Carnegie Mellon
Software Engineering Institute

news @ sei
interactive

Volume 4 | Number 3 | Third Quarter 2001

In This Issue

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

news@sei interactive

From the Director

The SEI was established to assist the overall strategic capabilities of the U.S. Department of Defense (DoD) by helping to improve and promote best practices within the software engineering community. The theme that we had chosen for the 2001 Software Engineering Symposium, "Acquiring the Strategic Edge," was intended to reflect our objective of enabling acquisition and development organizations to make measured improvements in their software engineering practices.

We had high hopes for a very successful symposium in Washington, D.C., on October 15-18. However, as a result of the catastrophic events of September 11, we ultimately concluded that we had to cancel our symposium. These events resulted in a number of changes to the day-to-day affairs of our entire country. Some of our speakers were no longer able to attend, including our main keynote speaker, General Lester L. Lyles of the US Air Force. We had also learned from a number of our colleagues that their organizations had been restricted from travel, and we concluded that the quality of the overall symposium program would have suffered without the participation of these key people from industry, government, and academia. I want to thank all those who worked very hard to make this year's symposium a success.

Along with the rest of our nation, we move forward from the events of September 11 with a renewed sense of purpose and a conviction that what we do at the SEI is important. Our CERT¨ Coordination Center, for example, has been on heightened alert and continues to provide an extremely valuable service in the defense of our country. Staying focused on what we do is the best way to help during our time of national crisis.

In light of the cancellation of our symposium, plans are now underway for the SEI to host a workshop for the acquisition community in our Arlington, Va. facility on January 22-24, 2002. This event will include presentations about the SEI technical program along with planning sessions with the DOD Software Collaborators (http://dodsis.rome.ittssc.com) and other appropriate DOD personnel. The event will provide us the opportunity to meet with and understand the DOD stakeholders' needs and to discuss transition and adoption strategies, success criteria, and outcome metrics with these stakeholders. We will announce this event on our Web site and communicate the results to those who would have attended the 2001 Software Engineering Symposium.

This planned event is consistent with a trend at the SEI over the past few years, in which 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, to be held for the first time in 2002 (http://wwwsel.iit.nrc.ca/iccbss)
- The annual Software Product Line Conference (http://www.sei.cmu.edu/SPLC2)
- The annual Information Survivability Workshop (http://www.cert.org/research/isw/isw2001)
- The DoD Collaborators' Workshops (http://dodsis.rome.ittssc.com/workshops.html)

We thank you for your help in improving software engineering practice through your participation in these events, and we wish you best regards in the coming year.

This issue's cover article is Software Product Line Practice Patterns. Software product line patterns, documented in a new book by SEI authors Paul Clements and Linda Northrop, offer a new way to help organizations adopt software product lines as a way of reducing time to market and increasing productivity. Other articles in this issue include Building Systems from Commercial Components, which describes some of the methods and techniques organizations can use to build systems from commercially available software components. Transitioning to CMMI describes plans for a transition strategy to help organizations implement CMMI-based process improvement. Finally, Using Easel to Study Complex Systems describes Easel, the SEI-developed modeling and simulation language that can be used to study systems with large numbers of interacting participants.

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

Stephen E. Cross
Director, Software Engineering Institute

Software Product Line Practice Patterns

More and more organizations today are taking a product line approach to software development and are achieving large-scale productivity gains, reduced time to market, and increased customer satisfaction.

To help organizations with their product line efforts, the SEI developed the Framework for Software Product Line Practice as well as supporting methods and training; sponsors and participates in conferences and workshops;[1] and created the Product Line Technical Probe.[2] Now the SEI offers a new aid to help organizations achieve their product line goals: software product line practice patterns.

About Product Line Practice Patterns

The Product Line Practice Framework identifies 29 practice areas whose mastery and application are necessary for success with software product lines. But organizations do not always know how to apply these practice areas or where to begin. SEI authors Paul Clements and Linda Northrop write in their recently published book, *Software Product Lines: Practices and Patterns*,[3] "You need to put the practice areas into play. You aren't going to attack all of the areas at once. You need to follow a divide-and-conquer strategy that permits you to divide the product line effort into chunks of work to be done."

The first thing an organization needs to do in this divide-and-conquer effort is understand its individual situation. Fortunately, although no two situations are exactly alike, similar situations occur again and again. It is through this similarity that product line practice patterns emerge. Patterns are a way of expressing common contexts and problem/solution pairs; they have been used effectively in many disciplines including architecture, economics, social science, and software design. For software product line practice patterns, the context is the organizational situation. The problem is what part of a software product line effort needs to be accomplished. The solution is the grouping of practice areas and the relations among those practice areas that together address the problem for that context. Clements and Northrop note that "patterns provide a helpful

---

1 "Advancing the State of Software Product Line Practice." news@sei 4, 1 (First Quarter 2001). *http://interactive.sei.cmu.edu/news@sei/features/2001/1q01/feature-5-1q01.htm*

2 "Probing Product Line Practices." news@sei 3, 2 (Spring 2000). http://interactive.sei.cmu.edu/news@sei/features/2000/spring/feature_4/feature_4.htm

3 Paul Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns*. Boston, Ma.: Addison-Wesley, 2001.

handle for selecting and applying the appropriate practice areas to meet your individual needs."

For product line patterns to succeed, they must be easy for a wide range of stakeholders to understand. They must have sufficient detail so that stakeholders can recognize the pattern context and problem and implement a solution. They must be consistent over the entire pattern set. They also must be able to handle static and dynamic information that will evolve as the product line develops. Following the lead of the "patterns community," the SEI has created a pattern template. It provides a standard way of expressing information:

- **Name**: A unique and intuitive pattern name and a short summary of the pattern.
- **Example**: One or more scenarios to help illustrate the context and the problem.
- **Context**: The organizational situations in which the pattern may apply.
- **Problem**: What part of a product line effort needs to be accomplished.
- **Solution**: The basis for the practice area pattern grouping underlying the pattern.
- **Static**: The grouping that lists the practice areas.
- **Dynamics**: A table, diagram, and/or scenario describing the relations among the practice areas in each group and/or among the groups if there is more than one.
- **Application**: Any suggested guidelines for applying the pattern.
- **Variants**: A brief description of known variants or specializations of the pattern.
- **Consequences**: The benefits the pattern provides and also any known limitations.

An Example Pattern

One of the early problems that new software product line organizations face is determining what products ought to be in their product line. The What to Build pattern is meant to assist them in this task. Determining what to build requires information related to the product area, technology, and market; the business justification; and the process for describing the set of products to be included in the product line. The following is a typical scenario suggesting the need for application of the What to Build pattern:

> A three-year old company develops Web-based registration systems for conferences. The company currently has contracts with three major computing conferences and these systems are in place. It has leads for another six conferences with the possibility for even more. The company is adopting a product line approach for its systems because there is considerable commonality among the three systems that have been fielded and because maintaining them separately has become a configuration nightmare. The company doesn't want to

> limit itself to the systems it has already developed, but it isn’t clear what other systems ought to be in the product line. On the other hand, it is a small organization in need of a target niche that is narrow enough to limit the complexity of the assets that need to be built. Because it is unclear what products to build, product line requirements cannot be generated, nor can the development of the other core assets begin. Application of the What to Build pattern focuses on just those five practice areas needed to address the problem at hand—namely, what products should be in the product line. It doesn’t tackle the whole product line approach. The application of other patterns would be necessary to complete the picture. However, the What to Build pattern provides a clear place to start. Using patterns an organization can move to software product lines in a manageable way that builds on the experience of others. The patterns are also now being incorporated into the SEI’s Product Line Technical Probe to assist in individual situations to narrow the probe focus, find the root cause of surfaced weaknesses, classify the results, and package and prioritize a course of action.

To date, the SEI has identified 22 practice area patterns. They are completely described in *Software Product Lines: Practices and Patterns,* along with a thorough description of the practice areas and multiple product line case studies. There are undoubtedly other product line practice patterns. Practitioners are encouraged to share their experiences and their own pattern creations.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

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

Building Systems from Commercial Components

Bob Lang

Increasingly, software systems are integrated from commercially available software components, including database management systems, middleware components, and domain-specific application components. The reasons for this are many, but include the need to reduce development costs, the need to reduce time to market, and the lack of technical expertise.

The development of systems from commercial components, however, introduces uncertainty and complexity into the software engineering process. Traditional software engineering methods and processes have proven inadequate and must be modified to face the new challenges imposed by the use of commercial components. In particular, component capabilities and liabilities are a principle source of design constraint in system development.

The SEI has used its experience with customers developing large complex systems from commercial components to develop a collection of methods and techniques that address these new challenges. SEI authors Kurt Wallnau, Scott Hissam, and Robert Seacord document these methods and techniques, as well as an extensive case study drawn from customer engagements, in a new SEI Series book from Addison-Wesley entitled *Building Systems from Commercial Components*.[1] These methods and techniques are practical applications of software engineering that have been successfully applied and reapplied in the development of large, complex systems," says Seacord.

Contingency, Model Problems, and Evaluation

Building systems from commercial components requires a significantly different development process than custom development because of the uncertainty and risks of using commercial components. These risks often must be mitigated by simultaneously pursuing multiple design contingencies. While one design contingency may be considered the principal contingency (and therefore receive a greater share of development resources) other, alternate design approaches may be simultaneously pursued. The number and characteristics of these design alternatives vary with the characteristics of the project.

[1] Kurt Wallnau, Scott Hissam, and Robert Seacord. Building Systems from Commercial Components. Boston, Ma.: Addison-Wesley, 2001. http://www.sei.cmu.edu/cbs/bscc/bscc.htm

For example, a project for which the greatest risk is time to market may select components whose characteristics and interactions are well understood. Another project may choose to use newer components that are not as well understood to incorporate newer technologies.

Within each contingency, it is important to determine the feasibility of component interactions required to achieve the desired capabilities of the system. The authors of *Building Systems from Commercial Components* suggest the use of model problems—focused experiments that can be used to answer specific design questions. These model problems are intended to lead to a determination of design feasibility.

Systems can be built using any number of commercial components. For example, systems that do not include any commercial components are considered “custom built” and are outside the scope of the book. Systems can also be built around a single commercial component, such as SAP or other large enterprise resource planning (ERP) system.

In these cases, the design of the system often becomes an exercise in customizing the commercial component within the guidelines allowed by the product. The final possibility is that systems are built from an ensemble of components. This last case is the principle focus of *Building Systems from Commercial Components*.

Even when systems are constructed using multiple components, a typical practice is to evaluate each component in isolation. For example, a project may enumerate a list of components required to implement a system and then assign different teams to evaluate and select a product for each required component. While selecting “best-of-breed” components has obvious attractions, these components often prove incompatible when combined in an ensemble. Also, components have unanticipated interactions when combined in a single system. When building a system consisting of multiple components, component ensembles and not individual components must be evaluated to ensure the feasibility of the design approach.

The feasibility of the design approach can in turn be used to determine the feasibility of a design contingency.

These processes have been applied successfully in the development of a distributed image retrieval system (as documented in the SEI Series book) and in the development of various commercial and government systems since completion of the original case study. For example, the use of model problems in the Air Force logistics domain is documented in the SEI technical report *Maintaining Transactional Context: A Model Problem* (available on the SEI Web site at http://www.sei.cmu.edu/publications/documents/01.reports/01tr012.html).

Transition Efforts

Seacord is leading continuing SEI efforts to transition the practices described in *Building Systems from Commercial Components* into practice. The foremost transition vehicle is the SEI Series book itself, but additional products under development include an undergraduate software engineering course incorporating methods and techniques from the book.

The techniques described in the book are also being applied in the development of the knowledge-based automated component ensemble evaluation (K-BACEE) prototype at the SEI. K-BACEE includes a component repository and knowledge base of component integration rules. System integrators can use the tool by creating a manifest, which defines their systems requirements, the system context, and the architecture for the new system. The system then identifies ensembles of components that can be used to realize the manifest. K-BACEE automatically ranks the ensembles based on component compatibility using the integration rules knowledge base. While techniques from the book are being used in its development, K-BACEE is unique in that it also automates portions of the component search and evaluation processes prescribed by the book.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

World Wide Web
http://www.sei.cmu.edu/cbs/bscc/bscc.htm

Transitioning to CMMI™

Bob Lang

When a new software product, process, or technology is introduced into an organization, significant non-technical changes need to occur. Most technical managers, software engineers, and information technology specialists can address the technological problems but need guidance for the change process.

Through workshops and other collaborative efforts with early adopters, the SEI is developing a transition strategy and supporting materials to help organizations successfully implement Capability Maturity Model Integration℠ (CMMI℠).

Background

The CMM Integration project was formed to address the problems some organizations were having with using multiple Capability Maturity Models. The purpose of CMMI is to provide guidance for improving an organization's processes and its ability to manage the development, acquisition, and maintenance of products and services. CMM Integration places proven practices into a structure that helps an organization assess its organizational maturity and process area capability, establish priorities for improvement, and guide the implementation of these improvements.

In addition to the CMMI models released in August and December 2000, the CMMI product suite includes

- assessment products, such as Appraisal Requirements for CMMI (ARC) V1.0, and Standard CMMI Appraisal Method for Process Improvement (SCAMPI℠) method description
- courses, including introductory courses, intermediate courses, and SCAMPI lead appraiser training

With these products in place, many organizations have begun using CMMI as a basis for their process improvement efforts. In collaboration with these organizations, the SEI is now working to develop a transition strategy to help other organizations adopt CMMI.

"We are approaching this from several angles," says Mike Phillips, CMMI program manager. "We're developing strategies for organizations based on proven approaches, but we're also providing support in the form of workshops, technical reports, and other support materials."

Workshops

The SEI's Technology Transition Workshop Series invites members of the software engineering community to share lessons learned about, and receive recognition for, successful software technology transition.

The first workshop in the series, "The Road to CMMI: What Works, What's Needed?" was held on May 30, 31, and June 1, 2001 in Pittsburgh. Participants explored successful practices for accelerating an organization's transition to the CMMI product suite.

During the workshop, participants discussed their experiences implementing CMMI at their organizations. The group identified and evaluated more than 60 best practices for adopting the product suite, 40 mechanisms that they felt were needed, and 30 traps and timewasters. These findings are expected to enable future adopters to make more effective technology transitions, as well as target some problem areas for the larger CMMI community to address. The results of the workshop will be published as an SEI technical report.

A Guide For Executives

In addition to the technical report, the results of the workshop are being used to develop a first draft of Transitioning to CMMI: A Guide for Executives.

The guide for executives does not recommend a particular transition approach but rather helps an executive build a case for why his or her organization should use CMMI. The guide for executives rests on the assumption that the most critical element of any implementation is the leadership element. Organizational change must be designed, implemented, and led from the top for the following reasons:

- Competing alternative solutions result in fragmented efforts rather than integrated answers.
- Resources must be committed for the process improvement effort.
- A leader is needed to establish a mentoring environment for process improvement, and reward process improvement efforts.
- The leader's behavior is watched and emulated.
- A leader is required to establish and maintain the vision. Because of the importance of a leader, this guide is an important element in a transition strategy and contains the key information an executive will need to help his or her organization start down the path of CMMI-based improvement.

Other Transition Mechanisms

As part of a more a tailored transition strategy, future documents will help guide organizations through specific scenarios as they look to move from their current process improvement efforts to a CMMI-based approach. For example, organizations may be

- using the SW-CMM as a basis for a process improvement initiative
- using the Systems Engineering Capability Model (SE-CM) as a basis for a process improvement initiative
- unfamiliar with model-related process improvement|

A tailored transition strategy will make allowances for these and other factors such as the role of their external stakeholders, the process improvement areas they want to focus on, and whether the organization is a DoD contractor, government agency, or commercial organization. As it has throughout the CMMI development process, collaboration will play an important role here: Phillips says, "What we'd like to see is that organizations will say, 'I've got something that fits in here,' and start looking for ways of coupling what people are already doing with what the SEI has done. Because while we think we've got some pretty good insights into what aids transition, real products and strategies that have been put in place by an organization inherently have some advantages."

Efforts are also are underway to update existing SEI technical reports. For example, technical reports on forming a software engineering process group (SEPG); on measurement and team formation; and on the IDEALSM model are all valuable resources that can easily be adapted to CMMI.

Another resource for those interested in learning more about CMMI and how it has been used by organizations working on integrated process improvement is the book, *CMMI Distilled: A Practical Introduction to Integrated Process Improvement*, part of the Addison-Wesley SEI series in software engineering. For more information on this book see the Addison-Wesley Web site: http://cseng.aw.com/book/0,,0201735008,00.html.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

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

Using Easel to Study Complex Systems

Matt DeSantis

Simulation has long been a popular application within many computing environments, mainly because computers are quite good at handling repeating processes, often defined by what programmers or researchers can readily see. The biological sciences have evolution simulators, entertainment software companies sell simulations of the entire planet, computers can simulate other computer platforms. Most of these types of simulations proceed with known, assumed, or hypothesized information. But how do you simulate processes that are unobserved, ill defined, or virtually invisible?

The SEI is developing a general-purpose modeling and simulation language and tool called Easel that can be used to predict behavior in a seemingly uncertain world.

About Easel

Easel can be used to simulate systems in which there are large numbers of interacting participants. In the real world, our critical infrastructures exhibit such properties: the Internet, electric power grids, telephone systems, the stock market, and emergency response systems all have large numbers of interacting participants that, individually, have limited visibility or knowledge of the larger system that surrounds them. It is often not possible to predict the viability of a system solely on the behavior of its individual participants. Easel helps to predict the interactions of systems, each with many actors.

Traditional security approaches that model systems with a small number of actors are inadequate to protect large scale, highly distributed systems—let alone “systems of systems.” Lead developer David Fisher of the SEI says, “There’s a great difference between simulating systems by using 10 or 20 actors and using 100 or more. You need support for a large number of actors to have any relevance or application to the real world.” A typical Easel application handles thousands of individual actors—the alpha version supports 75,000 actors, and future releases will support 100,000.

From the start, Easel was designed with flexibility and modularity in mind. Easel allows programmers or researchers to tailor the environment based on the scope of a particular project. The Easel language has been extended to support more commands. Its graphics subsystem supports multiple simultaneous views of the simulation as it plays out, from perspectives defined by the programmer. “This past summer, we set out to improve the alpha version by three measures: robustness, performance, and diagnostics," says Fisher. Plans are also in place to offer Easel to selected customers outside the SEI.

Potential Uses for Easel

At the core of Easel's design lies the concept of survivability. A survivable system is one that fulfills its mission, in a timely manner, in the presence of attacks, failures, or accidents. At the CERT® Coordination Center, Easel is being used to study the response of networks to attacks and attack mitigation strategies. For example, one Easel demonstration predicts the effectiveness of software patching during a widespread computer virus event. As the simulation proceeds, it is possible to investigate what critical factors determine the outcome.

Easel can also model and monitor processes inherent in software development. The flow of artifacts through the software development cycle, for example, can be readily simulated with ordinary simulation packages. But Easel can simulate processes where interactions between actors have not been specifically defined. An Easel simulation can proceed in the presence of partial and imprecise information. For example, in a hypothetical traffic routing scenario, the researcher may not know exactly how many vehicles a particular road could realistically support in the face of extreme traffic congestion.

Within the Easel environment, such a scenario proceeds, not only providing information about the system, but also arming the researcher with a better hypothesis about the unknown pieces within that system. In this respect, Easel can simulate a larger set of less deterministic processes.

A Real World Application

One of the first complete Easel applications, a simulation for the Defense Advanced Research Projects Agency (DARPA), illustrates an emergent algorithm for location-independent IP routing within a survivable routing infrastructure. Another working demo depicts the coordinated movement of transportation vehicles over the infrastructure of a large city. Yet another simulates an emergency response scenario, in which ambulances carry patients to an array of hospitals.

Easel offers the potential to simulate the complex interdependencies at work in society. The importance to military planners resides in the interdependence of Department of Defense operations with that of private industry and critical infrastructure. By hosting “what-if” simulations and facilitating the study of cascade effects, Easel can expand understanding of information security and survivability for both critical infrastructure providers and the larger community.

Easel developers in the Networked Survivable Systems program welcome interested parties to the Easel project. Those who work in infrastructure assurance, particularly those with extensive knowledge of critical infrastructure systems, are encouraged to collaborate.

For more information, contact—

Customer Relations

Phone
412 / 268-5800

Email
customer-relations@sei.cmu.edu

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

Everyone’s a System Administrator

Lawrence R. Rogers

If your computer is connected to the Internet, you are a systems administrator, and you should take precautions just like a professional sys admin would take. That’s especially true if you want to avoid having your computer used in a distributed denial-of-service attack—and then getting sued by a corporation that was “attacked” by your computer.

Everybody who has a home computer is a system administrator—especially those who are connected to the Internet via cable modem or digital subscriber line (DSL) connections. Home computer owners have the same responsibilities—even if they don't accept them—as the professionals who take care of the computer systems at work. Home computer owners who don't take responsibility may change their perspective on security when their computer systems are used in a distributed denial-of-service attack against an organization that can afford to go after all computer systems used in the attack.

You've just purchased a state-of-the art and top-of-the-line personal computer system and you're running the latest version of your favorite operating system. To give yourself the highest speed Internet access available, you've chosen the always-on technology of a cable modem. You are ready to do some serious computing in your home. Let's go to it!

After a few weeks of enjoying your new system and your very fast Internet connection, you notice that the connection isn't so fast anymore. In fact, when you aren't doing anything on the system, you notice that the transmit light on your cable modem is on solidly. You poke around a little (or ask your child or the teenager down the street to poke around) and see some programs running that you don't recognize. With a little tinkering, you kill them off and are pleased to see that the modem's transmit light is taking a rest.

A few days later, the event repeats itself, and you counter with the same techniques that worked before. You stop the problem again, but you get a sinking feeling that you'll have to do this over and over again. Feeling a bit nervous, you look around for damage. Your applications still work and your bank account balance looks about right. That's a relief. You decide the problem is solved and you move on.

That day in the paper, you read about some high-profile attacks on well-known e-commerce sites. You learn that the sites that were attacked have suffered significant financial losses. They intend to go after the owners of the computer systems used in the attack. You think to yourself, "Corporations have deep pockets. They can afford to pay for their inability to keep hackers out of their computer systems. Serves 'em right!"

Soon after, you receive in the mail an official-looking document from an attorney’s office. Upon opening it, you find "legalese" describing a suit filed on behalf of one of those e-commerce sites you read about in the paper. You find that you and your computer system are listed as one of the systems against which the suit has been filed. Whoa! Corporations may have deep pockets, but you don’t, especially after just having spent your extra cash on that new computer system. Now it seems that you’ll need to spend even more money for legal services to defend yourself.

Could this happen to you? Yes, it could; and the fact that it hasn’t happened yet doesn’t mean that it never will. I firmly believe that the time will come when an e-commerce organization like the one mentioned above will seek compensation because you neglected the standards of due care and, thus, caused their loss. It’s a matter of when, not if.

Still not convinced that it could happen to you? Think about it another way. What is the difference between the computer system in your office and the brand-new system at home? Not a lot, except that within the corporate setting, there is almost always a group of employees who have administrative responsibilities for the care and well being of those computer systems. For the computer at home, you have that responsibility, whether you choose to accept it or not.

OK, so what if your machine doesn’t have an administrator? After all, you believe that there is nothing on your home computer system that would be of interest to an intruder, right?

Guess again. That system has all the features needed to participate in one of those popular distributed denial-of-service attacks that, unfortunately, characterize the Internet these days. Your new machine has lots of power, plenty of disk space, a lot of memory, and a high-speed and always-on Internet connection. Most importantly, its owner (you) is probably not looking very closely at how the system is being used and potentially abused. It’s a perfect target. Yesterday, you couldn’t spell systems administrator. Now you are one!

What does it mean to be a systems administrator for your home computer system? It means many things, including patching software, installing a firewall, using a virus checker, and keeping up to date about what’s happening on the Internet.

At the CERT® Coordination Center, we have learned that over 95% of all network intrusions could be avoided by keeping your computer systems up to date with patches from your operating system and applications vendors. If you do nothing else, you should install these patches wherever possible, and as quickly as possible.

Unfortunately, applying patches is often a hard, time-consuming task. Vendors don't always tell you whether their products will continue to work when patched. When you're not sure if you can apply a patch without repercussions, contact your vendor and ask. As more customers ask these questions, the more likely it is that the vendors will make their products work on patched systems—and publicize their efforts.

What else should you do? Your car has a physical firewall that sits between you and the engine compartment. Its purpose is to keep the bad things that can happen to and around your engine out of your lap. Your computer system ought to have a firewall too, a technological firewall. With a technological firewall, you can keep the intruders out of your lap.

There are many brands of firewalls, and they come in two basic varieties—hardware and software. The hardware firewall attaches directly to your cable modem or DSL connection, and your computer system plugs into the firewall. In 2001, they cost about $200. The software firewall is nothing more than an application that installs directly on your computer system. You can purchase them at prices of $20 and up, but there are good ones that are free. Do some research to see which firewalls meet your needs. While you're at it, consider getting one of each, especially if your home computer system is a laptop that may be attached to other networks besides the one at home. No matter where you connect that laptop to the Internet, you will have a firewall standing between you and—literally—the rest of the world.

Viruses and worms have a significant impact on computer systems. You should invest in anti-virus software and then be sure to keep the virus signatures file up to date. Most anti-virus software makes this job easy by automating the task. Money spent here is money well spent.

Finally, you need to keep up with the security issues surrounding your computer system and its applications. We suggest that you subscribe to the electronic mailing lists that are relevant to you. You need to know when there are patches, improvements, and new versions that have security implications for you.

Given the present state of technology, computer systems need attention—and lots of it—to keep them operating more securely. For your home computer systems, you are the person who has the responsibility to give that attention. You need to accept it and do what the professional systems administrators do.

In case you didn't know this already, when you are connected to the Internet, the Internet is connected to you. You need to be ready.

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 ten years at Princeton University, first in the Department of Computer Science on the Massive Memory Machine project, and later at the Department of Computing and Information Technology (CIT). While at CIT, he directed and managed the UNIX Systems Group, which was charged with administering the UNIX computing facilities used for undergraduate education and campus-wide services.

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.

Economic Modeling of Software Architectures

Rick Kazman, Jai Asundi, Mark Klein

The Cost-Benefit Analysis Method (CBAM) picks up where the Architecture Tradeoff Analysis Method (ATAM) leaves off: adding cost as an attribute to be considered among the tradeoffs when a software system is being planned.

Introduction

At the Software Engineering Institute, we have been doing analyses of software and system architectures, using the Software Architecture Analysis Method (SAAM) and the Architecture Tradeoff Analysis Method (ATAM), for more than five years. [For more on these subjects, see http://www.sei.cmu.edu/ata/ata_init.html.] When we do these analyses, we are primarily investigating how well the architecture has been designed with respect to its quality attributes (QAs): modifiability, performance, availability, usability, and so forth. In the ATAM, we additionally focus on analyzing architectural tradeoffs, the points where a decision might have consequences for several QA concerns simultaneously.

But the biggest tradeoffs in large, complex systems always have to do with economics: How should an organization invest its resources in a manner that will maximize its gains and minimize its risks? This question has received little attention in the software engineering literature, and where it has been addressed the attention has primarily focused on costs. Even in those cases, the costs were primarily the costs of building the system in the first place, and not its long-term costs through cycles of maintenance and upgrade. Just as important as costs are the *benefits* that an architectural decision may or may not bring to an organization. Given that resources for building and maintaining a system are finite, there must be some rational process for choosing among architectural options, both during initial design and subsequent periods of upgrade. These options will have different costs; will implement different features, each of which brings some benefit to the organization; and will have some inherent risk or uncertainty. Thus we need economic models of software that take into account costs, benefits, and risks.

The CBAM

For this reason, we have been developing a method for economic modeling of software and systems, centered on an analysis of their architectures. We call this method the Cost Benefit Analysis Method (CBAM). The CBAM builds on the ATAM to model the costs and benefits of architectural design decisions and to provide a means of optimizing such decisions. A simple way to think about the objectives of this method is that we are adding money to the ATAM as an additional attribute to be traded off. We are showing how to make decisions in terms of benefits per dollars, as well as in terms of quality-attribute responses.

The CBAM begins where an ATAM leaves off and depends on the artifacts that the ATAM produces as output, as depicted in Figure 1.

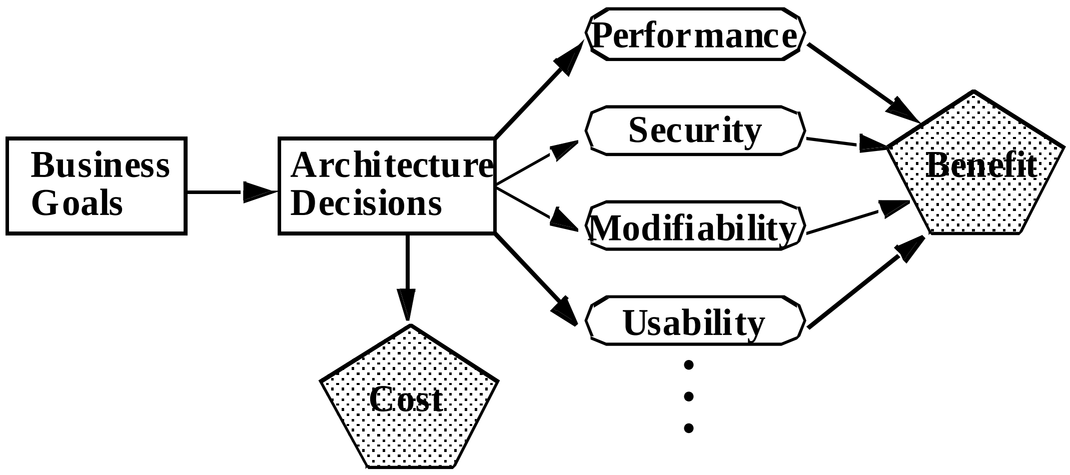

Figure 1: The Context for the CBAM

The ATAM uncovers the architectural decisions that are made (or are being considered) for the system and links these decisions to business goals and QA response measures via a set of elicited scenarios. The CBAM builds on this foundation, as shown by the shaded pentagons in Figure 1, 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 QA—perhaps believing that higher performance will have a better benefit/cost ratio.

A system always has a limited budget for creation or upgrade, and 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 elicited cost and benefit judgments are only estimates. The second stage operates on a much smaller set of architectural decisions (also called architectural strategies), which are examined in greater detail.

There is uncertainty involved with the design of any large, complex system with many stakeholders. The uncertainty comes from three relationships:

1. the uncertainty of understanding how architectural decisions relate to QA responses. That is to say, even if we are diligent in designing and analyzing our architecture, there is some uncertainty in knowing how well it will perform, adapt to change, or be secure, and there is uncertainty in understanding the environment in which the architecture will operate (e.g., knowing the distribution of service requests arriving at the system).
2. the uncertainty of understanding how architectural decisions relate to cost. Cost modeling is not precise, and the best models only provide a range of cost values.
3. the uncertainty of understanding how QA responses relate to benefits. Even with perfect knowledge of an architecture's responses to its stimuli and the distribution of these stimuli, it is still unclear in most cases how much benefit the organization will actually accrue from such a system.

As with the financial markets, different investments will appeal more or less to different stakeholders depending on those investments' inherent uncertainty. One function of the CBAM, then, is to elicit and record this uncertainty, because it will affect the decision-making process.

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 strategies' benefits
4. quantify the architectural strategies' costs and schedule implications
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, we elicit benefit information 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, we elicit cost and schedule information from the stakeholders. We have no special technique for this elicitation; we assume that some method of estimating costs and schedule already exists within the organization. Based on these elicited values, in step 5 we can calculate a desirability metric (a ratio of benefit divided by cost) for each architectural strategy. Furthermore, we can calculate the inherent uncertainty in each of these values, which aids in the final step, making decisions.

Given these six steps, we can use the elicited values 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.

For more information on the CBAM, including a case study of how it was applied to NASA's ECS project, see: R. Kazman, J. Asundi, M. Klein, "Quantifying the Costs and Benefits of Architectural Decisions", *Proceedings of the 23rd International Conference on Software Engineering (ICSE 23)*, (Toronto, Canada), May 2001, 297-306.[1]

About the Authors

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".

Jai Asundi is a Visiting Scientist at the SEI with the Product Line Practice Program. His interests are in the area of economics driven software engineering, decision analysis for

[1] This paper may be retrieved on-line at: http://ieeexplore.ieee.org/iel5/7340/19875/00919103.pdf?isNumber=19875

software systems, open-source software systems and outsourced software development. He has a Ph.D. in Engineering and Public Policy from Carnegie Mellon University.

Mark Klein is a Senior Member of the Technical Staff at the SEI. He has over 20 years of experience in research on software engineering, dependable real-time systems and numerical methods. Klein's most recent work focuses on the analysis of software architectures, architecture tradeoff analysis, attribute-driven architectural design and scheduling theory. He is co-leader of the SEI's work in the area of attribute-based design primitives. Klein has co-authored many papers and is co-author of two books including "Practitioner's Guide for Real-Time Analysis", and of "Evaluating Software Architectures: Methods and Case Studies".

Design and Search

Robert C. Seacord

Building systems from commercial components is often a completely different experience from building custom systems in that the focus of control shifts from the architect or designer to the commercial marketplace. To deal with this loss of control, the successful architect must take a risk-driven approach by considering multiple design contingencies, weighing benefit against risk, and generally playing the odds.

Introduction

Herbert A. Simon, winner of the 1978 Nobel Prize in Economics and many prestigious international scientific awards for his work in cognitive psychology and computer science wrote the following in the *Science of Design* in 1969:

> *Design procedures in the real world do not merely assemble problem solutions from components but must search for appropriate assemblies.*
>
> *... In carrying out such a search, it is often efficient to divide one's eggs among a number of baskets—that is, not to follow out one line until it succeeds completely or fails definitively, but to begin to explore several tentative paths, continuing to pursue a few that look most promising at a given moment. If one of the active paths begins to look less promising, it may be replaced by another that had previously been assigned a lower priority.*

Some 30 years later, we are saying almost the same thing about building systems from commercial components, namely that system architects and designers must pursue multiple simultaneous design contingencies to balance and reduce risk. While this is not a typical tenet of software engineering, it applies in the case of commercial components for a simple reason. Software design does not take place in the real world but consists instead of the formulation of an artificial world of abstractions that may or may not model real-world properties. However, the use of commercial components introduces a substantial dose of reality into the design equation.

Design as Search

When I say software design does not take place in the real world, I am not ignoring the realities of cost, schedule, people and other very tangible real-world issues. Building systems from commercial components, however, adds a new layer of reality on top of these already substantive issues. In particular, component capabilities and liabilities are a principle source of design constraint in system development.

Design of component-based systems consists of a search for compatible *ensembles* of commercial components that come closest to meeting system objectives. Because component integration is a principal risk area, the system architect must determine if it is feasible to integrate the components in each ensemble, and in particular, to evaluate whether an ensemble can support a required interaction of the system.

In effect, each ensemble amounts to a continued path of exploration. This exploration should initially focus on the feasibility of the path to make sure that there are no precipitous cliffs, uncrossable chasms, bands of thieves, or other obstacles that would prevent the journey being completed by the full development team (who perhaps are not quite so nimble as the explorer).

If only one path (that is, ensemble) is discovered, then the question of the optimal path does not arise. However, if more than one path is discovered, a decision is required. An architect or a design team can often make a decision when there is one (or at most a few) major criteria that weigh the evaluation in favor of a particular ensemble. When there are numerous criteria and the ensembles each have their own advantages and disadvantages, it may be necessary to take a more formal approach to decision making.

Fortunately, there are a number of multi-criteria evaluation techniques available that apply some science to the field of decision making. These techniques are often misused in the evaluation of individual components during the *formulative* phase of system development.

The formulative phase occurs before many decisions have been made. At this time, the availability of commercial components may cause revisions in user requirements, which in turn may affect decisions about commercial components. As components are identified, and an understanding of the features and interactions of these components are discovered, the formation of the design can vary tremendously. While multi-criteria evaluation techniques can be—and are—used at this time to evaluate products that satisfy the requirements of a particular component in the system, this evaluation is often premature and costly, because the role this component fills in the formulation of the design is not yet properly understood.

Evaluation at this time can also have a tremendous adverse effect in that a component selection can easily become a keystone of the design. In selecting additional components, it may be discovered that less-than-adequate components must be selected to ensure compatibility with the initial selection. Because these decisions are often difficult to revisit, the project becomes wedded to a flawed and non-optimal assembly of components.

The evaluation context therefore is more properly scoped at the evaluation of ensembles of components that achieve the design objectives and not at a single component. This often means that evaluation decisions are deferred to a later stage of the development process, which brings us back to the need to maintain and manage multiple design contingencies.

Summary

Building systems from commercial components is often a completely different experience from building custom systems in that the focus of control shifts from the architect or designer to the commercial marketplace. To deal with this loss of control, the successful architect must take a risk-driven approach by considering multiple design contingencies, weighing benefit against risk, and generally playing the odds.

In the book *Building Systems from Commercial Components,* in the SEI Series in Software Engineering, we have documented a number of techniques that have been applied in practice to manage this loss of control and maximize opportunities for success. We have also illustrated these techniques in an extensive case study involving the development of a Web-based application for image management and retrieval.

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.

The Future of Software Engineering: Part III

Watts S. Humphrey

In the previous two columns, I began a series of observations on the future of software engineering. The first two columns covered trends in application programming and the implications of these trends. The principal focus was on quality and staff availability. In this column, I explore trends in systems programming, including the nature of the systems programming business. By necessity, this must also cover trends in computing systems.

The Objectives of Systems Programs

The reason we need systems programs (or operating systems) is to provide users with virtual computing environments that are private, capable, high performance, reliable, usable, stable, and secure. The systems programming job has grown progressively more complex over the years. These programs must now provide capabilities for multi-processing, multi-programming, distributed processing, interactive computing, continuous operation, dynamic recovery, security, usability, shared data, cooperative computing, and much more.

Because of the expense of developing and supporting these systems, it has been necessary for each systems program to support many different customers, a range of system configurations, and often even several system types. In addition, for systems programs to be widely useful, they must provide all these services for every application program to be run on the computing system, and they must continue to support these applications even as the systems programs are enhanced and extended. Ideally, users should be able to install a new version of the systems program and have all of their existing applications continue to function without change.

Early Trends in Systems Programs

At Massachusetts Institute of Technology (MIT), where I wrote my first program for the Whirlwind Computer in 1953, we had only rudimentary programming support [Humphrey].[1] The staff at the MIT computing center had just installed a symbolic assembler that provided relative addressing, so we did not have to write for absolute memory locations. However, we did have to program the I/O and CRT display one

[1] I was a computer systems architect at Sylvania Electric Products in Boston at the time.

character at a time. Whirlwind would run only one program at a time, and it didn’t even have a job queue, so everything stopped between jobs.

Over the next 10 years, the design of both computing machines and operating systems evolved together. There were frequent tradeoffs between machine capabilities and software functions. By the time the IBM 360 system architecture was established in 1963, many functions that had been provided by software were incorporated into the hardware. These included memory, job, data, and device management, as well as I/O channels, device controllers, and hardware interrupt systems. Computer designers even used micro-programmed machine instructions to emulate other computer types.

Microprogramming was considered hardware because it was inside the computer’s instruction set, while software was outside because it used the instruction set. While software generally had no visibility inside the machine, there were exceptions. For example, systems programs used privileged memory locations for startup, machine diagnostics, recovery, and interrupt handling. These capabilities were not available to applications programs.

While the 360 architecture essentially froze the border between the hardware and the software, it was a temporary freeze and, over the next few years, system designers moved many software functions into the hardware. Up to this point, the systems programs and the computer equipment had been developed within the same company. Therefore, as the technology evolved, it was possible to make functional tradeoffs between the hardware and the software to re-optimize system cost and performance.

One example was the insertion of virtual memory into the 360 architecture, which resulted in the 370 systems [Denning].[1] Another example was the reduced instruction set computer (RISC) architecture devised by John Cocke, George Radin, and others at IBM research [Colwell]. Both of these advances involved major realignments of function between the hardware and the software, and they both resulted in substantial system improvements.

With the advent of IBM’s personal computer (PC) in 1981, the operating system and computer were separated, with different organizations handling the design and development of hardware and software. This froze the tradeoff between the two, and there has since been little or no movement. Think of it! In spite of the unbelievable advances in hardware technology, the architecture of PC systems has been frozen for 20 years. Moore’s law says that the number of semiconductors on a chip doubles every 18 months, or 10 times in five years. Thus, we can now have 10,000 times more

[1] At this time I was managing the systems software and computer architecture groups at IBM.

semiconductors on a single chip than we could when the PC architecture was originally defined.

Unfortunately this architectural freeze means that software continues to provide many functions that hardware could handle more rapidly and economically. The best example I can think of is the simple task of turning systems on and off. Technologically speaking, the standalone operating system business is an anachronism. However, because of the enormous investments in the current business structure, change will be slow, as well as contentious and painful.

The Operating Systems Business

Another interesting aspect of the operating systems business is that the suppliers' objectives are directly counter to their user's interests. The users need a stable, reliable, fast, and efficient operating system. Above all, the system must have a fixed and well-known application programming interface (API) so that many people can write applications to run on the system. Each new application will then enhance the system's capabilities and progressively add user value without changing the operating system or generating any operating system revenue. Obviously, to reach a broad range of initial users, the operating systems suppliers must support this objective, or at least appear to support it.

The suppliers' principal objective is to make money. However, the problem is that programs do not wear out, rot, or otherwise deteriorate. Once you have a working operating system, you have no reason to get another one as long as the one you have is stable, reliable, fast, and efficient and provides the functions you need. While users generally resist changing operating systems, they might decide to buy a new one for any of four reasons.

1. They are new computer users.

2. They need to replace their current computers and either the operating system they have will not run on the new computer or they can't buy a new computer without getting a new operating system.

3. They need a new version that fixes the defects in the old one.

4. They need functions that the new operating system provides and that they cannot get with the old system.

To make money, operating systems suppliers must regularly sell new copies of their system. So, once they have run out of new users, their only avenue for growth is to make the existing system obsolete. There are three ways to do this.

5. Somehow tie the operating system to the specific computer on which it is initially installed. This will prevent users from moving their existing operating systems to new computers. Once the suppliers have done this, every new machine must come with a new copy of the operating system. While this is a valid tactic, it is tantamount to declaring that the operating systems business is part of the hardware business.

6. Find defects or problems in the old version and fix them only in the new version. This is a self-limiting strategy, but its usefulness can be prolonged by having new versions introduce as many or more defects as it fixes, thus creating a continuing need for replacements. The recent Microsoft ad claiming that "Windows 2000 Professional is up to 30% faster and 13 times more reliable than Windows 98," looks like such a strategy, but I suspect it is just misguided advertising [WSJ]. The advertising community hasn't yet learned what the automotive industry learned long ago: never say anything negative about last year's model.

7. Offer desirable new functions with the new version and ensure that these functions cannot be obtained by enhancing the old version. This is an attractive but self-limiting strategy. As each new function is added, the most important user needs are satisfied first so each new function is less and less important. Therefore, the potential market for new functions gradually declines.

This obsolescence problem suggests a basic business strategy: gradually expand the scope of the operating system to encompass new system-related functions. Examples would be incorporating security protection, file-compression utilities, Web browsers, and other similar functions directly into the operating system. I cover this topic further in the next column.

The obvious conclusion is that, unless the operating systems people can continue finding revolutionary new ways to use computers, and unless each new way appeals to a large population of users, the operating system business cannot survive as an independent business. While its demise is not imminent, it is inevitable.

In the next column, I will continue this examination of the operating systems business. Then, in succeeding columns, I will cover what these trends in applications and systems programming mean to software engineering, and what they mean to each of us. 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.

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 Marsha Pomeroy-Huff, Julia Mullaney, Bill Peterson, and Mark Paulk.

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 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.

References

[Colwell] R.P. Colwell, et al. “Instruction Sets and Beyond: Computer, Complexity, and Controversy,” *IEEE Computer*, vol. 1819, 8-19, Sept. 1985.

[Denning] P.J. Denning, “Virtual Memory,” *Computing Surveys*, 2, 3, September 1970, pages 153–189.

[Humphrey] W.S. Humphrey, “Reflections on a Software Life,” *In the Beginning, Recollections of Software Pioneers*, Robert L. Glass, ed. Los Alamitos, CA: The IEEE Computer Society Press, 1998, pages 29–53.

[WSJ] *The Wall Street Journal*, February 1, 2001, page A18.

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.