Carnegie Mellon
Software Engineering Institute

news @ sei
interactive

Volume 4 | Number 4 | Fourth Quarter 2001

In This Issue

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

news @ sei interactive

From the Director

The right software,
delivered defect free,
on time, on cost, every time.

This is the vision of the Software Engineering Institute (SEI). To be successful, teams of acquirers, developers, and users must have the necessary software engineering skills to work together to ensure that the right software—well-designed software that is free of defects and that satisfies requirements for functionality, performance, and cost throughout its life cycle—is delivered to end users.

As a federally funded research and development center dedicated to achieving this vision, the SEI's role is to work with

- the research community, to help create and identify best software engineering practices;
- software developers and acquirers, to apply new and improved practices; and
- the community at large, to amplify the impact of these practices through widespread adoption.

I am proud of the progress we have made toward achieving this ambitious vision in 2001. I would like to use this space to review what I think have been some of our most noteworthy achievements during the past year, based on goals that we set for ourselves when the year began.

Create

Support development and initial use of the CMMI Framework. The Capability Maturity Model® (CMM®) Integration℠ (CMMI℠) project, jointly sponsored by the Office of the Under Secretary of Defense (Acquisition, Technology, and Logistics) and the Systems Engineering Committee of the National Defense Industrial Association, builds on our longstanding expertise in improvement based on the CMM for Software (SW-CMM). This project was formed to facilitate the use of multiple CMMs for improvement in multiple disciplines. Highlights in 2001 included the first public release of CMMI models and support from a wide range of government and industry organizations. To date, more than 20 pilots of CMMI have been conducted in a wide range of organizational contexts. The impact of this work will be amplified by the more than 40 organizations that have been certified by the SEI to offer education, training, and assessment services related to the CMMI models. See http://www.sei.cmu.edu/cmmi/.

Sustain technical leadership and publication record. Our staff continues to advance research in the field of software engineering. An article by R. L. Glass and T. Y. Chen in the Journal of Systems and Software 59 (2001), pp. 107-113, rates Carnegie Mellon/SEI the number one institution for publishing scholarly articles in the field of systems and software engineering (SSE).

Introduce new core competency in software components. Through a project that we call "Predictable Assembly from Certifiable Components," we are creating and maturing engineering practices for building systems from components.

Apply

Create preplanned programs of work with senior acquisition executives in the U.S. Army, Navy, and Air Force focused on the use of new and improved practices within the acquisition community and industry bases. An example of our progress on this goal is a portfolio of new work that we have undertaken for the Assistant Secretary of the Army (Acquisition, Technology, and Logistics). For more information, see the SEI Special Report Army Workshop on Lessons Learned from Software Upgrade Projects at http://www.sei.cmu.edu/publications/documents/01.reports/01sr021.html.

Position the CERT® Coordination Center (CERT/CC) and the SEI to anticipate new threats to networked systems and to have more impact. During the first three quarters of calendar year 2001, the CERT/CC handled 34,754 incidents, reported 1,820 vulnerabilities, published 29 security alerts, and provided testimony to two congressional hearings. The CERT/CC also played a major role in alerting the Internet community, providing reliable information, and helping to mitigate the damage caused by such threats as the Code Red and Nimda worms.

Amplify

Also in 2001, we continued to amplify the impact of the CERT/CC through our Networked Systems Survivability (NSS) Program.

- The Electronic Industries Alliance (EIA) and the SEI formed the Internet Security Alliance (ISA, www.isalliance.org), a comprehensive, global initiative seeking to advance information security practices.
- Addison-Wesley published the *CERT Guide to System and Network Security Practices*, written by Julia Allen, one of seven books published by SEI staff members this year in the SEI Series in Software Engineering (http://www.sei.cmu.edu/products/publications/ sei.series.html).
- We published the *OCTAVE℠ (Operationally Critical Threat, Asset, and Vulnerability Evaluation℠) Method Implementation Guide*. The OCTAVE Method is a self-directed risk evaluation for information security. See http://www.cert.org/octave/omig.html.

We also made progress on these amplification goals:

Demonstrate and document defect-free software-development methods. Results from adopters of the Team Software Process℠ (TSP℠) continue to demonstrate the viability of our vision of defect-free software. On efforts ranging from a few thousand lines of code up to one hundred thousand lines of code, typical TSP projects produce near-zero defects in delivered software; product quality that is from two to ten times better than comparable projects in the same organization; cost and schedule performance that

are within 10% of planned values; and reduced test costs and schedules (5-10 times—from months to days). See http://www.sei.cmu.edu/tsp/results.html.

Demonstrate and document a DoD case of product line practice. We published a case study of strategic and systematic reuse of software assets across a family of similar ground-based spacecraft command-and-control systems built under the direction of the U.S. National Reconnaissance Office. The case study, online at http://www.sei.cmu.edu/publications/documents/01.reports/01tr030.html, documents measurable benefits on one operational system including a sevenfold increase in productivity, tenfold increase in quality, and 50% reductions in cost and schedule.

Document evolutionary acquisition (EA) practices for software-intensive systems. We conducted a workshop/tutorial on EA at the 11th Annual PEO/SYSCOM (Program Executive Officers/Systems Command) Conference in October 2001. EA extends the risk-management aspects of spiral development to earlier stages of the transition from raw technology to deployed system. For more information about SEI support for EA, see http://www.sei.cmu.edu/cbs/spiral2000/.

As we move forward into a new year, we have recently revised our strategic plan to address key technical and management challenge problems defined in consultation with our primary sponsor. The challenge problems are in the areas of survivability/security, interoperability, sustainment, software research and development, acquisition management, use of metrics, and commercial off-the-shelf (COTS) software. Future issues of news@sei will provide you with more details on how we will "create, apply, and amplify" new and improved practices that meet the evolving challenges faced by acquirers and developers in these areas.

Stephen E. Cross
Director, Software Engineering Institute

Security Matters

Attack Scenarios: How to Get There from Here

Larry Rogers

If your goal is to attack a computer system, it is extremely useful to know all the attack scenarios available to you. If your goal is to defend a computer system, then knowing which vulnerabilities appear in the most attack scenarios helps you to defeat the wily intruders more effectively than by simply fixing each vulnerability that comes to light. If your goal is to search for and describe vulnerabilities, then using a consistent language is crucial to documenting your results.

Remember *Star Trek*'s transporter mechanism? "Beam me up, Scotty" allowed Jim Kirk, Spock, and their friends to be transported from one place to another instantaneously. Well, that is science fiction, but it is an analogy that is useful in describing how computer systems' vulnerabilities are exploited.

First, recall that in *Star Trek,* Spock and Kirk had to be somewhere that could be defined to the transporter. That definition was probably something like coordinates, such as 4 degrees north, 20 degrees west. We'll call that the precondition. Then, after the famous "beam me up" directive, the explorers magically reappear on the starship *Enterprise*, ready to move on to the next phase of their mission. We'll call that the impact.

On a computer system, each vulnerability must also have a precondition and, after the vulnerability is exploited, an impact. For example, there is a vulnerability in rlogin that enables users to access their computer accounts from a location other than their desktop computers. [For more on the rlogin vulnerability, see *rlogin(1): The Untold Story* at <http://www.sei.cmu.edu/publications/documents/98.reports/98tr017/98tr017abstract.html>.] The precondition is that the intruder must be able to execute the rlogin program. Usually, that intruder must be logged into a computer system with the vulnerable version of rlogin installed, so we'll call that precondition *become a local user*. Once the vulnerability is exploited, the impact is that that intruder has now gained the ultimate access to a UNIX system, namely *root* access (*root* is the all-privileged user similar to administrator on Windows/NT systems). So, with respect to the rlogin vulnerability, we can say that a *local user* can become *root* on a vulnerable system.

Now, *Star Trek* is fiction and, unfortunately, transportation in real life is much more complicated. Say that you are going on a trip and starting from your home. You need to get to the airport, so you get into your car and drive. Eventually, you get to the parking lot at the airport and board a shuttle to the terminal. The shuttle drops you at the curb, and you then walk to security and perhaps to a "people mover" inside the airport. At the end of that ride, you walk to the gate, check in, and board the plane. Once you land, you walk to the local transportation area

for a cab, car rental, or public transportation. You eventually get to your hotel and check in. What a trip!

The key to notice here is that each activity dovetails nicely to the next activity. That means that when you parked at the airport, there was a shuttle to take you to the terminal (eventually). When you were in the terminal, there was a way to get to the appropriate gate, and so on. The impact of each activity matched the precondition of the next activity. You can chain the activities together to achieve your goal of "taking a trip."

There are alternatives. You can build your "take a trip" chain from many different activities. The only requirement is that the impact of one activity matches the precondition of the next activity. For example, you could park at the airport and walk to the terminal. Both the "take the shuttle" and "walk" activities have preconditions that match the impact of "drive to the airport."

Let's return to the example of the rlogin vulnerability. Its precondition is to be a local user. To achieve the impact of gaining full privileges, you need to exploit at least one other vulnerability. To determine which one, it's helpful to have a vulnerability catalog in which the preconditions and impacts of vulnerabilities are clearly defined in a language that helps identify the "chain" of preconditions and impacts. After identifying your goal (or ultimate impact) you can chain backwards to whatever precondition suits you, taking into account the operating system version, patches, and so on. This chain of vulnerabilities is an *attack scenario*.

For all the vulnerabilities in the catalog, you can automatically build all the possible attack scenarios given an initial precondition and an ultimate impact. What is a reasonable initial precondition and ultimate impact? The most general initial precondition is that of an *arbitrary Internet user* and the ultimate impact is *root/administrator privileges*. Given those, it is possible to glue together the vulnerabilities according to the precondition and impact directives to construct all possible attack scenarios for achieving the goal.

Now, having done this, imagine that some of the attack scenarios require one vulnerability to achieve their goal. These are *home run* vulnerabilities—you can achieve your goal in one step. Examples include exploiting vulnerabilities in electronic mail, the Web, and FTP servers. Organizations typically support these protocols by passing them through their firewalls and other perimeter defenses and running the associated programs with high privileges. Most vulnerabilities found in these services can be exploited by an *arbitrary Internet user* to achieve *root*, resulting in a serious security compromise: the intruder gains complete control of your system.

All vulnerabilities to computer systems are important, either by themselves or in combination with others. Knowing what is possible helps you to defend your systems against those who seek to attack you.

"Shields up!"

“Aye, aye, Captain!”

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

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.

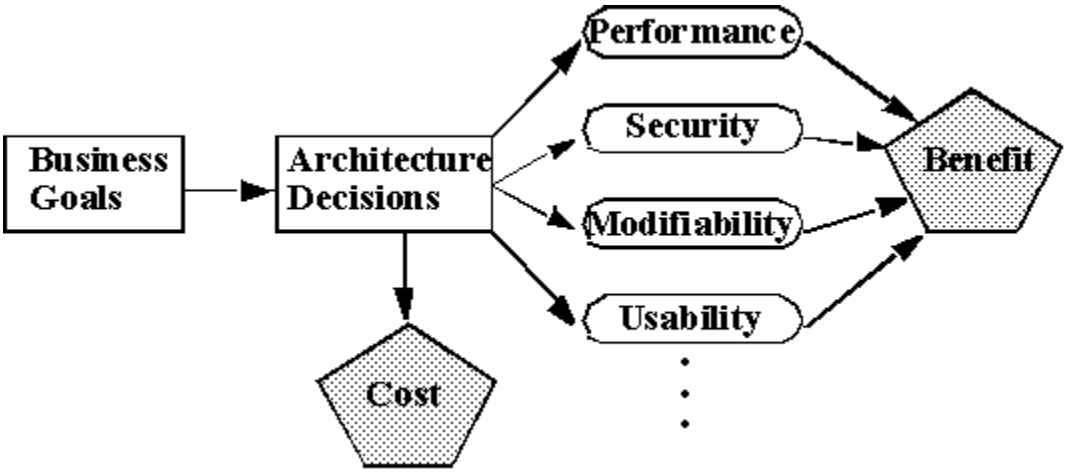

Figure 1: The Context for the CBAM

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".

---

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

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 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”.

The COTS Spot

Automating Design Search

Robert C. Seacord

In my last column, I discussed the relationship of design and search, namely that the design of systems based on commercial off-the-shelf (COTS) components consists of a search for compatible *ensembles* of commercial components that come closest to meeting system objectives[1]. In this column, I discuss a technique and tool for automating this process that, of course, incorporates search technology.

Broad Landscape

The commercial software market offers a broad landscape of software standards, technologies, and products. While it is sometimes easy to discern and relate the features of this landscape, other times components are hopelessly commingled. COTS-based system-development processes, such as the use of model problems [see <http://interactive.sei.cmu.edu/news@sei/columns/the_cots_spot/2001/2q01/cots-spot-2q01.htm>], can be used to establish compatibility among defined component interactions. Evaluation techniques, such as risk-misfit, can be used to establish preference of one solution over another. [For a presentation about risk-misfit, see <http://www.sei.cmu.edu/cbs/cbs_slides/99symposium/056pr.pdf>.] However, neither of these processes guarantees that the entire design space is searched, primarily because these are labor- and knowledge-intensive processes (with both labor and knowledge often in short supply).

Automation

We at the SEI have made a number of attempts to solve this problem. One effort, a collaboration with the National Institute of Standards and Technology's Manufacturing Engineering Laboratory, attempted to automate knowledge collection. The resulting prototype, Agora [link to http://www.sei.cmu.edu/publications/documents/98.reports/98tr011 /98tr011abstract.html], was successful in automating the collection of component interface information using a combination of search technology and introspection [2]. However, Agora was limited to gathering information maintained at runtime by the component model. In no case did this include semantic descriptions of the component or component methods, and in many cases the amount of useful information that could be gathered was nearly nil.

Having probed the limitations of automated knowledge collection, we decided to adopt another approach, namely automating the evaluation of ensembles formed from qualified component information. In this case, knowledge about components, in the form of component specifications, is provided by component developers, system integrators, and other software engineering professionals. The knowledge-based automated component ensemble evaluation (K-BACEE)

tool can then reference this information in evaluating the compatibility of component ensembles [3].

K-BACEE

K-BACEE helps automate the search for compatible ensembles of commercial components that come closest to meeting system objectives. This, of course, requires that the system integrator provide a statement of the system objectives. To this end, K-BACEE accepts a *system manifest* that, simply stated, provides a list of components that are required by the system. This is more than a simple software-requirements specification because it assumes a decomposition of functionality among components. The system manifest can also be complemented by a description of the interactions allowed among the components and additional constraints on both individual components and the overall system. The extended manifest, in this case, is very close to an architectural description of the system.

K-BACEE searches the repository of component specifications for components that match the system objectives described in the manifest. These components are then grouped into possible ensembles and evaluated for compatibility. Compatibility is assessed using a knowledge base of integration rules.

As part of the K-BACEE effort, the SEI is fostering a community of system integrators and component developers to assist with knowledge-base development. In support of this community-building effort, a birds-of-a-feather session is planned for the International Conference on COTS-Based Software Systems (ICCBSS) conference (see <http://wwwsel.iit.nrc.ca/iccbss/> in Orlando, FL, in February 2002. For more information on the K-BACEE community effort please see the K-BACEE Web site at <http://www.sei.cmu.edu/cbs/k-bacee>.

Summary

While K-BACEE cannot possibly guarantee that the entire design space is explored, it does allow a computer to do what computers do best—evaluate a large amount of information quickly in an unbiased manner. When used in this manner, K-BACEE can be considered a “discovery aid” that allows the system integrator to consider a broader range of possible component ensembles than is otherwise possible.

An initial K-BACEE prototype has been developed and continues to be improved. Community assistance is actively being solicited and incorporated in the ongoing development effort.

References

[1] Wallnau, K., Hissam, S., & Seacord, R. *Building Systems from Commercial Components*. Addison-Wesley, June 2001, ISBN: 0201700646.

[2] Seacord, R., Hissam, S., & Wallnau, K., "Agora: A Search Engine for Software Components." *IEEE Internet Computing* 2:6, November/December 1998, pgs. 62-70.

[3] Seacord, R., Mundie, D., & Boonsiri, S. "K-BACEE: Knowledge-Based Automated Component Ensemble Evaluation," published in proceedings of the 2001 Workshop on Component-Based Software Engineering, held in conjunction with the 27th Euromicro Conference, Warsaw, Poland, September 4-6, 2001, IEEE Computer Society.

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.

Watts New

The Future of Software Engineering: IV

Watts S. Humphrey

This is the fourth of five columns on the future of software engineering. The first two columns focused on trends in application programming, particularly related to quality and staffing. The previous column covered systems programming and the systems-programming business. In this column, I explain the three kinds of operating-systems (OS) businesses and predict where these businesses are likely to go in the future.

Systems Programming

To refresh your memory, the principal points made in the previous column were as follows:

- The objective of systems programs is to provide users with a virtual computing environment that is private, secure, and reliable.
- Over time, systems-control functions have gradually migrated from software to hardware. For example, when I wrote my first program, we had to read and handle each character. Hardware now does that. Similarly, many functions that were previously handled by software, such as memory management, interruption handling, and protection, are now handled by hardware. While the advent of personal computers temporarily halted this migration, it is driven by technology and will almost certainly resume in the future.
- The objectives of the organizations that make and market operating systems and the objectives of their users are naturally opposed. To maintain and grow their businesses, operating-system suppliers must continually enhance their systems or otherwise entice their users to upgrade. Conversely, computer users seek stability, reliability, and compatibility, and generally want to continue using their current versions. Since operating systems do not wear out, rot, or otherwise deteriorate, the installed life of old versions of operating systems could become very long indeed.

The principal conclusion of the previous column was that a standalone business for operating systems is not viable over the long term. Ultimately, the supply of attractive new functions will be depleted. Then, while people will need occasional enhancements and new operating-system versions for new hardware, they will prefer to stay with their existing operating-system version, rather than buy a new one. This rather stable operating-system business will likely be viable, but it will be very different from what we know today.

Even though the operating-system business will probably not survive by itself, it would be a natural companion to a hardware business. In that case, you might expect the hardware companies to absorb the operating-systems businesses. However, in today's world, it seems more likely that the operating-systems businesses will absorb the hardware companies.

The Internet

One might argue that the Internet changes everything. It is true that the Internet is a radical change and that it presents enormous opportunities for innovation and creativity. However, since the Internet revolution is still in its infancy, there are likely to be many surprises. But, since I am being controversial in this column, I might as well stick my neck out and hazard some predictions.

One likely way to couple computers and the Internet would be essentially to move the Internet inside the application programming interface (API). This would use the Internet to reference data and programs, regardless of their physical location. It would also presumably permit multiple remote systems to cooperate much as they could if they were at the same location. Producing these capabilities would be a substantial challenge and, when accomplished, would provide a glorified data, file management, and distributed computing capability. While such offerings will almost certainly be started by the software businesses, the Internet can be viewed as just another device. As advantageous as this Internet capability would be, its support would best be handled by hardware. Ultimately, the most efficient and cost-effective way to handle device support is as a hardware facility and not as a new operating-system function.

Second, the idea that people will use the Internet as a pervasive computing resource is not realistic. People will certainly use the Internet for communications, for retrieving programs and data, and for incidental and cooperative processing. However, most will not use it as some kind of computing utility, and anyone who believes they will does not understand history. The problem is not communication speed or computing capacity. We had computing centers decades ago with fast access and private terminals. Even when computing power greatly exceeded people’s needs, users were not satisfied with remote support. The problem was not technical; it was both personal and political. People simply wanted to control the resources they needed, and no amount of remote capacity could satisfy them. This was true then and, as computing capability becomes less and less expensive, it is inconceivable to me that people will want to use remote computers for their bread-and-butter needs. This will be particularly true when they can

get supercomputer power of their own for less than it costs to buy the desk on which they will put it.

Application Service Providers

This implies that the advent of application service providers (ASPs) is an anomaly. ASPs provide computing capability, essentially as a utility. Many view ASPs as the wave of the future and have invested a great deal of money in them. While it is possible that ASPs will become big business, the prime reason that organizations subscribe to ASPs is to avoid the cost and expense of operating their own computing systems [1].

In essence, the reason that the ASP business is attractive is not because it is a fundamentally new way of doing business. It is attractive as a way to avoid the costs and headaches of current computing systems. This implies that the entire ASP industry depends on our inability to make computing systems that are easy for the public to install and use. However, depending on the continued unresponsive performance of an entire industry is highly risky. That may be a viable strategy for a niche offering, but now that the ASP and software service businesses are growing faster and generating more profit than all other parts of the computer industry combined, we can expect things to change.

There is no question that many organizations can make a great deal of money operating computing facilities for their clients. However, this business presents a tempting target for someone to produce a computing system that is so simple to install and operate that most people could do it with little or no training. Then, much of this service industry would be replaced by a new class of highly usable systems. Of course, there would still be three important parts of the software service business that would not go away:

1. the custom business of adapting computing systems to the unique needs of businesses

2. adapting the business practices of some organizations to the features and capabilities of available systems

3. incidental use of the large volumes of programs or data resources that are likely to be available in online libraries

While these three categories are all likely to be important businesses, they are not the principal reason that most organizations currently use ASPs. Most do so to avoid the cost and aggravation of installing, maintaining, and using their computer systems.

The Time Scale

Before concluding my argument for why a standalone operating-system business is not viable, I must comment on timing. I started preaching about the importance of usability many years ago. At the time, I was IBM's director of programming and the company's 4,000 systems programmers all worked for me. While I should have been able to affect what they did, and while nobody disagreed with me, I was unable to get much done. Since many others have long preached the same usability story, you might wonder why so little has been accomplished. I have concluded that there are four reasons:

1. A great deal has already been done, but the steps to date have been only a small fraction of what is needed.

2. Since true usability will require an enormous computing capability, we are just now beginning to get the technology we need.

3. The software community has had many other, more pressing problems.

4. Even when usability is a top priority, there is so much to do that it will take a long time to build the kinds of systems that are needed.

This suggests that the current ASP and software service businesses are likely to be viable for many years, but not forever.

Kinds of OS Businesses

In exploring what is likely to happen in the future, we must first consider the three main kinds of operating-systems businesses and their characteristics. These three business types are as follows:

1. First are hardware manufacturers that offer operating systems as product support. Examples of this are IBM, Apple, Sun, and others.

2. Second are standalone operating-systems businesses like Microsoft with its Windows offerings.

3. The third case is the "open-source" operating-system movement. Here, the prime examples are Linux and Unix.

The Hardware-Coupled OS Business

In the hardware-coupled operating-systems business, the objective has been to sell hardware. In projecting what will happen in the future, the automobile industry provides a useful analogy. For the first 50 years or so, automobile technology was engine-centric. That is, the design and marketing of automobiles featured the engine's power and reliability. Leading up to and following World War II, however, this changed. While engines continued to be important, they were no longer a principal discriminator in the buying decision. In fact, today, few people could tell you the horsepower or displacement of their car's engine. The last 50 years or so of the automobile industry have been largely dominated by comfort, style, service, and economy. We are also beginning to see safety, quality, and environmental concerns emerge as important buying discriminators.

This suggests that the hardware-coupled OS business will evolve from selling power, cost, and function to featuring usability, installability, reliability, and security. Since the hardware and software are likely to be marketed together, there will be little motivation to add capabilities that do not sell new systems. The profit motive would also limit new functions and features to those that could be financially justified. Since this is precisely the kind of business IBM had before we unbundled software, experience shows that operating-systems development will be tightly constrained and that there will be little motivation to add features purely to improve the capabilities of the existing systems. The key is what sells new products.

The Standalone OS Business

Not surprisingly, the objective of the standalone operating-systems business is to sell operating systems. Since one of the principal ways to sell them is with new hardware, we can expect the OS vendors to strive to increase the market for the hardware that uses their systems. While selling new operating systems with new hardware is an attractive business, it is largely captive to the ups and downs of the hardware business. This suggests that operating-systems vendors will add functions and features to make their systems attractive to installed hardware users. These OS vendors will then urge the customers to upgrade to the new operating systems without necessarily buying new hardware.

Because of the growing volume of application programs and because of the necessity of continuing to support an increasing number of old applications with every new OS version, the API must become progressively more stable. It might even become public. Then, possibly many years down the road, some clever and well-financed entrepreneur will produce a new OS that emulates the API of one or more of the dominant operating systems. This new operating system would presumably integrate the latest hardware and software technology to offer dramatically improved performance, usability, reliability, and security. Since the stand-alone OS suppliers would have trouble competing with software alone, they would either have to team up with hardware suppliers or lose much of their business.

The Open-Source OS Business

The third case is the open-source OS business. Here, the motives are entirely different. There is no desire to sell new hardware or software, only to provide a more usable, installable, reliable, and secure system. The great attractiveness of the open-source OS business is that it caters to the desires of a steadily growing body of installed users. These people feel that their current systems are marginally adequate and do not want to change or evolve their OS versions. They are not even terribly interested in the latest "gee-whiz" chip. They would just like installable, usable, reliable, maintainable, and secure systems that do precisely what their current systems do. While the operating-systems suppliers could largely eliminate the attractiveness of the open-system offerings by dramatically improving the user characteristics of their systems, that is not likely to happen very quickly. As a result, the open-source business will likely continue to grow. This also suggests that the open-source movement is, at least to some degree, competing with the software service and ASP suppliers.

Middleware

All of the preceding argument has ignored an important segment of the software industry: middleware. By middleware, I mean that growing family of programs that are used to administer, support, and use computing systems. This kind of software includes programs to handle administrative, operational, and support activities; provide support for application development; and furnish generic application support for system developers and users. Since, as I noted in the first column of this series, the volume of application programs will continue to grow for the foreseeable future, all of these middleware categories are also likely to continue to grow.

The challenges in the middleware business include all of the challenges of starting and running a new business in a competitive industry. They also include the challenge of resisting the threats and blandishments of the OS suppliers. Middleware businesses really are in the middle. While they must have creative and marketable ideas and the funds and know-how to start and run a business, once their ideas are financially successful, they become attractive targets. Since all three types of operating-systems businesses must continually add features to their systems to survive, the natural trend will be for the OS suppliers to incorporate the most attractive middleware features into their systems. They might either acquire the middleware companies or simply appropriate their ideas. This suggests that most middleware businesses will be transient. While they are likely to continue to be valuable sources of innovation, they will have to do four things to survive:

1. continue to have good ideas
2. be very effective marketers
3. have substantial financial support
4. protect their intellectual property

Summary

While the current situation is likely to continue essentially as it is today, at least for many years, technology will ultimately win, and we will see the standalone operating-system business merge into the larger computing-systems business. This, I am convinced, is the long-term answer. However, since the nature of the first two types of operating-systems businesses has been essentially static for more than 20 years, the long term could be very long indeed.

In my next column, I will explain what these trends in applications and systems programming mean for software engineering and what they mean for each of us.

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, Jim McHale, Julia Mullaney, and Bill Peterson.

In closing, an invitation to readers:

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

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

References

[1] Kerstetter, Jim, "Software Shakeout," *Business Week*, March 5, 2001, pp 72-80.

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.

At the Heart of the Revolution: The International Conference on COTS-Based Software Systems

Bob Lang

Industry and government are increasingly moving toward the use of COTS products—either as stand-alone solutions or as components in complex, heterogeneous systems.

But implementing software systems based on COTS products presents unique challenges that traditional software development practices do not address. As more systems depend more on the successful integration of COTS products, practitioners and researchers must be ready to meet these challenges.

The SEI, along with the National Research Council (NRC) Canada, and the USC Center for Software Engineering (USC-CSE), is proud to sponsor the International Conference on COTS-Based Software Systems (ICCBSS), the first conference to focus solely on the challenges of acquiring, building, fielding, and supporting systems that incorporate COTS software products. The conference will be held February 4-6, 2002 in Orlando, Florida.

Background

Fewer and fewer organizations today are building systems entirely from scratch. Instead, organizations are relying increasingly on COTS products as a basis for their software systems. This trend is a result of the expectation that building a system using proven, commercially available components will result in faster time to market and improved capability. More to the point, the use of COTS components is simply being mandated by everyone from industry executives to government lawmakers. In 1997, it was estimated that COTS software accounted for 25.5% of a U.S. corporation's IT portfolio. In 2002, that figure is expected to be at around 40%.[1]

For organizations designing and implementing a COTS-based system, the current market state presents a number of challenges. For example, it is difficult to discover the actual technical capabilities of a product or set of competing products, since there is no objective forum for product evaluation. Once individual products are selected, it is difficult to identify and resolve mismatches between products and the organization's business processes. In short, successfully implementing systems based on COTS requires new ways of doing business: new skills, knowledge, abilities, changed roles and responsibilities, and different processes.

For organizations working through these challenges or interested in sharing their experiences, ICCBSS offers a unique sharing, learning, and networking opportunity.

The Conference

ICCBSS consists of three intensive days of tutorials, presentations, and panel sessions. Keynote speakers, such as Ivar Jacobson of Rational Software Corporation and Barry Boehm of the USC Center for

Software Engineering, will contribute their insights on vital issues. The conference provides a variety of tutorials:

Evaluating COTS Software Products.
John Dean, National Research Council
Grace Lewis, Software Engineering Institute

Estimating COTS-Based Software Systems.
Chris Abts, USC Center for Software Engineering
Betsy Clark, Software-Metrics

COTS-Based Systems: Keys to Success.
Patricia Oberndorf, Software Engineering Institute

Building Systems from Commercial Components.
Robert C. Seacord, Software Engineering Institute

The conference also offers three tracks of refereed papers: process, technology, and experience.

Process

Traditional software development processes often do not work when applied to COTS-based systems. Commercial products in the marketplace are not built to meet the specific needs of a project. The papers in this track present new processes needed for building and maintaining COTS-based systems.

Examples of the papers in this track:

Decision-Making Techniques in the Procurement of COTS Software: the unholy alliance between requirements, decision-making and COTS selection.
Cornelius Ncube, Zayed University

Rethinking Process Guidance for Selecting Software Components.
Neil Maiden, City University

Technology

Integrating and trouble-shooting systems that use commercial products is challenging. Integrators lack visibility and control of COTS products. This track highlights both state-of-the-art and state-of-the-practice techniques used for building and maintaining COTS-based systems.

Examples of the papers in this track:

Identifying Evolvability for Integration.
Rose Gamble, University of Tulsa

On Building Testable Components.
Jerry Gao, San Jose State University

Experience

This track features commercial and government practitioners from a variety of software domains who will reflect on issues, successes, and helpful hints drawn from their experiences in integrating commercial off the shelf software in real COTS-based systems.

Examples of the papers in this track:

European COTS User Working Group: analysis of the common problems and current practices of the European COTS users.
Sandy Tay, European Software Institute

Five Hurdles to the Successful Adoption of Component-Based COTS in a Corporate Setting.
Anthony Earl, Sun Microsystems, Inc.

Join us at ICCBSS to discuss current practices and promising research directions.

For more information, contact—

Barbara Hoerr

Phone
412-268-3007

Email
iccbss2002@sei.cmu.edu

World Wide Web
http://www.iccbss.org

1 "Disposable Information Systems: Putting Maintenance in a Whole New Light." Jeffrey Voas. Available WWW: http://wwwsel.iit.nrc.ca/projects/cots/icsewkshp/slides/voas/tsld001.htm.

A Process for Evaluating COTS Software Products

Bob Lang

Many organizations are now constructing major software systems from commercial off-the-shelf (COTS) products. An essential part of such an undertaking is evaluating the commercial products that are available to determine their suitability for use in the system. Virtually all organizations perform an evaluation of COTS software products before using them, but still projects fail. These failures are often directly traceable to the quality of the organization's evaluation process.

In response to these problems the SEI and the National Research Council Canada (NRC) have co-developed a COTS software product evaluation process that can be tailored to suit the needs of a variety of organizations. This evaluation process helps organizations evaluate COTS products to determine their fitness for use in their systems. The process is being taught in a two-day tutorial that can be delivered at the SEI or a customer organization. A half-day version of the tutorial is also being offered at the International Conference on COTS-Based Software Systems in Orlando, Florida, February 4-6, 2002.

The Need for Evaluation

The importance of choosing the right product for a COTS-based system cannot be overstated. The influence that the product has on the system is pervasive: the COTS product can determine the system architecture, the functional capabilities of the system, and even the maintenance processes for the system. With some COTS-based systems running into the tens and even hundreds of millions of dollars, the risk of failure is too great not to invest in evaluating the products that these systems are based on.

When choosing to make use of commercial components, the question then becomes how to assess or evaluate these products. In addition to considering specific techniques for evaluation, there is also the need to define some of the more general process-related issues that arise when evaluating COTS products. For example, whose job is it to do this? How do the traditional notions of evaluation differ from COTS evaluation? What new activities might be implied when COTS products are under evaluation?

To help answer these questions, the SEI has developed a framework for an evaluation process that organizations can tailor to their needs. In addition, a set of techniques is identified that can be applied in this process.

The Evaluation ProcessError! Bookmark not defined.

The evaluation process is based in part on the experience of the SEI in working with organizations that have struggled with building COTS-based systems. Ed Morris of the SEI says, "A lot of organizations were struggling in part because of inadequate evaluation of commercial products. They would select a

product and they would find out later that, for example, the product didn't do as much as they expected. That's a failure of evaluation. And we saw other cases where an organization bought a product and a few months later the company that sold the product would go out of business, which is another failure of evaluation."

Based in part on ISO 14598, the high level process is flexible and can be adapted to many specific process implementations. It consists of four basic elements:

- Planning the evaluation
- Establishing the criteria
- Collecting the data
- Analyzing the data

In addition to the basic process, there is a set of techniques to help in planning, establishing criteria, and collecting and analyzing data.

The elements are summarized below. The tutorial provides an in-depth look at each, and suggests techniques that practitioners can use throughout the process.

Planning the Evaluation

The organization determines level of effort required and estimates cost and schedule. To identify the level of effort, organizations must consider the criticality of the components and candidate products in relation to strategic objectives. The greater the technical risk, and the more critical the strategic objectives, the greater the rigor required.

Although there are few specific techniques for estimating resources and schedule for COTS evaluation, several general techniques are applicable—for example, expert opinion, analogy, decomposition, and cost modeling.

Establishing Evaluation Criteria

The first step in establishing evaluation criteria is to distill from the full set of requirements the requirements that are appropriate for the evaluation. Determining evaluation requirements involves analyzing system requirements to determine their applicability, and generating new requirements specific to the use of the COTS product. From these requirements, evaluation criteria are developed, consisting of a capability statement (a measurable statement of ability to satisfy a need) and a quantification method (a means for assessing the product's level of compliance with the capability statement).

Collecting Data

The organization collects information about how the products perform against the evaluation criteria developed previously. Different criteria and situations require different data collection techniques. For example, the technique applied to determine the value of a critical criterion will be quite rigorous; other techniques for non-critical criteria will be less rigorous. The tutorial outlines a number of techniques that can be used to collect data. It emphasizes "hands-on" techniques that collect data by actually running the product in sample scenarios.

Analyzing Results

The organization takes the data collected and consolidates it into a form that can be analyzed. Some useful techniques for data analysis are sensitivity analysis, gap analysis, and cost of fulfillment. Sensitivity analysis helps determine how the evaluation results are affected by changes in assumptions, such as a change in the weighting of criteria. Gap analysis highlights the gap between the capability provided by a COTS component and the capability required for the system. Cost of fulfillment helps determine the effort needed to narrow such a gap. For example, fulfillment could involve altering the system architecture, adding features, or modifying the requirements.

For more information, contact—

Lorraine Nemeth

Phone
412-268-7777

Tricia Oberndorf

Email
po@sei.cmu.edu

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

The Internet Security Alliance: Leadership in Information Security

The Internet has grown exponentially in the last decade. As the infrastructure has grown so has the number of users. What was once a small community of professionals exchanging research information has become a diverse group of students and researchers, novices and experts. As users have become more diverse, so have the hardware, software, and services available from Internet service providers, Web sites, programmers, and technology companies.

This particular combination of users, services, and high expectations poses serious threats to industries and organizations who now live in an electronic world where, ten years ago, trust was typically assumed.

The Internet Security Alliance

was created to provide a forum for information sharing and leadership on information security issues. It represents industry's interests to legislators and regulators and aims to identify and standardize best practices in Internet security and network survivability while creating a collaborative environment to develop and implement information security solutions. The alliance is a collaborative effort between Carnegie Mellon' Software Engineering Institute (SEI), its CERT¨ Coordination Center (CERT/CC), and the Electronic Industries Alliance (EIA), a federation of trade associations.

The Mission of the ISA

The mission of the ISA is to use the collective experience of its members to promote sound information security practices, policies, and technologies that enhance the security of the Internet and global information systems.

The ISA offers members a single portal for up-to-the-minute threat reports, best security practices, risk management strategies, and more, which will give them the edge in the competitive and volatile environment of the Internet. Further, the Internet Security Alliance will undertake these and other crucial activities:

- Provide early warning of emerging security threats
- Facilitate executive-to-executive communications about solutions to threats and emerging trends
- Conduct research leading to identification and resolution of root causes of problems
- Develop training and certification programs in information assurance and other fields
- Initiate standard-setting activities on the foundation of EIA's 75-year heritage in the standards world
- Develop organizationally viable models for integration and adoption of security practices

Benefits of Membership

ISA membership benefits are many-fold. Members receive otherwise unobtainable early warnings of information security threats, as well as mitigation strategies, expert analysis of reported software vulnerabilities and intruder activity, the opportunity to contribute to the development of industry standards, and a forum to collaborate with staff from other organizations across multiple industries on shared information security concerns.

A key distinction between the ISA and existing groups organized around information security issues is the breadth of its activities. Many of the existing information security groups are centered around specific issues and industries or market sectors. The ISA aims to cut across industries and market sectors to develop a truly global approach to the problems inherent in electronic commerce and communications. While the ISA will readily promote information sharing among its membership, it also will provide advanced notice and detailed analysis of information security threats, access to historical software vulnerability and intrusion data, as well as a means to develop globally recognized standards and practices.

Working Groups

The Internet security issue continues to be a critical issue facing today's corporations and small businesses. By combining the strengths of its diverse membership, the Internet Security Alliance offers its members the opportunities to both share their experiences and develop helpful solutions to some of today's Internet problems. The initial core efforts for Internet Security Alliance will cover the following areas: Management Practices, Technology, and Policy. Each of these working groups is challenged to provide input and knowledge to establishing a base set of recommendations for their respective areas to the whole of the membership.

Management Practices

Management practices is a core element of Internet security. The challenge for this working group is to determine, along with Alliance staff, overall best practices for Internet security. The following are some of the areas that the managing practices working group are concentrating on for Alliance members:

- Establish the Internet Security
- Alliance as an accredited standards organization
- Determine business risk metrics
- Develop cost vs. loss tables
- Develop a standard process to calculate business loss

Technology

The technology working group will be primarily responsible for facilitating the ISA's efforts in information sharing, tools exchange, early threat warning, and vulnerability analysis. Technical staff will also work with the technology working group on dispersing technical threat reports as well as participation on upcoming white papers and reports.

Policy

The recent call for legislation to assist in helping defend governments and corporations from the threat of cyber attack truly denotes the importance of sound policy. The policy working group is charged with determining valid recommendations to government bodies on information security issues. Though the Internet Security Alliance does not include government entities as potential members, corporate leadership in public forums on information security is critical. The policy working group will look at global policies and legislative priorities that affect companies around the world. Legislation dealing with issues such as privacy and information-sharing continue to be important steps for the private sector to undertake. Here are some preliminary objectives that the policy working group will be considering:

- Establish Internet Security Alliance Privacy agenda; defending against potentially binding legislation that may be introduced or is already in existence
- Promote legislation calling for increased levels of information sharing among corporations and government

Membership in ISA is open to corporations from around the world.

For more information, contact—

Internet Security Alliance

Phone
703-907-7090

World Wide Web
http://www.isalliance.org

TransPlant: Helping Organizations to Make the Transition

When the CERT® Coordination Center at the SEI needed a plan for disseminating 50 Internet security practices, Julia Allen and her team relied on TransPlant. A facilitated planning process developed by the SEI for diffusing and adopting emerging software engineering technologies, TransPlant helps technology developers to take a practical approach to technology transition.

For many research and development organizations, technology transition—the process of creating or maturing a technology, introducing it to its intended adopters, and facilitating its acceptance and use—is a challenging and unpredictable activity. TransPlant solves this problem by enabling organizations to develop an actionable plan for improving a technology's successful transition. It can be used by Department of Defense (DoD) researchers and developers, managers of advanced technology development (ATD) programs, managers of acquisition or operational programs, and commercial organizations planning the rollout of a technology to multiple organizations or units.

For Allen, TransPlant resulted in a transition strategy for deploying her team's security practices and an array of useful materials for communicating that strategy to others. Allen notes how effective TransPlant was in jump-starting its transition efforts: "We derived benefit at every step of the TransPlant process in terms of creating artifacts, descriptions, and other products that we could immediately put to use," says Allen, whose team piloted TransPlant from August 2000-July 2001. "This provided strong motivation and measurable value to continue. We did not need to wait until we had a completed plan to get 'useful stuff.'"

This "useful stuff" is one of TransPlant's most compelling advantages, says Eileen Forrester, team leader for TransPlant. Forrester, who has been evolving TransPlant since the late 1990s, believes technology transition can be an effective process if technologists, managers, and other professionals are taught the appropriate skills and given the right tools for thoughtful planning. "Those of us who work in technology transition can get up in the clouds, musing over academic theories that account for how transition happens. But I like to take academic theories and models and figure out how to make them pragmatic, so that hard-nosed, results-oriented engineers and their managers can see an improvement in the adoption of their technologies," Forrester says.

TransPlant features a set of work products that comprise the transition plan for an organization's particular technology, and a template with guidance on how to construct the plan from these work products. TransPlant can also show technology developers how to attract the right kind of adopters and collaborators for their technology, and overcome obstacles for getting their technology into use.

By incorporating elements of strategic planning, product management, marketing, and communication, TransPlant helps teams to create dynamic transition strategies through its seven process steps:

1. Define problem, solution, and scope for planning.
2. Decide on a transition strategy.
3. Characterize adopters.

4. Identify effective transition mechanisms.
5. Select and synthesize: refine scope and strategy, design interactions among adopters, refine whole product and set priorities for action.
6. Prepare to manage risk.
7. Document the plan.

The impetus for TransPlant, Forrester says, stems from some misconceptions in the software community about technology transition itself. Often, people in software-intensive organizations assume that technology transition will happen automatically if the team has built a good technology. “I call this the “better mousetrap” fallacy,” Forrester says. “Technologists tend to think that if they build a good thing, people will find their way to it and adopt it on their own, based on its inherent goodness...Wrong.” The value of the technology has never been a good predictor of adoption and use.

“Plus, engineers sometimes think transition is one of those things anyone could do if they thought about it for five minutes,” Forrester says. “Effective transition practices are built from solid principles and disciplines. On the other hand, some software practitioners take the opposite tack, and think transition is something mysterious and not amenable to planning and design. Transition can be managed, much as any software project, though it requires skills engineers may not have.”

TransPlant currently exists as a facilitated process, with Forrester or one of her team acting as coach. There are three ways to apply TransPlant: 1. as a hands-on series of facilitated working meetings, 2. as a three- to four-day workshop, and 3. as a “hands-off” process, in which the technology team receives primers and mentoring from the coach only as needed. In all three approaches, participants learn about the drivers in the TransPlant process, such as dimensions of transition strategy, understanding the characteristics of adopters and how to use them in a transition strategy, the design and use of a value network (the total set of stakeholders, collaborators, and adopters who contribute to effective transitions), types of transition mechanisms (products and interventions for making transition happen), and risks (and mitigations for these risks) to successful transition.

Yet another way to learn about the process is through a Trans-Plant tutorial, which helps organizations to apply the Trans-Plant process to their own technology transition efforts. This full-day tutorial offers information on the challenges and opportunities in effective technology transition, the concepts behind TransPlant, and gives the participants exercises that are smaller scale versions of the activities in TransPlant.

TransPlant has been applied by 10 teams within the SEI as well as one DoD Science and Technology (S&T) organization and a small commercial software start-up company. Forrester is developing ways to make TransPlant into a tool that organizations can apply themselves. Next steps for TransPlant will include applying the process in large commercial software settings and in acquisition programs. The TransPlant team will also be looking for transition partners. Please contact the SEI if your organization is interested in learning more about TransPlant.

For more information, contact

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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