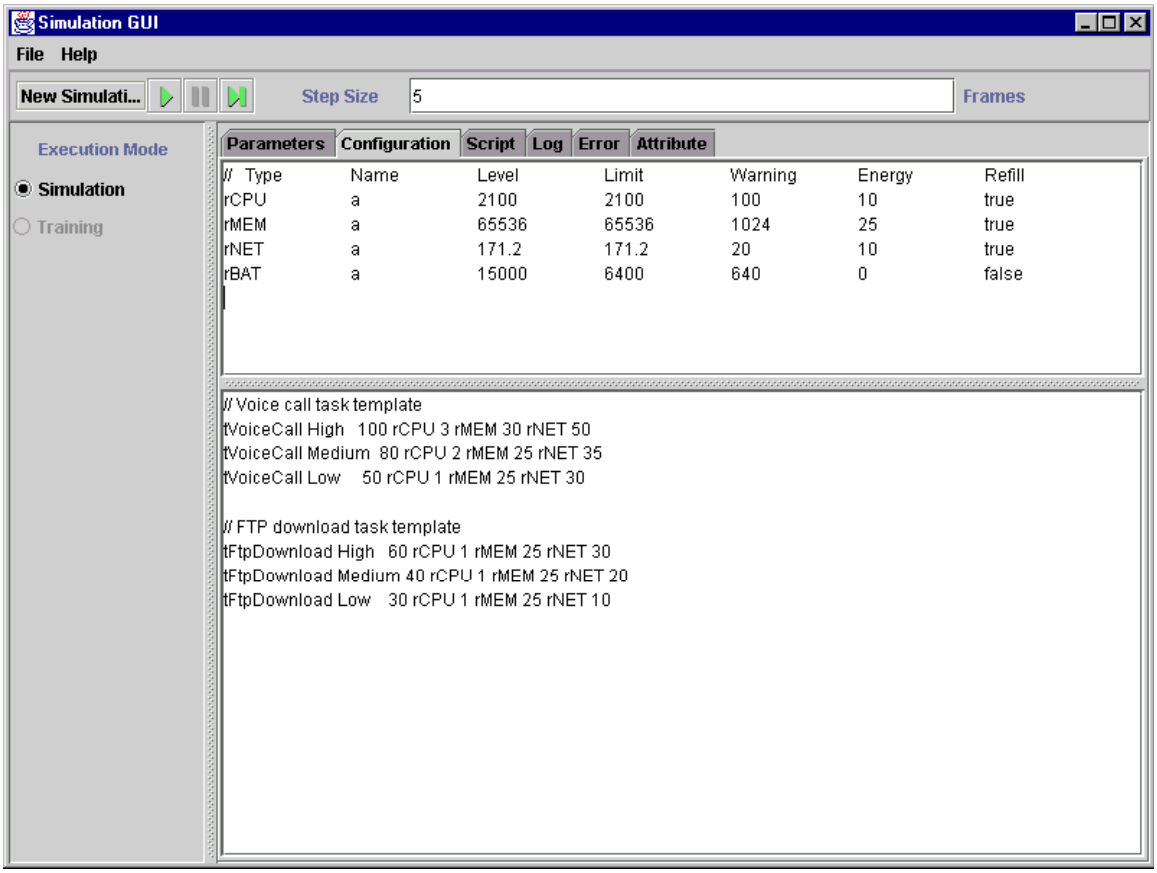

Figure 1: The AAMS Test Bed in Configuration Mode

During execution, the test bed executes a script, which is also specified by the user, and the results of the simulation are updated in real time or at any rate that a user specifies. A portion of a sample run is shown in Figure 2 below.

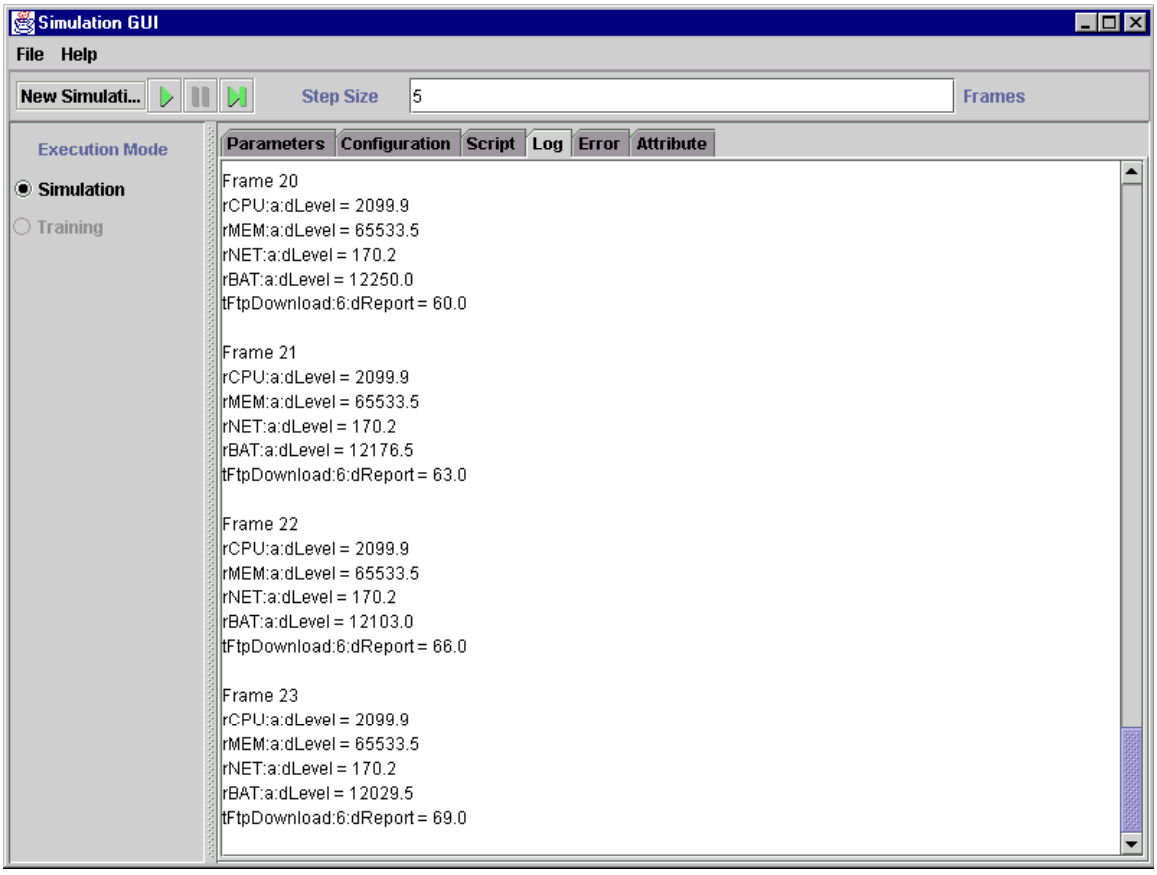

Figure 2: The AAMS Test Bed Displaying an Execution Log

In addition to scripted events, the test bed supports the generation of stochastic events, such as changes in environmental conditions and new user requests. This allows test runs to be more realistic and removes the burden on the user to specify every detail of a simulation.

We feel that this test bed will be a valuable tool in exploring the design space surrounding adaptive mobile systems. We are currently implementing the test bed and expect to be testing it on real-world examples within the next four months.

About the Author

Rick Kazman is a Senior Member of the Technical Staff at the SEI. His primary research interests are software architecture, design and analysis tools, software visualization, and software engineering economics. He is the author of over 50 papers and co-author of several books, including *Software Architecture in Practice* and *Evaluating Software Architectures: Methods and Case Studies*.

The COTS Spot

Building Systems from Commercial Components: Classroom Experiences

Robert C. Seacord

It often happens that a teacher learns as much in the classroom as his or her students. Columnist Robert Seacord shares insights he gained from teaching systems engineering students to integrate commercial off-the-shelf (COTS) and custom components in their system designs. Seacord anticipates using these insights to improve COTS-based development processes.

Introduction

There are two points of view regarding software engineering. The first holds that human intelligence is a constant, and to improve the quality and reduce the cost of developing software-intensive systems, we must improve the techniques, technologies, process, methods, and tools we use to build software. The second holds that, if you take an axe away from a simpleton and replace it with a chainsaw, he will still cut off his leg—only now he will do it more efficiently. The solution, in this case, does not require that you improve the tools, only that you improve the users' ability to apply the tools they already have. Hence, we have software engineering education.

Armed with the latter philosophy, I set off this spring to teach software engineering at the undergraduate level. When I took software engineering as an undergraduate in 1983, software engineering still meant "Ada." Unfortunately, there was no Ada compiler available for our use at that time, so we got to scribble our designs on sheets of paper and hand them in for credit. As this was not my fondest memory of my undergraduate years, I decided not to share this experience with my students. Instead, I fashioned a software engineering course from Ian Sommerville's text on software engineering [1] and my own text, Building Systems from Commercial Components [2].

While one hopes to fashion young students into future software engineers (and not scare them away to the fast food industry), I think it is often the case that the teacher learns as much as the student. In this column, I will discuss one of the lessons I've learned from this experience and how I see this lesson potentially improving COTS-based system processes down the road.

Software Engineering with Commercial Components

The software engineering course I am teaching consists of two main elements, a semester-long project developed by teams of students, and the classroom "experience," which culminates in a final examination. The project teams are assigned based on the result of a student assessment, in the hope that balanced teams can be formed.

The actual assignment is to develop an enterprise information system that could be used by car dealerships for tracking maintenance activity on cars under warranty. Mechanics would use the system at individual service stations to record all maintenance activity. Senior management would use the system at an enterprise level to collect and analyze maintenance trends. In addition to functional requirements, several technology requirements were also imposed, including requirements that the system be Web-accessible, use Enterprise JavaBeans™ (EJB) in the middle tier, and store data in a relational database management system (RDBMS).

Execution of the project was separated into three phases, with milestones established at the end of each phase identifying required products that must be handed in for grading. Each set of milestone deliverables was given equal weighting:

First milestone:

1. a Unified Modeling Language™ (UML) specification/design for the system
2. blackboards for three component ensembles

Second milestone:

1. completed blackboards for all three ensembles
2. risk/misfit evaluation

Third milestone:

1. PowerPoint presentation and demonstration to be delivered in class
2. source code and URL for the working system

™ Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. The Software Engineering Institute is independent of Sun Microsystems, Inc.

™ Unified Modeling Language and UML are trademarks of the Object Management Group.

The goal of the project was to model, as closely as possible in a classroom situation, a real-world development project. In the initial milestone, the students were asked to create a design for the system, as well as three *component* ensembles. Component ensembles are groups of compatible components that can be used to achieve the system objectives. In this case, these components are all infrastructure components. The students were asked to vary each ensemble at least in the choice of an EJB server (since the choice of an EJB server, more so than the other components, drives other component choices). Blackboards are UML-based notation for describing component ensembles. Component ensembles and blackboards are both subjects of *Building Systems from Commercial Components*, and are taught in the classroom.

In the initial set of blackboards, the students are told to identify both "knowns" and "known unknowns." These are expressed in terms of credentials (facts associated with a degree of confidence), postulates (required information associated with a plan to acquire), and constraints on component interactions. Students are penalized for failing to identify unknowns ("unknown unknowns").

The second milestone consists of two deliverables. Postulates in the blackboard are replaced with credentials, with the overall effect being that the feasibility (or infeasibility) of the ensembles is demonstrated. Student teams are required to produce at least two feasible ensembles from the original set of three, so that these ensembles can be evaluated using the Risk/Misfit techniques also described in *Building Systems from Commercial Components*. A single ensemble is then chosen as a result of the evaluation, and this ensemble is driven through to implementation.

A Lesson Learned

One lesson I've learned (so far) from administering and evaluating project deliverables is that, once someone is enmeshed in the field of COTS-based software engineering, it is too easy to de-emphasize or ignore non-COTS aspects of the development effort. For example, in the first milestone, the students were required to create both a UML design for the system and a set of blackboards. Performing these tasks concurrently is a practical necessity for any real-world development effort as well as a class project, as component compatibility must be accessed while the system specification is created and custom components are designed. This introduces two interesting and opposed sets of challenges—incorporating constraints imposed by the component ensembles on the design while developing a design specification independent of any particular ensemble.

The solution is a tightrope-walking exercise in integrating common, technology-driven aspects of the component ensembles with the design, while designing abstract classes not tied to a particular product for implementation. Some student teams also jumped right off the tightrope and specified generic sequence diagrams for interactions among technology components. These diagrams were a reasonably useful extension of the blackboard.

Once a component ensemble has been selected, it is now feasible (and necessary) to allow component-specific concerns to influence the design. Still, these designs should be maintained in a configuration-management system that directly connects the artifacts to the selected design contingency.

In retrospect, this lesson was evident from the beginning in my choice of two texts for the course. Eventually, to be done right, this course will need to be taught with a single text that applies all the traditional areas of software engineering, including software process, project management, requirements engineering, architectural design, verification and validation, and software testing with COTS-based development practices.

Summary

Teaching software engineering using commercial components at the undergraduate level provides a great opportunity to evaluate these processes in a controlled setting, since students in general are not shy about complaining about all the hardships being forced on them. I hope that by repetition of this course, combined with refinement of the processes and industry involvement, a more prescriptive COTS-based development process can be formulated.

References

[1] Sommerville, Ian. *Software Engineering*. New York: Addison-Wesley, 2000, ISBN: 020139815X.

[2] Wallnau, Kurt C.; Hissam, Scott A.; and Seacord, Robert C. *Building Systems from Commercial Components*. Boston: Addison-Wesley, 2002, ISBN: 0201700646.

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.

For more information on the book *Building Systems from Commercial Components*, please visit this Web page: <http://www.sei.cmu.edu/cbs/bscc/bscc.htm>.

Watts New

The Future of Software Engineering: V

Watts S. Humphrey

In the previous four columns, I discussed application programming, systems programming, and some of the likely future trends in these areas. In this column, I talk more broadly about the overall trends in our industry and what we will likely see in the future. In particular, I address the forces at work on software-intensive businesses and how businesses are likely to change in response to these forces. I then close with some comments on how software professionals can better prepare themselves for the challenges of the future.

As I discussed in the previous columns, there are some differences in the forces on the systems, applications, and middleware businesses, but there are also some commonalities. These common forces will impact all of us, regardless of what we do. First, to segment the discussion into manageable chunks, I start by reviewing the principal kinds of software businesses. Then, I discuss the common forces on our industry. Next, I talk about the implications of these forces on a software-intensive business. Finally, I explore how these forces will likely change the kind of work that software people do and what this is likely to mean for each of us.

While the positions that 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 that I express are entirely my own.

The Kinds of Software Businesses

As I noted in the earlier columns in this series, we can expect the volume of application development work to continue growing. This work will require people who know and understand various application domains and are also competent programmers. For such work, skill needs will span the full gamut from traditional business and accounting applications to embedded controllers that manage complex devices and processes. On the other hand, systems developers will principally be concerned with developing, maintaining, and enhancing the operating and support systems needed by the application development community.

The operating systems community will be split into three rather loosely defined groups: the software houses like Microsoft and Oracle; the systems businesses like Apple, Sun, and IBM; and the growing body of open source programmers supporting systems like Unix and Linux. While it is too early to tell exactly how this business mix will evolve, the fuzzy middleware boundary between the operating systems and application worlds is where a large body of people are now developing and marketing software. Their objective is to fill the cracks between the

operating systems and application domains. As pointed out in Part IV of this series of columns, this middleware business faces unique challenges, and it is likely to be the principal competitive battleground for the next several years. This intersection was precisely the focus of Microsoft's antitrust lawsuit, and it will continue to be both the legal and competitive focus for some time to come.

We can also expect the interface between the systems and application worlds to fluctuate in response to evolving user needs. The development community that most quickly and effectively satisfies users' needs is likely to earn a larger share of the competitive pie. While various suppliers may use contractual or other means to force users to use their products exclusively, such strategies have only been temporarily effective in the past, so they are not likely to work over the long term. Of course, the long term could be very long indeed. However, if an offering is not truly in the users' best interests, sooner or later it will lose out to the better competitor. While the fight may take a long time and it may be won either in the marketplace or in the courts, the final result is inevitable.

The Forces on Software Businesses

There are many forces on software businesses, but the most significant ones I see today are the following:

- The functional content and complexity of systems is increasing rapidly, as is the size of the software parts of these systems. As noted in the first column of this series, the size of the software used for any given function has been growing by roughly 10 times every 5 years. If software growth continues at this historical rate, this will mean an increase of 10,000 times in the next 20 years.

- Increasingly, software plays a central role in controlling and managing systems. Software is not just getting bigger, it is a crucial part of the products and services in almost all industries.

- Most computing systems will be interconnected. The Internet is merely the latest step in the long progression from stand-alone computing to pervasive computing networks.

- We will see more internal and external threats to our systems. In the past, when our principal preoccupation was getting systems to work, we assumed a friendly and law-abiding environment. Now, in the interconnected world, these systems must work, be safe, and stay secure, even when exposed to attack by criminals and terrorists.

One could write pages on each of these topics but I will just state these forces as facts and deal with their implications. I next talk about each of these four forces and what businesses should do about them.

Software Size and Complexity

The principal concern with size and complexity is the scalability of the development process. To handle the massive increases in system scale, organizations must employ processes that scale up. To appreciate the scalability problem, consider transportation. Vastly different technologies are involved in traveling at 3 miles an hour, 30 miles an hour, 300 miles an hour, or possibly even 3,000 miles per hour. You can't transition from one speed range to the next by merely trying harder. You need progressively more sophisticated vehicles that use progressively more advanced technologies.

Unfortunately, we have yet to learn this lesson in software. We attempt to use the same methods and practices for 1,000 LOC programs as for 1,000,000 LOC systems. This is a size increase of 1,000 times, and the commonly-used test-based software development strategies simply do not scale up. Organizations that don't anticipate and prepare for scalability problems will someday find that they simply cannot get systems to work safely, reliably, and securely, no matter how much their people test and fix them. When systems hit this wall, you can either test until the available time or money runs out, or you can scrap the system and do it over again correctly. Unless you are a Microsoft or an IBM, however, you probably can't afford to start over. As systems get larger, we can expect most organizations that keep following their current test-based processes to face this problem. It is only a question of time until they do.

The Central Role of Software

The second force is software's now central role in controlling and managing business-critical systems. This is because much or even all of the functions that customers find attractive about modern products and services is embodied in their software. And it is just these attractive functions that make products unique in the competitive marketplace.

Many executives view software as a problem that they don't understand and have no idea how to manage. They try to subcontract their software work or to find some other magic solution that will relieve them of the problems of managing software. This is almost always a mistake. When management subcontracts the technologies that make their products unique, they lose the ability to manage their future. I have just published a book that discusses this problem and some of my experiences with it. It might give you some ideas on what to say to management about the importance of software in your organization [Humphrey 2002].

Interconnectedness

The third major force on the software industry concerns the growing interconnectedness of systems. Much like telephone systems, the value of a computing system is increasingly

determined by the number of other systems to which it can connect. In the past, companies could focus on something IBM used to call "exclusivity." Now, however, systems that only work with hardware and software from one vendor are less and less attractive. In the old "exclusivity" days, IBM would sell, install, service, or support systems only if they were composed entirely of IBM products. As long as IBM was the dominant supplier of all important offering elements, this strategy worked quite well. But with the advent of the PC and the rapid introduction of many PC clones, IBM could no longer force its customers to use its products exclusively.

So, in the interconnected world, the keys to broad market acceptance are compatibility, interoperability, and interchangeability. Each of us is working on a small part of one enormous, world-wide, borderless computer-plex, and we are just now glimpsing its implications. While it is hard to predict what this trend will mean, it is clear that this new environment will force us out of the comfort and security of single-system thinking. We will need to think in interconnected ways and to remove any limitations that make it hard to interconnect and to interoperate our systems. We must recognize that the interconnected systems of the future will be used in ways that their developers could not imagine. It is precisely this ability of our systems to be used in new and innovative ways that will make them attractive to the users of the future.

Real-World Threats

The fourth force on the software industry is one we are just now facing. This new world is vastly different from the closed and comfortable one of the past. It is populated with many wonderful people but also with a few unpleasant, inconsiderate, and even threatening characters. As long as our systems were stand-alone, our exposure to the realities of a dangerous and unpleasant world were limited. But with the Internet, and with the growing interconnectedness of our systems, this is no longer the case.

This new environment will affect businesses in many ways. In particular, as we increasingly invoke the law to punish miscreants, those who have been damaged will also seek to recover damages from the organizations that built unsafe or insecure systems. Soon, secure and safe systems will be an economic necessity, and users will band together to seek damages from suppliers who don't provide such systems. There is currently a movement to modify contract law to protect software vendors from these problems. It is called UCITA, or the Uniform Computer Information Transactions Act. While UCITA has been enacted into law in two states (Maryland and Virginia), there is growing opposition to it and further expansion is unlikely unless it is substantially changed.

What these Trends Will Mean to All of Us

Regardless of your place in this future, there are some strategies that you should consider, both to make your organization more competitive and to make your personal employment more secure and rewarding. The first and broadest consequence of the increasingly central role of software is that most professional workers will be involved in developing, supporting, marketing, or using software. As a consequence, the trends that affect the software world will also affect most of us. The principal challenge is to have the vision and imagination to capitalize on this future world and to help make it happen in an orderly and useful way.

The most obvious force on our industry is security. We will probably always have criminals and terrorists, so we must write our programs to operate in a threatening and unfriendly world. To appreciate what this means, consider that over 90% of the Internet's software security vulnerabilities result from common types of software defects. That means that the software security problem is, at least for now, a quality problem. If quality was not important before, it soon will be. This suggests that you should examine your personal quality practices and look for and adopt a set that are demonstrably effective. Then, follow these quality practices religiously. While you should consider all of the available candidates, my personal recommendation is the Personal Software Process℠ (PSP)℠ [Humphrey 1995].

From a project and organizational perspective, you should also look for processes that are demonstrably scaleable. When you find a scaleable process that fits your organization's needs, start a movement to adopt that process. While you might argue that one working-level developer could not possibly get a business to make such a change, every important change is started by one person, and that person is rarely a manager or an executive. Usually, it is someone like you who is close enough to the problem to appreciate its implications. Talk to the people around you, build a support network, and then start talking to the managers. You will be surprised at what you can accomplish.

Regarding a scaleable process, my favorite is the Team Software Process (TSP)SM [Humphrey 2002]. However, before you pick your candidate process, look around and see what other methods are available. Also look for documented evidence of the effectiveness of these processes. Then, pick up the spear and get this method adopted by your organization. After all, in the last analysis, it is your job you are fighting for.

℠ Personal Software Process, PSP, Team Software Process, and TSP are service marks of Carnegie Mellon University.

The Accelerating Pace of Change

To appreciate what these trends mean for each of us, remember that the world is now changing faster than it ever has before. The accelerating pace of change has been with us for so long that it seems almost trite to discuss it, but it does mean that the tools and methods we will use in the future will be vastly different from those that we use today.

In describing what this means to you and me, the best example I can think of is my personal experience. When I graduated from college in 1949, ENIAC, the first digital computer, had just recently been demonstrated. After a few years of graduate school and a brief university job, my first industrial position was designing a digital cryptographic system. Within two years, I was designing computers, and I have been working with computers ever since. Except for a class that I took in cost accounting, not one of my other college courses has been directly applicable to my subsequent work. This does not mean that my education was wasted but just that it was not enough. In this rapidly changing world, if you do not keep learning and remain open to new ideas and challenges, you will not play an important or even a very useful role in this challenging and exciting future.

Some Final Comments

The future of software engineering is quite unpredictable, but we can perceive some trends, particularly by considering the forces at work on our industry. In the last analysis, it is up to each of us to continue learning and to continue preparing ourselves for the challenges ahead. Then we will be prepared to take advantage of whatever opportunities present themselves.

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 Don McAndrews, Julia Mullaney, Mark Paulk, and Marcia Pomeroy-Huff.

In closing, an invitation to readers

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

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

References

[1] Watts S. Humphrey, A Discipline for Software Engineering, Reading, MA.: Addison Wesley Publishing, 1995.

[2] Watts S. Humphrey, Winning with Software: an Executive Strategy, Reading, MA.: Addison Wesley Publishing, 2002.

SEI Architecture Practices Propel Successful Startup

Bill Pollak

Since 1996, the SEI has focused on software architectures in its Architecture Tradeoff Analysis (ATA) Initiative, based on two key premises:

1. Desired quality attributes in a system such as performance, modifiability, interoperability, and reliability depend more on software architecture than on code-level practices.
2. Quality attributes ultimately determine a system's success.

SEI architecture practices complement the SEI's well-known work in software process improvement by focusing on the design and quality of the product itself. A balanced product and process focus is important regardless of the development context—DoD contractor or "dot com" startup.

Jeromy Carriere and Steve Woods, former members of the SEI technical staff in the Architecture Tradeoff Analysis Initiative, are two of the founders of Quack.com, a startup company that was acquired by America Online (AOL) in August 2000. Their story and the story of the company they helped found demonstrate the broad applicability and utility of the SEI's work in software architectures. Jeromy Carriere came to the SEI from Nortel in 1997. Early in his tenure at the SEI, Carriere was an integral part of the SEI team that supported development of the Control Channel Toolkit[1] for the National Reconnaissance Office (NRO). He also led work at the SEI in architecture reconstruction and participated in defining the SEI's Architecture Tradeoff Analysis Method (ATAM).

"It was this concentration on architecture and the attention to quality attributes that differentiated them from their competitors and elevated them to a place of prominence"

After completing a PhD in constraint-based reasoning for reengineering at the University of Waterloo, Steve Woods did post-doctoral work in Hawaii, where he first met Alex Quilici and began to explore ideas with Quilici for startup companies. Woods came to the SEI in 1998,

---

[1] See *Control Channel Toolkit: A Software Product Line Case Study* (CMU/SEI-2001-TR-030), http://www.sei.cmu.edu/publications/documents/01.reports/01tr030.html.

where he immediately began to work on architecture reconstruction with Carriere and began discussing with Carriere some of the ideas that he and Quilici had been considering: initially, an Internet radio and other consumer wireless devices. As the discussions among the three continued, their ideas evolved first into a device that could aid a consumer during a shopping trip, and finally to a product that would enable speech recognition on a cell phone. "Eventually it became clear that we should try to make a go of it," says Carriere, "and that's when we decided to leave the SEI. All through this, we were developing an architecture to support our ideas. In those days, I had been steeped in the SEI ideas about architecture, a highly structural perspective, but one driven by quality—looking first at what qualities my system has to achieve, and then making architectural decisions to achieve those qualities. Architecture is usually conceived from a functional perspective—what a system needs to do—and then you allocate functions to components, build those components, and then cross your fingers and hope that they work together; and then cross your fingers again and hope that they meet your reliability, performance, and flexibility goals."

"We were thinking of the qualities first. The quality that we needed primarily to achieve was flexibility. And that's because we needed an architecture and a set of architectural principles that were going to be able to adapt quickly to changing market drivers and business goals; our architecture enabled us to be maximally flexible with respect to all of those things."

A Short History of Quack.com from AOL's Web Site[2]

Quack.com has had a short but exciting history. Quack was founded in late 1998 by a University of Hawaii professor (Alex Quilici) and two Carnegie Mellon University research scientists (Steve Woods and Jeromy Carriere). After a few false starts, they hit on a big idea: Make the best online commerce and content available to anyone with a telephone! Just by speaking. They built a cool little demo of their idea, and by late summer 1999, they had convinced a few angels and early stage VCs to fund them. They kept cranking away, not only building a "voice portal," but building it in a novel way. They could have just hacked it together, as just another complex software application, which was a challenge in itself. But they decided to try for something even better. They were going to build a speech application publishing platform and toolkit and then use that to build the voice portal. That would allow them to quickly build and maintain a wide variety of applications and potentially be the underlying platform for a whole new industry. By March 2000, only 9 months after raising their first funding, they released the first Web-based consumer voice portal. It allowed people to get information about weather, stocks, sports, traffic, movies, and restaurants. And only seven weeks later, on October 25, 2000, American Online released AOLByPhone, built by Quack's team and using Quack's platform and toolkit.

2 The text of this sidebar is taken from http://www.filmphone.com/aboutquack.html

Carriere and Woods attribute much of their success with venture capitalists (VCs) and ultimately with AOL to this quality-driven approach to architecture that was reinforced by their experiences at the SEI. “In most VC audiences,” says Carriere, “there is someone in the group whose job it is to see if you are actually going to be able to build what you say you want to build. This architecture-derived thinking was a core part of our pitch, and it seemed to resonate well with VCs. I think it served us well right up to the time when we were acquired by AOL. As we were being examined closely by AOL, we had all this thinking behind us that enabled us to answer the questions successfully.” In fact,” adds Woods, “AOL now says that they bought us because our technology was more flexible than anyone else in the space.” Although flexibility was primary, Carriere and Woods, consistent with the principles of the ATAM, also looked at the tradeoffs among other key qualities. When the acquisition by AOL changed the focus, making performance and scalability more important, they knew exactly what tradeoffs they were making.

“AOL is all about scale,” says Woods. “At any given time, there are roughly two million members online at the same time, which is an enormous number. That’s a kind of scale in systems that you just don’t see anywhere else. Performance and scalability were key goals of our architecture from the beginning, even though they were secondary to flexibility. And so we were prepared to make the transitions that our acquisition by AOL required.”

Carriere and Woods retained their disciplined architecture focus throughout. “I always had the goal of being rigorous with respect to documentation of the architecture,” says Carriere. “This is a specific outgrowth of the work with Control Channel Toolkit. What I learned there was the basis for how I documented the architecture for Quack, and how I still document architecture today.

Key to Quack’s success, says Linda Northrop, Director of the SEI ATA Initiative, were the principles that Carriere and Woods applied to the way they engineered products. “They believed in architecture.” she says, “They focused on the architectural platform that generated products rather than on any particular product itself. This meant that they did not have to redesign whenever functionality changed.

“This suggests that our architecture principles can serve any organization that is concerned about operating in the context of volatile requirements."

The architecture was designed so that functionality changes were local rather than topological; they abstracted interfaces wherever possible with external systems, and in fact, with systems they anticipated could be external. So they were able to put out revisions and change their product focus very quickly. And, it was a relatively quick task to integrate to AOL, which is a big deal for an acquisition. It was this concentration on architecture and the attention to quality attributes that differentiated them from their competitors and elevated them to a place of prominence.”

Northrop says that there is a lesson to be learned by the managers of large DoD systems from the success of Quack.com. “Architecture does not mean rigidity. The flexibility afforded by careful architectural design brought success to a small team of folks working in a volatile marketplace in which the definition of the project is changing on the fly,” says Northrop.

“DoD program managers often express concern about the inability to get requirements set. Imagine a world in which not only the requirements for a specific product change, but the whole product you’re building changes over and over again. That was the situation at Quack, and yet they were able to retain the same architecture. From the start, they had a strong set of architectural foundations and a reasonably accurate sense of the likely (or at least probable) change scenarios for the business. In other words, they applied the ATA philosophy. This suggests that our architecture principles can serve any organization that is concerned about operating in the context of volatile requirements. A carefully crafted architecture and disciplined architecture practices provide the necessary grounding.”

Quack.com absorbed some Netscape staff and became AOL Voice Services, which comprises a platform and any applications that are built on that platform. The AOLByPhone product is supported by the resources of the AOL Voice Services team. Woods and Carriere had leading roles on that team. Recently, however, following the completion of the version 2 platform in voice services, Woods and Carriere have moved to broader positions of impact in the AOL Technology Department. Carriere is a chief architect in the AOL Systems Infrastructure organization—a team of approximately 600 engineers responsible for building the core AOL services. Woods is VP, Systems Infrastructure. Both are now involved in a systems-wide AOL architecture effort.

For more information about the SEI Architecture Tradeoff Analysis Initiative, contact—

Linda Northrop

Phone
412-268-7638

Email
lmn@sei.cmu.edu

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

Cost Benefit Analysis Method

A software architecture is an essential part of a complex software-intensive system. For more than five years, the SEI has been performing software architecture analyses to help software developers to examine the consequences of architectural strategies before committing resources to implementing them.

Architecture analysis, using the SEI's Architecture Tradeoff Analysis Method (ATAM) focuses on a system's quality attributes, such as performance, modifiability, usability, and availability. The ATAM provides software architects, while designing or maintaining a software system, a framework to reason about the tradeoffs among these quality attributes. But the biggest tradeoffs in large, complex systems always have to do with economics: How should an organization invest its resources to maximize its gains and minimize its risks?

The Cost-Benefit Analysis Method (CBAM) picks up where the Architecture Tradeoff Analysis Method (ATAM) leaves off, adding costs, benefits, and schedule as attributes to be considered among the tradeoffs when a software system is being planned.

About the CBAM

The ATAM uncovers the architectural decisions that are made (or are being considered) for the system, and links these decisions to business goals and quality attributes. The CBAM builds on this foundation by enabling engineers to determine the costs and benefits associated with these decisions. Given this information, the stakeholders could then decide, for example, whether to use redundant hardware, checkpointing, or some other method to address concerns about the system's reliability. Or the stakeholders could choose to invest their finite resources in some other quality attribute—perhaps believing that higher performance will have a better benefit/cost ratio.

A system always has a limited budget for creation or upgrade; so every architectural choice, in some sense, competes with every other one for inclusion. The CBAM does not make decisions for the stakeholders; it simply helps them elicit and document costs, benefits, and uncertainty and gives them a rational decision-making process. This process is typically performed in two stages. The first stage is for triage, and the cost and benefit judgments used here are only rough estimates. The second stage operates on a much smaller set of scenarios and architectural decisions (also called architectural strategies), which are examined in greater detail.

Using the CBAM

The CBAM consists of six steps, each of which can be executed in the first (triage) and second (detailed examination) phases:

1. choose scenarios and architectural strategies
2. assess QA benefits
3. quantify the architectural benefits of the strategies
4. quantify the architectural costs and schedule implications of the strategies
5. calculate desirability
6. make decisions

In the first step, scenarios of concern to the system's stakeholders are chosen for scrutiny, and architectural strategies are designed that address these scenarios. For example, if there were a scenario that called for increased availability, then an architectural strategy might be proposed that added some redundancy and a failover capability to the system.

In the second and third steps, benefit information is elicited from the relevant stakeholders: QA benefits from managers (who, presumably, best understand the business implications of changing how the system operates and performs); and architectural strategy benefits from the architects (who, presumably, best understand the degree to which a strategy will, in fact, achieve a desired level of a quality attribute).

In the fourth step, cost and schedule information is elicited from the stakeholders. In step five these elicited values are used to calculate a desirability metric (a ratio of benefit divided by cost) for each architectural strategy. Furthermore, the inherent uncertainty in each of these values can be calculated, which aids in the final step, making decisions.

Given these six steps, the elicited values can be used as a basis for a rational decision-making process—one that includes not only the technical measures of an architectural strategy (which is what the ATAM produces) but also business measures that determine whether a particular change to the system will provide a sufficiently high return on investment.

An important feature of the next version of the CBAM will be the ability to perform multiple iterations, where each iteration adds some information and pares down the space of scenarios considered. For example, separate iterations will consider the side effects of ASs, and the correlation between ASs. Jai Asundi, one of the developers of CBAM, says, "If you don't have

the resources to do multiple iterations, you can do just one or two, and you'll still derive benefits."

Case Study

The Earth Observing System is a constellation of NASA satellites that gathers data about the Earth for the U. S. Global Change Research Program and other scientific communities worldwide. The Earth Observing System Data Information System (EOSDIS) Core System, also called the ECS, collects data from various satellite downlink stations for further processing. The ECS consists of about 1.2 million lines of code in 12,000 modules and about 50 customer off-the-shelf (COTS) products. The SEI and members of the ECS project performed a CBAM, demonstrating the feasibility of the method for large-scale projects. The CBAM helped structure an unstructured architecture design problem and offered the project manager a disciplined process aimed at arriving at a manageable set of architectural solutions to choose from.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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

Information Security Training and Education

Over the past two years, the number of security incidents reported to the CERT¨ Coordination Center has increased sharply. Incidents are occurring more frequently, and the resulting damage to systems and networks has been increasingly severe.

To help organizations protect themselves from and respond to network security threats, the SEI offers enterprise-wide training for organizations. The courses incorporate current trends and developments in network security and computer security incident response.

About the Courses

As opposed to technology-based point solutions, the courses approach information security, survivability, and risk from a broad perspective to provide a more comprehensive solution. Courses can be taken individually, or as part of a larger information security curriculum (see Figure 1).

"These courses address the need to increase the numbers of managers and technical staff trained to incorporate security practices," explains Barbara Laswell, manager of practices development and training. "Collaboration with strategic customers provides valuable real-world examples that drive development of the course content."

Incident Response

Five courses derive from the work of the CERT Coordination Center, and provide introductory and advanced training for technical staff and managers of computer security incident response teams (CSIRTs):

- Creating a Computer Security Incident Response Team provides a high-level overview of the key issues and decisions that must be addressed when establishing a CSIRT.
- Overview of Managing Computer Security Incident Response Teams provides insight into the type and nature of the work that CSIRT managers and staff may be expected to handle. It also provides an overview of the incident-handling arena, the Internet and CSIRT environment, intruder threats, organizational interactions, and the nature of incident response activities.
- Managing Computer Security Incident Response Teams provides current and future managers of CSIRTs with a practical view of the issues they will face in operating an effective incident response team.
- Fundamentals of Incident Handling is designed for CSIRT technical personnel with little or no incident-handling experience. Through interactive instruction and practical exercises, the

course provides insight into the type and nature of work that an incident handler typically performs.

- Advanced Incident Handling for Technical Staff is designed for CSIRT technical personnel with several months of incident-handling experience. Building on the methods and tools discussed in the fundamental course, this course focuses on practical exercises constructed around various incidents involving privileged compromises.

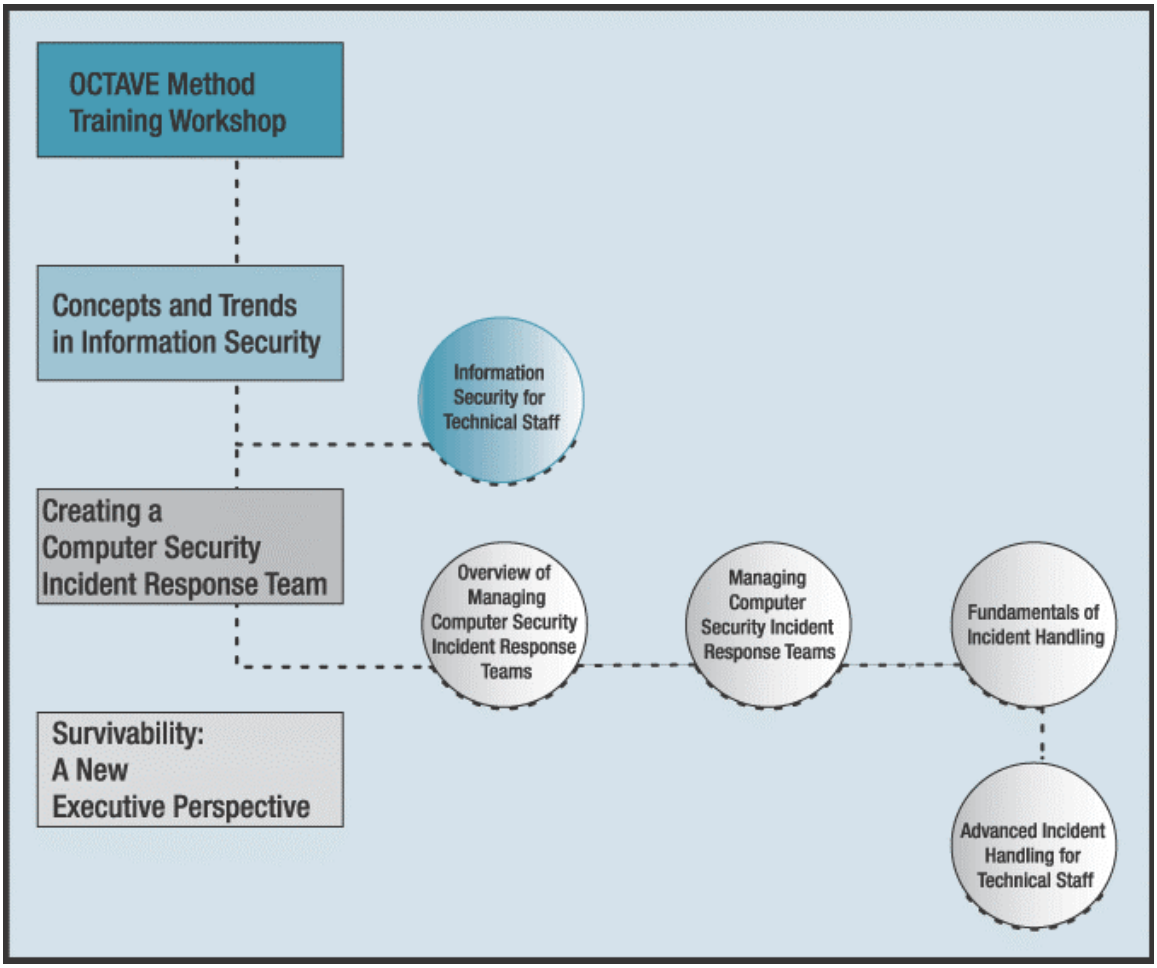

Figure 1: Information Security Curriculum

Broader Internet Security Issues

Three courses focus on broader Internet security issues designed to educate technical staff, policymakers, managers, and senior executives who are responsible for protecting information assets that are critical to their enterprise's mission.

- Concepts and Trends in Information Security provides an overview of security issues, techniques, and trends related to the confidentiality, integrity, and availability of information assets on an organization's computer systems.
- Information Security for Technical Staff provides attendees with practical techniques for protecting the security of an organization's information assets and resources. Security issues, technologies, and recommended practices are addressed at increasing layers of complexity, starting with data security and progressing to host system security, network security, and Internet security.
- Survivability: A New Executive Perspective provides participants with a foundation for understanding the activities and resources required to address the information survivability needs of an organization.

The NSS Program is also offering a new course in support of the Operationally Critical Threat, Asset, and Vulnerability Evaluation[SM] (OCTAVE[SM]) Method.

The OCTAVE Method Training Workshop is designed for interdisciplinary analysis teams that will lead and perform information security risk evaluations for their organizations. The workshop covers the OCTAVE Method, preparation for implementing the method, and guidelines for tailoring the method to meet an organization's unique needs.

As a result of implementing OCTAVE, enterprises may identify training needs related to the protection of critical information assets. For example, in one enterprise conducting an OCTAVE, the need arose to provide a common frame of reference for information security concepts across the organization. The NSS course Concepts and Trends in Information Security addressed this need.

Evolving to Meet Future Needs

To date, a variety of organizations from the United States and abroad have participated in the courses, including representatives from all of the critical infrastructure sectors. The SEI continues to work with strategic customers to create courses that serve the needs of the greater community.

Currently, the program is developing a Department of Defense-sponsored introductory level security and survivability course for system and network administrators. In addition, the program

is collaborating with partners in law enforcement and the academic community to develop a computer forensics workshop for managers and technical staff from industry, academic, and law enforcement organizations. There is an acute need for these sectors to work together to collect, analyze, and preserve artifacts as well as to develop digital forensics methods related to electronic crime. Laswell explains, “By transitioning best practices through our courses, we help organizations protect against today’s threats, mitigate future threats, and improve the information assurance posture of organizations and their networked systems.”

For more information, contact—

Kimberly Lang

Phone
412-268-9564

Email
klang@sei.cmu.edu

World Wide Web
http://www.cert.org/nav/index_gold.html

The Software Technology Review

Lauren Heinz

When Dan Ialenti's new job took him from the familiar realm of hardware to the uncharted regions of reusable software components, he needed a quick reference to get him up to speed on the latest software tools and practices.

Ialenti looked up the Software Technology Review (STR), a Web-based resource that features concise and informative summaries on emerging software technologies. Ialenti found descriptions on middleware and remote procedure call—two areas that he needed to evaluate to develop a technical reference model for a client. "The STR gave me the direct information that I needed to get a grasp on the various technologies occurring at the integration layer," said Ialenti, who is a systems engineer for an information resource center.

Engineers such as Ialenti are part of a growing number of software professionals who regularly access the STR (http://www.sei.cmu.edu/str/) for up-to-date information on current software technologies. While project managers might use it to evaluate software risk, maturity, and cost when selecting a new software system, other software professionals, including students, can use it to research new and related technologies. Originally targeted for a particular Department of Defense (DoD) audience, the STR has evolved to serve the diverse needs of both the DoD and commercial software engineering communities.

A Variety of Uses

The STR was first prototyped in 1997 when the Air Force acquisition community asked the SEI to create a reference document that would provide the Air Force with a clearer picture of software technologies. Contributors from the SEI and outside experts from organizations such as Lockheed Martin helped to populate the Web site and a paper version of the reference was released.

Today the STR serves the entire DoD acquisition community and is one of the most highly visited areas of the SEI Web site. An STR board was recently formed to revitalize and expand the site after some temporary funding shortages. During fiscal year 2001 it experienced nearly 2 million page hits, was viewed by more than 166,000 users in 139 countries, and had more than 18,000 documents downloaded from its Web pages.

While the STR's mission is to provide the DoD with a better understanding of software technologies that will enable it to systematically plan for the upgrade and evolution of current systems, as well as the development of new systems, it also serves commercial project managers and engineers looking for informative data on software for building or maintaining systems. Ialenti, for example, spends roughly four hours a week on the STR site, retrieving data on

software capabilities, storage, planning, and other issues for his team's projects. In addition, software professionals can contribute to the STR by adding descriptions of new technologies that they have investigated. In most cases, a software professional's existing documents can be easily reworked into the STR template. Thorough instructions and guidelines are available on the STR site for those interesting in submitting new technical descriptions and the STR board can help authors to fine-tune descriptions for proper placement.

Technical Descriptions

The STR currently features about 70 technology descriptions on a variety of topics, ranging from virus detection to network management to architecture description languages. STR descriptions are structured on a template that features a high-level summary of a software technology, an assessment of its maturity, usage considerations, costs and limitations, links to further information sources, and other valuable data. Technical descriptions also contain bibliographies so that users can access source data and further literature on their own.

In addition to a search feature, the STR also includes a taxonomy for navigation. This method is an effective way to lead users to a set of possible technologies that address their software problem area without having to read through every description. Using the taxonomies, users can search by a specific software quality measure (such as reliability), by a particular software use (such as design or testing), or by the ACM Computing Reviews Classification System, which categorizes subdisciplines within computer science.

The STR Board

John Goodenough, leader of the SEI's Performance Critical Systems Initiative, chairs the STR Board. He describes the STR as an authoritative source for evaluating a software technology's strengths, weaknesses, opportunities, and threats. While Goodenough says most online technology guides tend to focus on a technology's strengths, the STR provides a more balanced description of a technology's overall value. "It's not just where the technology is now but where it's going, how it's evolving," he said. "This difference provides an added benefit to managers who need to evaluate several technologies for their acquisition needs."

Goodenough and the ten-member STR board meet monthly to discuss new topics for the site and to review and update existing descriptions. The board is currently addressing ways to increase contributions and is considering implementation of a new navigation system.

Future Plans

Goodenough and the STR Board are working to build participation in the STR over the next several months in order to create a fuller, and more frequently updated resource. He encourages more software professionals from outside the SEI to contribute to the STR site by authoring, updating, or reviewing technology descriptions. If you would like to contribute a technical description to the STR, please see http://www.sei.cmu.edu/str/feedback/ or contact str@sei.cmu.edu.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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