Carnegie Mellon
Software Engineering Institute

news@sei

Editor's Message

For more than 20 years, the Software Engineering Institute has created and promoted methods that help organizations efficiently develop high-quality software. Microsoft, one of the world's largest software vendors, recently piloted a project using the SEI's Personal Software Process℠ (PSP℠) and Team Software Process℠ (TSP℠) with a team of developers to deliver better software. The article "Microsoft's Pilot of TSP Yields Dramatic Results," discusses Microsoft's effort to build teams, reduce defects, and improve quality using TSP.

Another way the SEI is helping organizations improve quality is by demonstrating the benefits of product line engineering. Often, the first thing a manager in an organization considering product lines wants to know is the return on investment for changing the company's development strategy. "Calculating Return on Investment for Software Product Lines" provides a simple model to help software development managers understand the costs and benefits they might expect when switching to the product line approach.

Because of work by the SEI and others, software product lines have emerged as an important development paradigm, with an active community of practitioners. The Third Software Product Line Conference (SPLC 2004) will be held August 30 through September 2 in Boston. The conference will feature technical papers, topical panels, a rich selection of tutorials and workshops, demonstrations, birds-of-a-feather meetings, and new inductions into the Software Product Line Hall of Fame. For more information about SPLC 2004, go to http://www.sei.cmu.edu/SPLC2004/.

Another SEI conference, the Software Engineering Process Group (SEPGSM), was held in March and attracted just under 2000 attendees. "SEPG 2004 Showcases Enterprise Process Improvement in Orlando" describes how the conference theme of enterprise process improvement was developed in presentations, tutorials, and panel discussions.

Security is another aspect of SEI's emphasis on quality software. "CERT®/CC Instrumental in National Security Effort" discusses the announcement in September 2003, by the U.S. Department of Homeland Security, of the creation of the US-CERT, a joint effort of the Department of Homeland Security's National Cyber Security Division (NCSD), the CERT Coordination Center (CERT/CC), and the private sector to improve the nation's cyber security capability. US-CERT will build on CERT/CC capabilities to help prevent cyber attacks, protect systems, and respond to the effects of cyber attacks across the Internet.

Thanks for reading news@sei; if you've got questions or comments about what you see here, let me know at news-editor@sei.cmu.edu.

Janet Rex
Editor in Chief

The Architect

The Recovery of Runtime Architectures

Rick Kazman, Hong Yan, David Garlan, Bradley Schmerl, Jonathan Aldrich

An increasingly important requirement for software-based systems is their ability to adapt at run time to handle such things as resource variability, changing user needs, changing demands, and system faults. In the past, systems that supported such self repair were rare, confined mostly to domains such as telecommunications switches or deep-space control software where taking a system down for upgrades was not an option and where human intervention was not always feasible. However, today more and more systems have this requirement, including ecommerce systems and mobile embedded systems. For systems to adapt themselves, one of the essential ingredients is *self reflection*: a system must know what its status is, and it must be able to identify opportunities for improving its own behavior.

Traditionally software systems have operated in relatively stable, fixed environments (such as a desktop) and could be taken down for maintenance, upgrading, or replacement. However, software systems must increasingly function in environments where resources (such as wireless bandwidth) change rapidly, where their resource demands are difficult to predict, where they must interact with potentially faulty components and services not under their control, and yet where they must continue to operate continuously. In short, such systems must begin to take more responsibility for their own health and welfare, adapting at run time to handle errors, changing resources, and varying user needs.

Today software engineers have few tools or techniques to create such self-adaptive systems reliably, flexibly, and at low cost. Most techniques at our disposal rely on low-level mechanisms such as exceptions and timeouts. But these mechanisms generally provide little help in allowing a system to determine the true source of problems or to choose a response to them. Moreover, they are ineffective at dealing with softer problems, such as gradual performance degradation, or at recognizing opportunities to improve behavior even when things are not broken.

For the past year, we have been investigating a new paradigm for software systems that is already showing promise for solving this problem. The underlying idea is to associate with each software system a *reflective architecture model* that allows a system to reason about its own behavior at run time and take action to modify its own structure and behavior when necessary. By reflecting the current state of a system as an architectural model that exposes only the main components, interactions, and their high-level properties, a system can more easily understand what its current state is and take necessary actions.

A critical step toward achieving this vision is the ability to know exactly what the architecture of a running system is. We use run-time monitoring and abstraction, together with codified knowledge about architectural styles, to develop a dynamic view of a system's architecture as it runs. In this way, we can instrument a system with *probes* that produce streams of low-level system observations that are then interpreted by a rule-based abstraction engine to produce higher-level architectural events and operations reflecting the system's architecture as it is running. (See Figure 1.)

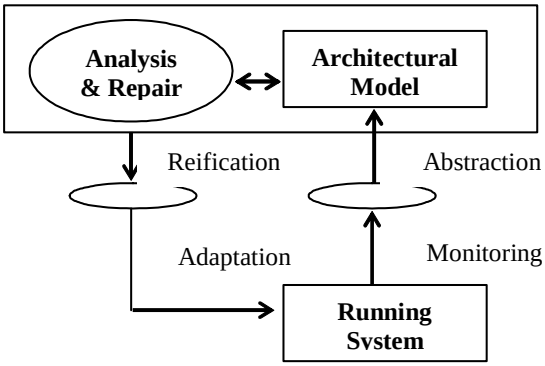

Figure 1 Architecture-Based Adaptation

Currently two techniques have been used to determine or enforce relationships between a system's architecture and implementation. The first is to ensure consistency by construction. This can be done by embedding architectural constructs in an implementation language where program analysis tools can check for conformance. Or it can be done through code generation, using tools to create an implementation from a more abstract architectural definition. While effective when it can be applied, this technique has limited applicability. In particular, it can usually be applied only in situations where engineers are required to use specific architecture-based development tools, languages, and implementation strategies. For systems that are composed out of existing parts or that require a style of architecture or implementation outside those supported by generation tools, this approach does not apply.

The second technique is to ensure conformance by extracting an architecture from a system's code using static code analysis.[1] When an implementation is sufficiently constrained that modularization and coding patterns can be identified with architectural elements, this can work well. Unfortunately, however, the technique is limited by an inherent mismatch between static, code-based structures (such as classes and packages) and the run-time structures that are the essence of most architectural descriptions. In particular, the actual run-time structures may not even be known until the program runs: clients and servers may come and go dynamically; components not under direct control of the implementers may be dynamically loaded; etc.

1. For more information see http://www.sei.cmu.edu/ata/ata_extraction.html.

A third technique—the one we are following here—is to determine the architecture of a system by examining its behavior *at run time.* Observations about its behavior can then be used to infer its dynamic architecture. This approach has the advantages that in principal it applies to *any* system that can be monitored, it gives an accurate image of what is actually going on in the real system, it can accommodate systems whose architecture changes dynamically, and it imposes no a priori restrictions on system implementation or architectural style.

There are a number of hard technical challenges in making this technique work. The most serious is finding mechanisms to bridge the abstraction gap: in general, low-level system observations do not map directly to architectural actions. For example, the creation of an architectural connector might involve many low level steps, and those actions might be interleaved with many other architecturally relevant actions. Moreover, there is likely no single architectural interpretation that will apply to all systems: different systems will use different runtime patterns to achieve the same architectural effect, and conversely, there are many possible architectural elements to which one might map the same low level events.

We have developed a technique to solve the problem of dynamic architectural discovery for a large class of systems. The key is to provide a framework that allows the mapping of implementation styles to architecture styles. This mapping is defined as a set of conceptually concurrent state machines that are used at run time to track the progress of the system and output architectural events when predefined run time patterns are recognized. By parameterizing the framework by both architectural and implementation styles, we are able to exploit regularity in systems while still providing flexibility in defining new abstraction mappings.

The system we have built to do this is called DiscoTect [Yan 04]. Any approach that supports dynamic discovery of architectures must be able to (a) observe a system's runtime behavior, (b) interpret that runtime behavior in terms of architecturally meaningful events, and (c) represent the resulting architecture. In DiscoTect we are primarily concerned with the second problem of bridging the abstraction gap between system observations and architectural effects.

In DiscoTect we adopt an approach illustrated in Figure 2. Monitored events are first filtered by a trace engine to select the subset of system observations that must be considered. The resulting stream of events is then fed to a state engine The heart of this recognition engine is a state machine designed to recognize interleaved patterns of runtime events and, when appropriate, to output a set of architectural operations. Those operations are then fed to an architecture builder that incrementally creates the architecture, which can then be displayed to a user or processed by architecture-analysis tools.

To handle the variability of implementation strategies and possible architectural styles, we provide a language to define new mappings. Given a set of implementation conventions (an *implementation style)* and a vocabulary of architectural element types and operations (an *architectural style*), we provide a description that captures the way in which runtime events should be interpreted as operations on elements of the architectural style. Thus each pairing of

implementation style and architectural style has its own mapping. A significant consequence is that these mappings can be reused across programs that are implemented in the same style.

We have used DiscoTect to recover the architecture of a number of systems, the largest of which are the JBoss J2EE Server (http://www.jboss.org) and Sun's J2EE Server (http://java.sun.com/j2ee/). Such is the generality of our approach that we were able to reuse most of the effort in creating the JBoss mapping when we analyzed Sun's J2EE Server.

We are now working on recovering the architecture of systems written in C and C++, to test the generality of our approach. Early results from this work appear promising. DiscoTect is showing tremendous potential as a tool to recover and monitor run-time architectures, a critical step in realizing architecture-based run-time adaptation.

Reference

[Yan 04] Yan, H.; Garlan, D.; Schmerl, B.; Aldrich, J.; & Kazman, R. "DiscoTect: A System for Discovering Architectures from Running Systems," *Proceedings of the 26th International Conference on Software Engineering (ICSE 26)*, Edinburgh, Scotland, May 23-28, 2004.

About the Authors

Rick Kazman is a senior member of the technical staff at the SEI, where he is a technical lead in the Architecture Tradeoff Analysis Initiative. He is also an adjunct professor at the Universities of Waterloo and Toronto. His primary research interests within software engineering are software architecture, design tools, and software visualization. He is the author of more than 50 papers and co-author of several books, including a book recently published by Addison-Wesley titled *Software Architecture in Practice*. Kazman received a BA and MMath from the University of Waterloo, an MA from York University, and a PhD from Carnegie Mellon University.

Jonathan Aldrich is an assistant professor at the Institute for Software Research International at Carnegie Mellon University, working in the areas of software engineering and programming languages. At CMU he leads the ArchJava project, which integrates a software architecture specification into the Java language, using type-based techniques to ensure that the code respects architectural constraints. Aldrich received a B.S. from the California Institute of Technology and a Ph.D. from the University of Washington.

CMMI in Focus

Choosing a Supplier: Due Diligence and CMMI Levels[1]

MIKE PHILLIPS

We often find that customers who are considering the acquisition of software intensive products and services demand a CMM/CMMI level from potential suppliers as a perceived guarantee of a provider's capabilities. We do believe that higher maturity organizations are more likely to deliver higher quality software intensive systems faster than those with less process discipline. We also believe that acquiring organizations should behave like those doing corporate acquisitions: they should apply "due diligence" to the choice of supplier.

For those who may be unfamiliar with such efforts, due diligence often includes a report containing information such as a Dun & Bradstreet financial review, debt analysis, and even turnover rate, as well as relevant past performance references. (A prospective developer with a track record of displeased clients must be identified prior to any contract awards regardless of an advertised CMMI benchmark claim.) Other acquiring organizations may not call their investigations "due diligence," but instead characterize them as "risk appraisals" for their proposed buy. For the purposes of this column, we'll consider these approaches to be the same.

The reasons for suggesting due diligence go well beyond stories about CMM or CMMI levels; but a conversation with John Vu of Boeing, as I began preparing this column, was too instructive to pass up. John mentioned that in his travels through Asia, he came upon a bag of rice in a store. On the bag, quality was assured by both the "Underwriters Laboratory" symbol ... and "CMM Level 3"!!!

Will Hayes, SEI's lead for appraisal quality assurance, has developed a number of questions that can be used to gain more confidence in the choice of a supplier. These are listed below. Not all will be appropriate in every case, but they may help with the due-diligence process.

1. Some of these questions were published in the March 21, 2004, *CIO Magazine*, in the sidebar to "Bursting the CMMI Hype" by Christopher Koch. For more information, go to http://www.cio.com/archive/030104/cmmi_sidebar_2.html.

Focus first on the company….

Was a SCAMPI Class A appraisal performed on the organization?

We sometimes hear of CMMI levels being claimed using various methods. Some of these are homegrown, while others have been well developed both within the United States and internationally. The SEI cannot assure the quality of appraisals done outside the scope of its training and quality assurance. Thus the only CMMI levels that the SEI can attest to are those performed using the SCAMPI Class A method by an appraisal team led by an SEI-authorized Lead Appraiser. The appraisal team of at least four individuals may be from within the organization, from outside of the organization, or a mix of both.

Other appraisal methods may be useful for determining process strengths and weaknesses within the organization and may assist in the due-diligence effort–but a level rating from anything other than a SCAMPI will not be within SEI purview.

Where are the appraisal reports?

This question provides assurance that an appraisal was actually performed. The SEI receives the appraisal disclosure statement, which is a top-level look at elements of the SCAMPI. Further, both CMM and CMMI appraisals deliver to the organization the results of the investigation, with strengths and opportunities for improvement. The breadth of organizational coverage (covered in the appraisal plan), staff interviewed, and findings (addressed in the final report) may all be valuable to a discerning acquirer. These cannot be obtained from the SEI because of our commitment to confidentiality, but these documents may well be available from the appraised organization. (The prospective customer needs to recognize the responsibility to protect proprietary information. This can be assured by executing a non-disclosure agreement.)

The SEI has committed to make elements of the SCAMPI appraisal disclosure statement available to the public if the sponsor of the appraisal requests this in writing. This approach is new, so an organization's absence from this list should not be viewed negatively. Some organizations may consider their results sensitive internal information and therefore choose not to allow public release. The current list can be seen at: http://seir.sei.cmu.edu/pars/. The Web pages only summarize appraisal results, but may give some of the answers to the questions below.

What part of the company was appraised?

Often appraisals focus on a portion of the overall enterprise. This is called the "organizational scope" of the appraisal. For various reasons, the portion of the enterprise of interest to the acquisition organization may or may not be represented by the results of the appraisal. In an ideal world, the development teams of interest will have been on actual projects that were specifically part of the appraisal. For example, a multi-site appraisal may have been performed, but the projects of interest to the acquisition organization may have been at a site not visited by the

appraisal team. In addition, most appraisals use project sampling. Thus projects of greatest interest to the acquirer might not have been interviewed, but were still considered "covered," if the appraisal team has confidence that the processes checked are "institutionalized" across that part of the organization.

Were the types of projects in the appraisal relevant to your business?

The CMMI material is indifferent to the types of development being accomplished. There can be greater confidence in the due-diligence investigation when the projects included in the appraisal are closely aligned with your needs.

Are any of the "not applicable" process areas important to your decision?

In addition to organizational scope, the appraisal allows choices in model scope. Differences in choice of coverage may be important to the investigation. For example, effective partnerships with other organizations may be essential for a large, complex development. The Integrated Supplier Management (ISM) process area (and the CMMI-SE/SW/IPPD/SS model) addresses this need. Did the organization choose to appraise its practices in this area? (Supplier Agreement Management, at Level 2, provides the initial capability, but some organizations have indicated that this area is "not applicable" as well.)

Did the organization choose to appraise only software engineering or systems engineering? If capabilities associated with integrated operations, such as those characterized under the Integrated Process and Product Development (IPPD) category, are needed in a multi-organizational effort, has the organization measured itself against the IPPD elements? Sometimes what has been excluded from coverage is as important as what has been included.

When was the appraisal?

The SEI does not require appraisals on any timeline. The U.S. Department of Defense, in earlier policy statements, described a two year "acceptability" of appraisal results. (The two year time frame is also addressed if the CMMI appraisal is being registered, a function the SEI has offered the DoD when trained government participants are on the appraisal.) The length of time since the last appraisal must be considered within the context of the nature of the business environment. In a stable domain, older results might still be relevant; but in a dynamic environment, often with organizational changes and mergers and acquisitions, confidence in continued relevance diminishes. Further, since the intent of the levels is continuous process improvement, an aging appraisal may suggest that progress may be at risk.

How long did it take the organization to move up through the levels?

This question should remind the investigator that improvement does take time and that rapid movement to high maturity levels is neither easily accomplished nor frequently observed. (See the last question on high-maturity ratings below.)

Have there been significant organizational changes since the appraisal?

Mergers deserve special mention here. Acquiring a high-maturity company does not automatically confer that level on the newly merged organization. In fact, when two high-maturity organizations merge, some time is often required to integrate process approaches across the new enterprise.

Changes in senior leadership–even without merger and acquisition activity–can cause significant change in the commitment to process discipline. Staff continuity since the appraisal could therefore be worthwhile to check.

How does the organization train its people for process excellence?

This question follows those above. Effective training programs sustain the commitment to process excellence and are essential for effective organizational growth. Without this commitment, sustainment of existing levels as well as further improvement come into question.

Additional information for the due diligence can be obtained by searching the Web for publications of the developer's strategic plans. Typically, the information in these plans merges the process improvement initiatives and required training for as long as five years.

…then on the Lead Appraiser...

Who was the Lead Appraiser?

The SEI maintains a list of all of the Lead Appraisers authorized to perform a benchmark "Class A" appraisal. Checking the name on the appraisal report with the list at http://www.sei.cmu.edu/managing/scampi.html is a simple way to build confidence.

Was the Lead Appraiser independent or from within the organization appraised?

The SEI does not require that appraisals be conducted by Lead Appraisers from outside the organization. Many organizations have determined that to avoid concerns about objectivity, they hire independent appraisers for the benchmarking appraisal. Others prefer an appraiser from within the company, but, for example, from a function charged with assuring organizational performance of operational elements of the company.

Did the Lead Appraiser guide the improvement effort?

While we understand that Lead Appraisers are experts at process improvement, extensive involvement in preparing the organization for the appraisal, and then leading the appraisal, puts the appraiser in a difficult position. From a due-diligence perspective, the investigators might wish to dig deeper if this phenomenon is evident.

…and then for "high-maturity" (Level 4 & 5) claims…

If the organization was given a high maturity rating, what experience did the Lead Appraiser have in high-maturity appraisals?

All Lead Appraisers receive standard training at the SEI (Introduction to CMMI, Intermediate CMMI, Lead Appraiser Training) and all Lead Appraisers are observed (by an SEI representative) before they are allowed to lead an appraisal on their own. However, there is a difference in appraisals of low-maturity organizations (1,2,3) and high-maturity organizations (4,5). Organizations behave differently at high maturity, and it takes an experienced appraiser to recognize the cultural differences. It is important to determine if the Lead Appraiser has had previous experience on high-maturity appraisal teams or has led a previous high-maturity appraisal (an appraisal where a level 4 or level 5 was recognized).

About the Author

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

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

Eye on Integration

Guiding Principles for Interoperability

Dennis Smith

At the SEI, we are addressing the emerging need of interoperability between software systems and systems of systems. Addressing this need is essential for the integration of large military systems as well as other software domains, including ebusiness, egovernment, mergers and acquisitions, and communications between embedded devices of systems that are traditionally considered to be hardware, such as automobiles and aircraft.

To meet this increasing need, organizations are attempting to migrate existing individual systems that employ disparate, poorly related, and sometimes conflicting systems to more cohesive systems that produce timely, enterprise-wide data that are then made available to other users. Meeting this goal has often proven to be difficult.

As Fred Brooks pointed out more than 15 years ago, the factors that make building software inherently difficult are complexity, conformity, changeability, and invisibility [Brooks 87]. With apologies to Brooks, I assert that achieving and maintaining interoperability between systems is also inherently difficult because of

- complexity of the individual systems and of the potential interactions between systems
- lack of conformity among human institutions involved in the software process and resulting lack of consistency in the systems they produce
- changing expectations placed on systems (particularly software) and the resulting volatility in the interactions
- invisibility of all of the details within and between interoperating systems

In spite of considerable effort, technical innovations aimed at improving software engineering have not successfully reduced the problems represented by these essential characteristics. Today's interoperating systems are likely more complex (because of the massive increase in the number of potential system-of-systems states) than those examined by Brooks. They exhibit less conformity (because of the increased diversity of the institutions involved in construction of the constituent parts), are more volatile (because of the need to accommodate widely diverse users) and have even less visibility (because of size, number of participating organizations, etc.)

I suggest five principles that will inform our efforts in the selection of problems to address and in the analysis of potential solutions.

1 There Is No Clear Distinction Between Systems and Systems of Systems

The distinction between a system and a system of systems is often unclear and seldom useful. By this I mean that many--perhaps a majority--of "systems" are actually systems of systems. The distinguishing factor is less where a boundary might lie and more where control lies: most systems are now created with some components over which the integrator has less than complete control. Further, most systems must cooperate with other systems over which the integrator often has no control.

It is often stated that what someone considers to be a system of systems somebody else considers a system. Thus, any given entity could be seen as a component of a larger system, as a system in itself, or as a system of systems. And, more importantly, *there usually is no top level*, because inevitably there will be some demand to include any system of systems in a more encompassing system of systems.

2 Interoperability Problems Are Independent of Domain

Most complex systems are now expected to interact with other complex systems. Regardless of domain, interoperability problems persist, and the costs of failures are huge. As an example, within the U.S. auto supply chain, one estimate put the cost of imperfect interoperability at one billion U.S. dollars per year, with the largest component of that cost due to mitigating problems by repairing or reentering data manually [Brunnermeier 99].

Our expectations are for even greater degrees of interoperability in the future, a goal that may prove difficult to achieve. The current generation of interoperable systems at least tends to encourage knowledgeable participants in the interaction—that is, the systems are being designed (or modified) specifically to interact with a particular system (or limited set of systems) in a controlled manner and to achieve predetermined goals. What is new about the future generations of interoperating systems is an emphasis on dynamically reconfigurable systems. These systems—or more accurately the services they provide—are expected to interoperate in potentially unplanned ways to meet unforeseen goals or threats.

I do not suggest that the solutions eventually found for the interoperability problems should be identical across domains. However the various communities should be aware of each other and look for commonality of high-level purpose and solution strategy—if not of solution detail—within other communities.

3 Solutions Cannot Rely on Complete Information

Classic software engineering practice assumes a priori understanding of the system being built, including complete and precise comprehension of

- assumptions or preconditions expected of the system that are required for successful use, including standards, system and environmental conditions, and data and interactions expected of other hardware, software, and users
- functionality, services, data, and interactions to be obtained from and provided to outside agents
- non-functional properties or quality of service required by the system and expected of the system from interacting components

For interoperable systems, the same information is required by all participants: the individual components (i.e., the individual systems), the links between them, and the composite system of systems. It would therefore seem that for an organization building a component (system), complete knowledge of all expectations is necessary to complete it. Unfortunately, we seldom (if ever) have such complete and precise specification even when a single system is expected to operate in isolation.

The reality is that multiple organizations responsible for integrating multiple systems into interoperating systems of systems have multiple—and rarely parallel—sets of expectations about the constituent parts as well as different expectations about the entire system of systems. The decisions that they make about the overall system of systems (e.g., assumptions, preconditions, functionality, and quality of service) are just as likely to be as incomplete and imprecise as those of organizations responsible for a single system.

Given that having complete and precise information about a system of systems (and its constituent parts) is not possible, two approaches to managing the potential chaos are evident:

1. Reduce imprecision by enforcing common requirements, standards, and managerial control.

2. Accept imprecision and apply engineering techniques that are intended to increase precision over time, such as prototyping and spiral models of development.

The first approach alone may significantly increase interoperability, but it is also highly static and does not address the inherent imprecision in the software engineering process or the legitimate variation in individual systems. The second approach is limited in a different way, since without agreeing on some level of commonality, we will not approach the levels of interoperability we require.

4 No One-Time Solution Is Possible

We live in a dynamic and competitive world in which the needed capabilities of systems must constantly change to provide additional benefits, to counter capabilities of adversaries, to exploit new technologies, or in reaction to increased understanding or evolving desires or preferences of users. Simply put, systems must evolve to remain useful.

This evolution affects both individual systems and systems of systems. Individual systems must be modified to meet unique and changing demands of their specific context and users. The expectations that systems of systems place on constituent systems will likewise change with new demands. However, the changing demands placed on a system by its immediate owners and those placed by aggregate systems of systems in which it participates are often not the same, and in some cases are incompatible.

The result is that maintaining interoperability is an ongoing problem. This was verified by SEI interviews with experts who had worked with interoperability. In some cases, desired system upgrades did not happen because of the impending effect on related systems. In other cases, expensive (often emergency) fixes and upgrades were forced on systems by changes to other systems.

To maintain interoperability, new approaches are needed to

- vet proposed requirements changes at the system and system-of-systems level
- analyze the effect of proposed requirements and structural changes to systems and systems of systems
- structure systems and systems of systems to avoid (or at least delay) the effect of changes
- verify interoperability expectations to avoid surprises when systems are deployed

New approaches to structuring systems that anticipate changes, that vet requirements and structural changes and analyze their consequences, and that verify that systems of systems perform as anticipated will help to maintain the interoperability of related systems.

5 Networks of Interoperability Demonstrate Emergent Properties

Emergent properties are those properties of a whole that are different from, and not predictable from, the cumulative properties of the entities that make up the whole. In very large networks, it is not possible to predict the behavior of the whole network from the properties of individual nodes. Such networks are composed of large numbers of widely varied components (hosts, routers, links, users, etc.) that interact in complex ways with each other, and whose behavior “emerges” from the complex set of interactions that occur.

Of necessity, each participant in such real-world systems (both the actor in the network and the engineer who constructed it) acts primarily in his or her own best interest. As a result, perceptions of system-wide requirements are interpreted and implemented differently by various participants, and local needs often conflict with overall system goals. Although collective behavior is governed by control structures (e.g., in the case of the networks, network protocols), central control can never be fully effective in managing complex, large-scale, distributed, or networked systems.

The net effect is that the global properties, capabilities, and services of the system as a whole emerge from the cumulative effects of the actions and interactions of the individual participants propagated throughout the system. The resulting collective behavior of the complex network shows emergent properties that arise out of the interactions among the participants.

The effect of emergent properties can be profound. In the best cases, the properties can provide unanticipated benefits to users. In the worst cases, emergent properties can detract from overall capability. In all cases, emergent properties make suspect predictions about behavior such as reliability, performance, and security. This is potentially the greatest risk to wide-scale networked systems of systems. The SEI recognizes that any long-term solution must involve better understanding and management of emergent properties.

These principles provide a basis for understanding interoperability. Future columns will outline how we are using the principles to identify solutions for basic interoperability problems.

References

[Brooks 87] Brooks, Fred. "No Silver Bullet: Essence and Accidents of Software Engineering." *IEEE Computer 20*, 4 (April 1987): 10-19.

[Brunnermeier 99] Brunnermeier, Smita B. & Martin, Sheila A. Interoperability Cost Analysis of the U.S. Automotive Supply Chain. National Institute of Standards & Technology, March 1999. http://www.nist.gov/director/prog-ofc/report99-1.pdf.

About the Author

Dennis Smith is the leader of the SEI initiative on the integration of software-intensive Systems. This initiative focuses on interoperability and integration in large-scale systems and systems of systems. Earlier, he was the technical lead in the effort for migrating legacy systems to product lines. In this role he developed the method "Options Analysis for Reengineering" (OAR) to support reuse decision making. He has also been the project leader for the computer-aided software engineering (CASE) environments project. Smith is a co-author of the book, *Principles of CASE Tool Integration*. He has an M.A. and PhD from Princeton University, and a B.A from Columbia University.

Security Matters

The Goal of Computer Security *or* What's Yours is Yours Until You Say Otherwise!

LAWRENCE R. ROGERS

Computer security has been, is, and will continue to be a hot topic for discussion. Newspapers frequently chronicle computer security breaches and estimates of lost revenue. Bookstores carry books that describe how to secure home and work computers against would-be intruders. Television news features depict high-profile computer security incidents and show interviews with computer system owners and sometimes even those who broke in. We're being barraged by computer security information that includes recommendations about software that we should install and other steps we should take to secure our home and office computer systems.

But when all is said and done, do we really know the problem we're trying to solve? That is, do we really know the goal of computer security?

Simply stated, the goal of computer security is this: keep your computer-based possessions yours unless and until you explicitly give them to others. This includes your computer system (CPU cycles, memory, disk space, and Internet connectivity and speed), the software you've purchased, and the files and folders you've created. As you'll soon see, most mitigation strategies discussed in those books and self-help articles on the Internet are ultimately aimed at keeping what belongs to you yours.

And this concept isn't new. It's what you've been doing for years with most of your other possessions. For example, the doors on your house have locks, and you use them. So do the windows and so does your car, and you use them too. You don't give the keys to anyone who asks for them without a really good reason. You don't leave your CD player and your CDs out for all to use. You don't store your financial or your personal medical records on your front porch.

Why then are we so willing to give up our computer possessions to anyone who wants to take them?

Back in the days before the Internet became popular and affordable, we could treat our personal computer possessions much like anything else we owned. The computer was in a room in our house and we locked our doors. Intruders who wanted access had to come to the house, break in, and take what they wanted.

We knew how to deal with that situation. We had locks and deadbolts on our doors and security systems to notify the police when someone tried to break in. Yes, there were break ins and yes, computer assets were stolen. But the incidents were few and the signs of a break in were well

understood by law enforcement. Just watch *CSI* or any other television programs of that genre to see how well understood they really are.

These days, with widespread and inexpensive access to the Internet, the only thing that's changed is that intruders can be anywhere in the world and still gain access to your computer possessions. They don't need to be where your computer is. It's like giving your credit card to the waiter or waitress at a restaurant to pay your bill and discovering that the whole world is waiting in the kitchen, prepared to make a copy of the information on your card.

And unfortunately those computer assets are not protected like your house is. That is, they don't always come with locks, and those that do can sometimes be too easily "picked" by an intruder. In fact, in some cases, your computer assets are shared automatically with anyone who comes knocking, and you have to do something to lock them. One of the challenges of using a computer is finding the locks that keep intruders out and making sure they work correctly and appropriately.

Another challenge, which may be even more significant, is *keeping* these locks working correctly. Again, we know how to deal with this type of situation. For example, if your house needs to be painted, you'd paint it after first scraping off what's loose and doing any other necessary preparatory steps. But you know that paint job won't last forever. In a few years, you'll need to do it all again. You accept this as part of the responsibility of home ownership.

With your home and office computer system, it's the same thing. You first install a piece of software, a firewall, for example, as described in Task 4 below, and then you tune it to match your Internet usage patterns. Over time, your patterns may change, as may the programs you use to access the Internet. You'll need to tune the firewall program again. Unfortunately too many home computer system owners and users get frustrated by the attention that some software requires. Rather than mastering it, they remove it. They don't accept this as part of their responsibility of home computer ownership.

Let's now return to this goal of home computer security—keeping what belongs to you yours—and look at one set of recommendations to see how they support this goal. The recommendations are taken from the Home Computer Security Guide, which is available at http://www.cert.org/homeusers/HomeComputerSecurity/.

- Task 1 – Install and use an anti-virus program (http://www.cert.org/homeusers/HomeComputerSecurity/#1) – A virus is a program that runs on your computer system without your permission. This means that when the virus runs, somebody else is using your computer possessions. A virus may also be destroying your files or disclosing them to others who aren't otherwise allowed to see them. An anti-virus program attempts to stop this from happening.
- Task 2 – Keep your system patched (http://www.cert.org/homeusers/HomeComputerSecurity/#2) – Programs that need to be patched are weak spots through which intruders can more easily gain access to your computer

possessions. Patching attempts to eliminate this kind of access. To protect your possessions, you need to keep all of the software you've purchased patched with all of the patches provided by the vendors who write that software. Each vendor will tell you where to find and how to patch the software you've purchased from them.

- Task 3 – Use care when reading email with attachments (http://www.cert.org/homeusers/HomeComputerSecurity/#3) – Email attachments that you weren't expecting are usually viruses, so the comments from Task 1 also apply here. Whether they are viruses or not, they are most often programs that run on your computer system without your permission. By using care, you are attempting to stop running unwanted programs on your computer system.
- Task 4 – Install and use a firewall program (http://www.cert.org/homeusers/HomeComputerSecurity/#4) – A firewall program attempts to keep outside access out and limits inside access to outside resources. That is, it works like your locked front door that keeps unwanted people out and your toddler in. If intruders can't get to your computer resources, they can't use them for their purposes.
- Task 5 – Make backups of important files and folders (http://www.cert.org/homeusers/HomeComputerSecurity/#5) – If a file or folder is destroyed by accident, by an intruder, or in some other way, then a backup provides another copy. You are keeping what is yours yours by having more than one copy.
- Task 6 – Use strong passwords (http://www.cert.org/homeusers/HomeComputerSecurity/#6) – These days, most computer resource access uses a login and a password. Selecting a complicated password makes it harder for intruders to access your computer resources, because those passwords are harder to guess.
- Task 7 – Use care when downloading and installing programs (http://www.cert.org/homeusers/HomeComputerSecurity/#7) – The Internet is a powerful resource for finding and using the work of others to enhance your computing resources. Programs are one example. However, not all programs on the Internet are what they are said to be. Some programs are viruses such as those described in Task 1, while others are like the email attachments described in Task 3. By doing some research before downloading and installing programs, you are trying to improve the chances that these programs are what they are said to be, will do to your computer resources what you want them to do, and will do nothing more.
- Task 8 – Install and use a hardware firewall (http://www.cert.org/homeusers/HomeComputerSecurity/#8) – A hardware firewall does the same job as the firewall program described in Task 4. It provides another layer that keeps unwanted outside access out and limits inside access to outside resources. A hardware firewall sits between your Internet connection (a cable or DSL modem) and the computer systems in your house or office. These days, a hardware firewall often comes bundled with that Internet connection hardware. Just like an airplane with two engines, where if one fails you can still fly,

the combination of a hardware and software firewall give your home and office computer systems two layers of defense against intruders.

2 Task 9 – Install and use a file encryption program and access controls (http://www.cert.org/homeusers/HomeComputerSecurity/#9) – Access controls are attributes of files and folders that limit access to only those who should have access. As a failsafe, encryption scrambles file contents so that only those who have access to a file *and* know the decryption keys can see a file's contents.

The intent of these tasks is to keep what belongs to you yours and deny access to all others. It doesn't matter whether an intruder tries to gain access by sending you a virus as an email attachment, exploiting a program that hasn't yet been patched, or accessing your system in a way that a firewall would normally prevent. They're all examples of the same fundamental concept: someone is trying to access your computer resources, and you don't want them to have that access.

Why is this important? Technology changes rapidly, as do the ways intruders take advantage of that technology. If you know the goal of computer security, you can better adapt to these inevitable technological changes. And you can better safeguard your computer resources against the inevitable intruder attacks, keeping what belongs to you yours until you say otherwise!

About the Author

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

Software Product Lines

Tiptoe Carefully or Dive Right In?

PAUL CLEMENTS

So you've decided to launch a product line. You're ready to define its scope, start working on its requirements definition, craft the architecture, build the core assets, come up with the concept of operations, unveil a new organizational structure, and all the rest. You've envisioned how your technical and managerial staff is going to turn out this product line, maintain and improve it daily, and bring home all the productivity and time-to-market gains that made the business case such a delight to read.

But there's a snag. Presumably the staff is busy doing things now, things that are making your organization a profit or carrying out its mission. Probably they're building products, products with hard deadlines–"hard" having the double meaning of difficult to meet and difficult to change–and products with a targeted market segment and customer base that you cannot just breezily dismiss in the name of progress.

So if your staff is busy right now, who's going to build the product line? Put another way, if you re-direct your staff to start building the product line, what's going to happen to all those products whose delivery has already been promised?

This problem has two solutions. To solve it, all you need to do is pick one of them. The choice is almost like deciding what to do when faced with a swim in cold water: Do you tiptoe carefully, submerging one body part at a time, or do you dive in and get it over with?

The "dive in" solution is to say that giving up or delaying products in progress is the price to be paid for all of the business gains that the product line will afford your organization. During the product line's "incubation" period while core assets are being built and production plans put in place, the organization conducts no other business, and to the cost of the product line is added the opportunity cost of missing out on the profits, market share, and customer goodwill that other products might have brought in. This solution is fairly Draconian, but there are situations where it makes sense, and we have seen it used to good effect by several organizations both large and small. If your lead time on the planned products is long or their number is few, then you might be able to turn them out as part of your new product line anyway. If you can outsource their production, this might also provide a way out. Or if you have a funding source that will see you through the lean times of product line start up (such as an R&D budget), then this approach might be viable.

The disadvantages of this approach are, of course, the money and market lost on the interim products. But the advantages are considerable. First and foremost, this approach provides the shortest path to the product line paradigm. Everyone in the development organization comes on board at once, is in synch, and shares in the experience of adopting the product line approach. And the time between product line launch and the first delivery of a member of the product line is

minimized. The organizational disruption is broad, but happens all at once and is over as quickly as it can be.

The “tiptoe carefully” solution is to stage the adoption of the product line. You can assign a small team to define the product line’s scope, craft the product line architecture, plan the construction (or purchase) of the software core assets, and define product line operations and processes. And then, once the planning and design infrastructure is in place, this small team can be augmented to start building or accumulating the core assets. While this is going on, the bulk of the organization is still busy carrying out the business of releasing products the old-fashioned way.

This approach has several advantages. First and most obvious, it lets the organization continue to release interim products. Second, each core asset that becomes available is validated by the first few products that adopt it, thus providing every asset with a built-in pilot project or two of its very own; this can increase asset quality. Third, the few-products-at-a-time approach lets the organization adopt the new paradigm in a staged manner, reducing the potential disruption from an all-at-once adoption strategy. However, there are dangers. First, small teams may produce small results, and expecting a small team to produce all of a product line’s core assets and operational processes is unrealistic. Second, if the organization has not fully committed to a product line approach, it is tempting to regard the team as a sideshow set apart from the real business of the organization. Product line starvation through neglect is a real possibility. The team needs to grow, be given resources commensurate with its growing responsibilities over time, and be staffed with the highest quality people from the product-building projects–which, of course, will make the product managers shriek in protest. Third, if there is resistance to the product line idea within the organization, this staged effort will prolong the agony of the paradigm shift. It will give the naysayers time to (at worst) entrench themselves in the old ways and find subtly subversive tactics to resist the change. And fourth, there is a danger that the core assets may be designed with only the early products in mind and founder when applied to subsequent products.

And the approach comes with a cost. The cost is delaying the full benefit of the product line approach until the product line has fully blossomed. The cost is the lower productivity and the higher time to market that could have been remedied with a more aggressive approach.

The incremental approach usually works best if the organization has many products in the delivery pipeline with relatively low lead times, and cannot afford to simply shut down production until the first product line member is ready to be released.

So how does the incremental approach work? Members of one organization we know staged their product line adoption as follows:

- They formed a small steering group to define the product line scope and craft the architecture for the product line. The architecture, not surprisingly, was a layered one. The bottom-most layer provided portability across the variety of hardware platforms that were included in the scope. The next layer up provided a set of services that were common across products; that is,

every service in this layer would be used by at least two products in the intended scope.

- They identified three upcoming products that would become the first pilots to use the product line's core assets. These products had sequential delivery dates with gaps of six months to a year in between.
- For component-design work, they staffed the software architecture group with part-time members of those three products' development teams and members from other product groups in the organization.
- They began implementing the software in the lowest layer of the architecture, the portability layer, targeting its completion to accommodate the schedule of the first pilot product, which used it.
- Once this was complete, they began to implement services in the second layer that the second pilot would use. And the second pilot, then, was able to use the whole portability layer, and those parts of the common-services layer that they needed and that were ready in time.
- Finally, they completed the common-services layer, and the third pilot was able to use all of that.

Figure 4 illustrates this approach. This organization staffed and resourced its steering group at near-starvation levels, but its fundamental approach was sound. There was some discomfort among product line "purists" that the products weren't really members of the product line, because they were not compelled to use the full set of core assets. But in fact, they were members. Each one used a shared set of core assets–just not the *same* set. But more important than wrangling about definitions is to observe that *each product was better off for using the available core assets than it would have been otherwise*. Even the first pilot product, which used the fewest of the core assets, gained portability across hardware platforms, something it would not otherwise have enjoyed. And it became a member of the product line family, in that the support of that part of its software was the job of the core assets group, and this product joined a supporting suite of products using that layer. Once the product line is established, each of the pilot products can be reengineered to come into full alignment with the product line and adopt those core assets that were not yet ready. Developers of each product must make a business case for the product either fully joining the product line fold, or continuing its prodigal ways.

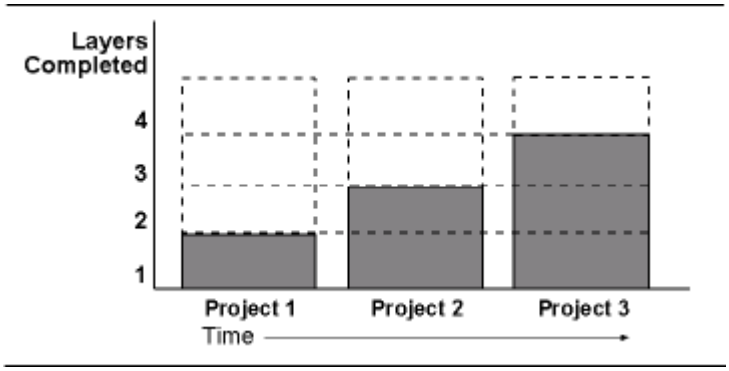

Figure 4: Phasing in a four-layer architecture

Whether you dive in all at once or step in lightly and carefully, deciding how to turn out your first products is one of the first decisions you'll have to make when launching and institutionalizing a product line. Once you're in, the water will feel much better.

About the Author

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

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

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

What's New

Security Changes Everything

WATTS S. HUMPHREY

The last column began a discussion of software quality. It stressed the growing importance of security as a motivating force for software quality improvement and noted that the recent Sarbanes-Oxley Act is changing management's attitudes about software security and quality. The quality of today's software products falls far short of what is needed for safe and secure systems. We can no long afford to power our critical corporate and national infrastructure with defective software. Since security is such an important concern for many businesses, the pressures for improvement will almost certainly increase.

In this column, I discuss why software quality has not been a customer priority in the past and how that attitude has influenced the quality practices of our industry. I then discuss the changes we will likely see in these quality perceptions and what our customers will soon demand. Finally, I describe the inherent problems with the test-based software quality strategy of today and why it cannot meet our future needs. In the next column, I will outline the required new quality attitude and the prospects for success in our quest for secure and high-quality software.

The Quality Attitude

Even though current industrial-quality software has many defects, quality has not been an important customer concern. This is because even software with many defects works. This situation can best be understood by examining the "Testing Footprint" in Figure 1. The circle represents all of the possible ways to stress test a software system. At the top, for example, is transaction rate and at the bottom is data error. The top left quadrant is for configuration variations and the bottom left is for the resource-contention problems caused by the many possible job-thread variations in multiprogramming and multiprocessing systems. At the right are cases for operator error and hardware failure. The shaded area in the center represents the conditions under which the system is tested.

Since software systems developed with current industrial quality practices enter test with 20 or more defects per thousand lines of code, there are many defects to find in test. Any reasonably well-run testing process will find and fix essentially all of the defects within the testing footprint. If this process is well designed, if it covers the ways most customers will use the system, and if it actually fixes all the defects found, then, as far as these users are concerned, the system would be defect free.

Trying Harder

While current quality practices have been effective in the past, this test-based quality strategy cannot meet our future needs for three reasons.

First, testing is a very expensive way to remove defects. On average, it takes several hours to find and fix each defect in test and, because of the large numbers of defects, software organizations typically spend about half of their time and money on testing.

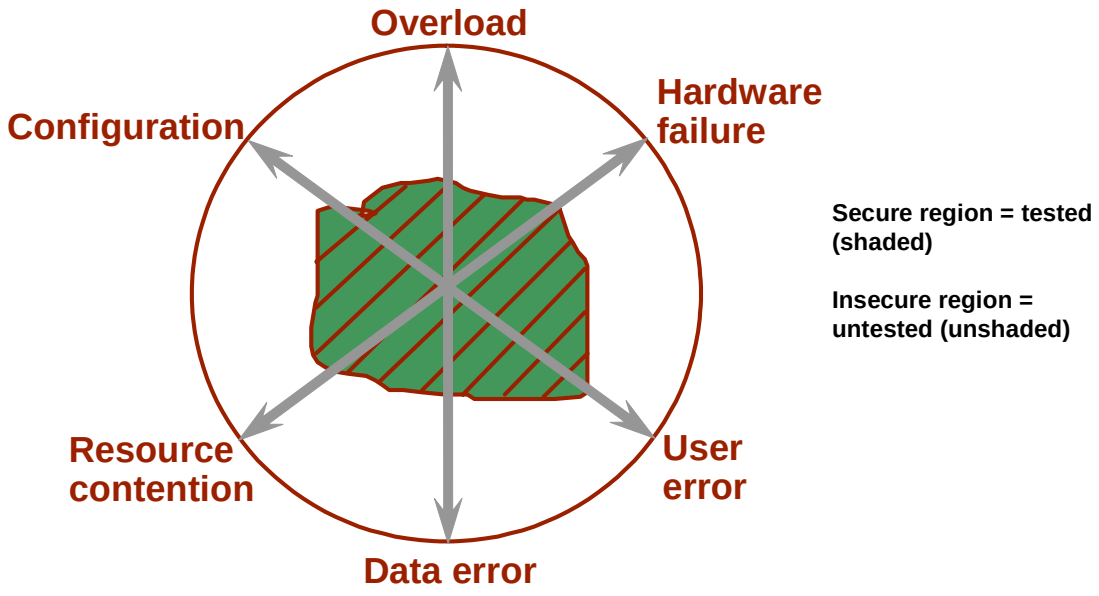

Figure 1: The Testing Footprint

The problem is that testing is being misused. Even with all the time and money software organizations spend on testing, they still do not produce quality products. Furthermore, the evidence is that even more testing would not produce significantly better results. The problem is not that testing is a mistake. Testing is essential, and we must run thorough tests if we are to produce quality products. In fact, testing is the best way to gather the quality data we need on our software products and processes.

The second reason that testing cannot meet our quality and security needs stems from the nature of general-purpose computers. When these systems were first commercially introduced, the common view was that only a few would be needed to handle high-volume calculations. But the market surprised the experts. Users found many innovative applications that the experts had not imagined. This user-driven innovation continues even today. So, in principle, any quality strategy that, like testing, requires that the developers anticipate and test all the ways the product will be used simply cannot be effective for the computer industry.

The third reason to modernize our quality practices is that we now face a new category of user. In the past, we developed systems for benign and friendly environments. But the Internet is a different world. Our systems are now exposed to hackers, criminals, and terrorists. These people want to break into our systems and, if they succeed, they can do us great harm.

The reason this will force us to modernize our quality practices is best seen by looking again at the figure. Since the test-based quality strategy can find and fix defects only in the testing footprint, any uses outside of that footprint will likely be running defective code. Since over 90% of all security vulnerabilities are due to common defects, this means that the hackers, criminals, and terrorists have a simple job. All they have to do is drive the system's operation out of the tested footprint. Then they can likely crash or otherwise disrupt the system.

One answer to this would to be to enlarge the tested footprint. This raises the question of just how big this footprint can get. A quick estimate of the possible combinations of system configuration, interacting job threads, data rates, data errors, user errors, and hardware malfunctions will likely convince you that the number of combinations is astronomical. While there is no way to calculate it precisely, the large-system testing footprint cannot cover even 1% of all the possibilities. This is why modern multi-million line-of-code systems seem to have a limitless supply of defects. Even after years of patching and fixing, defect-discovery rates do not decline. The reason is that these systems enter test with many thousands of defects, and testing typically finds less than half of them. The rest are found by users whenever they use the system in ways that were not anticipated and tested.

The Quality Requirement

Indeed, "security changes everything." Software pervades our lives. It is now common for software defects to disrupt transportation, cause utility failures, enable identity theft, and even result in physical injury or death. The ways that hackers, criminals, and terrorists can exploit the defects in our software are growing faster than the current patch-and-fix strategy can handle. With the testing strategy of the past, the bad guys will always win.

It should be clear from this and the last column that we must strive to deliver defect levels that are at least 100 times better than typically seen today. Furthermore, we need this for new products, and we must similarly improve at least part of the large inventory of existing software. While this is an enormous challenge and it cannot be accomplished overnight, there are signs that it can be done. However, it will require a three-pronged effort.

First, we must change the way we think about quality. Every developer must view quality as a personal responsibility and strive to make every product element defect free. As in other industries, testing can no longer be treated as a separate quality-assurance or testing responsibility. Unless everyone feels personally responsible for the quality of everything he or she produces, we can never achieve the requisite quality levels.

Next, we must gather and use quality data. While trying harder is a natural reaction, people are already trying hard today. No one intentionally leaves defects in a system, and we all strive to produce quality results. It is just that intuition and good intentions cannot possibly improve quality levels by two orders of magnitude. Only a data-driven quality-management strategy can succeed.

Finally, we must change the quality attitude of development management. While cost, schedule, and function will continue to be important, senior management must make quality ***THE*** top priority. Without a single-minded drive for superior quality, the required defect levels are simply not achievable.

Stay tuned in, and in the next column I will discuss the required quality attitude.

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, Dan Wall, and Alan Willett.

In closing, an invitation to readers

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

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

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 several books. His most recent books are *Introduction to the Team Software Process* (2000) and *Winning With Software: An Executive Strategy* (2002). 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.

Microsoft's Pilot of TSP Yields Dramatic Results

KELLY KIMBERLAND

In today's fast-paced software industry, organizations struggle to keep up with the demand for quickly produced software. Faced with reductions in budget and staffing, companies must make sacrifices to save time and money. Software quality suffers most in this schedule-driven market. For more than 20 years, the SEI has created and promoted methods that help organizations efficiently develop high-quality software. Microsoft, one of the world's largest software vendors, recently piloted a project using the SEI's Personal Software Process℠ (PSP℠) and Team Software Process℠ (TSP℠) with a team of software developers to change behaviors, improve processes, and deliver better software.

SEI Fellow Watts Humphrey has stated that the problem with defective software is not the programmers; it's how they are managed and trained. "The entire software culture continues to follow the expensive and ineffective test-based software quality strategy," Humphrey says. "Changing this mindset is difficult and time consuming. Microsoft's willingness to pilot the TSP demonstrated senior management's commitment to new quality and team-management strategies."

Microsoft Program Manager Carol Grojean was the team leader of the original pilot, and since then has become a TSP coach and mentor. She and Noopur Davis, a member of the SEI's technical staff, recently presented Microsoft's results at this year's Software Engineering Process Group (SEPG) Conference in Orlando, Florida.

TSP Builds Teams

After the original pilot, Grojean launched another Microsoft team using TSP. Prior to using the TSP, this team had just finished a 2.5 year project with an unsuccessful ending. The team had ten developers who were frustrated with the project, their jobs, and each other. The team was operating as individuals, not as a team. Now, after two weeks of intensive TSP training, they were being asked to spend a week in a room together making decisions as a team, not as individuals.

While a normal workday during a TSP launch is usually around nine hours, the first day of this TSP launch lasted more than 14 hours. The team members were reluctant to make goals and commitments, and they were concerned about management seeing their goals. When the second day of the launch lasted 15 hours, it was clear that the team members needed time to overcome the barriers built in the past few years. "They didn't know how to behave any differently," Grojean says. "They were reluctant to commit to clear, measurable goals for fear that those goals would be used against them if they failed. It is not easy for a team to realize that failing at a goal was better than not having a goal at all."

By the third day, the team was exhausted from the long days. Then, an amazing thing began to happen: the team members realized that they would accomplish nothing if they continued to argue, and they began to work as a team. Though individuals were still defensive and hesitant about the process, they became more engaged. They knew they needed to come together to make the project work.

By the end of the four-day launch, the group was truly a team. “It is such an amazing transformation–we started with ten individual contributors and ended up with a well-organized, cooperative team,” Grojean says. She added that the transformation from individual contributors to one team is now her favorite part of the launch process.

The TSP accomplishes these remarkable results by combining proven project and quality management principles with well-known team building and coaching methods.

TSP Reduces Defects, Improves Quality

Watts Humphrey has said that even experienced and capable software developers inject one defect for every nine lines of code they write. For example, today’s common software-development practices result in 110,000 defects injected into a typical million line-of-code system, most of which must be found and fixed before the software can be released. “On average, poor quality software at customer delivery has about five defects per thousand lines of code (KLOC),” says Humphrey.

During TSP training, Microsoft developers reduced unit-test defects dramatically from more than 25 defects/KLOC to about 7 defects/KLOC. “Unit-test defects were reduced because these developers were removing most defects before unit test,” says the SEI’s Noopur Davis. She adds that “the goal of the TSP is to change how developers view testing: instead of thinking of testing as a defect-removal activity, developers start thinking of testing as validation of not just product functionality, but also as validation of the quality of the software. Developers strive to remove as many defects as possible from the software before testing, when defects are the easiest to find and fix.”

People are sometimes skeptical about training results, and want to see data from real projects to be convinced. Microsoft’s pilot project continued to show the quality improvements seen in training. The project received a high score on the process quality index (PQI), a process quality measure that TSP teams use as an indicator of product component quality. A PQI of 0.4 or greater on a scale of 0.0 to 1.0 is considered to be an indicator of a high-quality component. Microsoft’s pilot project had a PQI of 0.52.

Microsoft teams, like other software development teams, spend 40 – 60% of their total development in test because they use that time to find and fix defects within the product. However, because Microsoft’s TSP pilot team spent time in earlier removal activities like personal reviews and team inspections, they spent only 11.5% of the total project effort in test. “The focus on

spending more time up front is an important behavior change; the team is using the process to plan their schedule rather than letting the schedule drive their plan," Davis says.

Ultimately, the pilot team ended up delivering its products to test on schedule and with high quality. "Microsoft results compared favorably with other TSP teams, which have delivered products with an average defect density of just 0.06 defects/KLOC," says Davis. Additionally, since the code was truly complete at the end of the coding phase, the developers were able to advance to the next development cycle of the product rather than having to stick around during the testing to fix defects. This resulted in a 35% project cost savings. While the team delivered to test on the original committed date, the sharp reduction in test time made the team members available for other work earlier than planned.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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

Calculating Return on Investment for Software Product Lines

PAUL CLEMENTS

Product line engineering has become a widely used methodology for the efficient development of complete portfolios of software products.[1] The core idea of the approach is to regard the development of a set of products as a single, coherent development task. This approach has been shown to produce orders of magnitude economic improvements over one-at-a-time software system development.

The SEI is helping organizations understand the benefits of product line engineering. Often, the first thing a manager in an organization considering product lines wants to know is the return on investment (ROI) for changing the company's development strategy. Until now, however, most of the economic arguments supporting the switch to product lines have been based on data derived from individual case studies,[2, 3] convincing arguments based on rationality and simplistic cost curves.[4] Existing models of product line development costs for the context of reuse can be applied only in a restricted way, as product line development makes some fundamental assumptions that are not reflected in these models.

A Product Line Economic Model

Recently, a small project team at the SEI has begun to work on the problem of calculating the cost benefit of the product line approach. The model is based on four cost functions:

- $C_{org}()$ is a function that, given the relevant parameters, returns the cost to an organization of adopting the product line approach for its products. Such costs can include reorganization, process improvement, training, and whatever other organizational remedies are necessary.
- $C_{cab}()$ is a function that, given the relevant parameters, returns the cost to develop a core asset base suited to satisfy a particular scope. C_{cab} differs from Cunique in that it must take into account the cost of performing a commonality/variability analysis, the cost of designing and then evaluating a generic (as opposed to a single-system) software architecture, and the cost of

1. Special Issue on Software Product Lines. IEEE Software 19, 4 (July/August 2002). http://csdl.computer.org/comp/mags/so/2002/04/s4toc.htm
2. Clements, P. & Northrop, L. Salion, Inc.: A Software Product Line Case Study (CMU/SEI-2002-TR-038, ADA412311). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tr038.html
3. Knauber, P.; Bermejo, J.; Böckle, G.; Leite, J.; van der Linden, F.; Northrop, L.; Stark, M.; & Weiss, D. "Quantifying Product Line Benefit," Proceedings of the Fourth International Workshop on Product Family Engineering, Bilbao, Spain, October 3-4, 2001. Heidelberg, Germany: Springer-Verlag, 2001.
4. Weiss, D. & Lai, C. Software Product-Line Engineering. Reading, MA: Addison-Wesley, 1999.

developing the software so designed. Ccab can be invoked to tell us the cost of developing a core asset base where none currently exists, or it can be invoked to tell us the cost of deriving a desired core asset base from one or more already in place.

- $C_{unique}()$ is a function that, given the relevant parameters, returns the cost to develop unique software that is not based on a product line platform. The result might be a complete product or it might be the unique part of a product whose remainder is built atop a product line core asset base.
- $C_{reuse}()$ is a function that, given the relevant parameters, returns the development cost to reuse core assets in a core asset base. C_{reuse} includes the cost of locating and checking out a core asset, tailoring it for use in the intended application, and performing the extra integration tests associated with reusing core assets.

These cost functions are not mathematically rigorous, but they do let us break down the costs into more manageable analytical chunks. They don't actually "return" values, but let us perform thought experiments, use legacy data, or invoke more conventional cost models on smaller pieces of the problem. Hence, the cost of developing a software product line from scratch can be expressed as

$$C_{org} + C_{cab} + \sum_{i=1}^{n}(C_{unique}(i)+C_{reuse}(i))$$

where n is the number of products in the product line. Other situations can be similarly formulated, and the cost of building and managing the evolution of a product line can be compared to the cost of building the same systems from scratch. ROI is simply cost savings of doing things the new way divided by the cost of investment – which in this case is usually $C_{cab} + C_{org}$.

We hope this simple model will prove useful in helping software development managers or corporate decision-makers understand the cost benefits they might expect when switching to the product line approach. Our team is working to expand the model to include the non-cost benefits of product line engineering, such as higher quality and decreased time to market. We are also seeking real-world opportunities to put the model to use.

Third Software Product Line Conference (SPLC 2004)
Boston, MA
August 30-September 2

Software product lines have emerged as an important development paradigm, with an active community of practitioners. The Third Software Product Line Conference (SPLC 2004), to be held August 30-September 2 in Boston, is a leading forum for researchers and practitioners working in the field. The conference will feature technical papers, topical panels, a rich selection of tutorials and workshops, demonstrations, birds-of-a-feather meetings, and new inductions into the Software Product Line Hall of Fame.

For more information about SPLC 2004, visit http://www.sei.cmu.edu/SPLC2004.

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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

CERT®/CC Instrumental in National Security Effort

MINDI MCDOWELL

In September 2003, the U.S. Department of Homeland Security announced the creation of the US-CERT, a joint effort between the Department of Homeland Security's National Cyber Security Division (NCSD), the CERT® Coordination Center (CERT/CC), and the private sector to improve the nation's cyber security capability. US-CERT will build on CERT/CC capabilities to help prevent cyber attacks, protect systems, and respond to the effects of cyber attacks across the Internet.

Goals

US-CERT's mission includes

- identifying, analyzing, and reducing threats and vulnerabilities
- disseminating threat warning information
- coordinating incident response
- providing technical assistance in continuity of operations and recovery
- serving as a national focal point for the public and private sector regarding cyber security issues

The goal of US-CERT is to reduce the frequency and severity of cyber attacks by building collaborative partnerships among organizations that participate in cyber watch, warning, and response functions. The organizations include computer security incident response teams, information sharing and analysis centers, managed security service providers, information technology vendors, and security product and service providers. The partnerships will strengthen national and international efforts, with each organization offering its own unique resources and expertise. Jeffrey Carpenter, manager of the CERT/CC, notes, "Today, most of the interaction between organizations is informal. But organizations are coming to realize that they have to work together on Internet security. We're much more powerful together than individually."

Products

As a national resource, US-CERT must serve a diverse audience that includes technically sophisticated users, inexperienced users, executives, and policymakers. This challenge extends to the products that US-CERT is offering. The CERT/CC and NCSD have jointly developed a new National Cyber Alert System, a series of information products targeted at home and non-technical corporate users and technical experts in businesses and government agencies. There are four products available.

- **Technical Cyber Security Alerts**
  These technical alerts, written primarily for system administrators, provide timely information about current security issues, vulnerabilities, and exploits, including potential impact and action required to mitigate threats.
- **Cyber Security Bulletins**
  A resource for technical users, these bulletins summarize security issues and new vulnerabilities and include information about patches and workarounds.
- **Cyber Security Alerts**
  Similar to Technical Cyber Security Alerts, these alerts also provide timely information about current security issues, vulnerabilities, and exploits, but they are written with language and advice suited to non-technical users. Cyber Security Alerts are published when there is an issue that affects the general public.
- **Cyber Security Tips**
  A resource for non-technical home and corporate computer users, Cyber Security Tips describe and offer advice about common security issues. They are published bi-weekly.

CERT/CC Celebrates 15 Years

In November 2003, the CERT Coordination Center celebrated its 15 year anniversary. Established by the Defense Advance Research Projects Agency (DARPA) in 1988, the CERT/CC had multiple functions:

- responding to computer security threats
- helping other organizations respond to emergency situations
- serving as a focal point for identifying and fixing security vulnerabilities
- assessing the security of systems
- increasing user awareness about security

Over the years, as the work of the CERT/CC has evolved with society's increased reliance on technology, the organization has remained committed to its efforts to secure networked systems. The CERT/CC has helped foster the creation and operation of many other response organizations around the world and has established strong relationships with vendors, government agencies, and security experts. Staff members actively participate in a variety of organizations committed to security and survivability and are regularly asked to testify before Congress.

At the anniversary celebration, Rich Pethia, the director of the CERT/CC, looked to the future: "While there is much work yet to be done, I am confident that the professionals in this global watch and warning network will continue to find increasingly effective ways to deal with the new challenges we are sure to face."

These products are available on the US-CERT Web site, where there are also instructions for how to subscribe to National Cyber Alert System mailing lists.

For more information, contact—
Rich Pethia

Phone
412-268-7739

Email
rdp@cert.org

World Wide Web
http://www.us-cert.gov/

SEPG 2004 Showcases Enterprise Process Improvement in Orlando

LAINE TOWEY

This year's Software Engineering Process Group (SEPG[SM]) 2004 conference was held on March 8-11, in Orlando, Florida, and attracted more than 1900 attendees, the highest number since SEPG 2001 in New Orleans. The conference theme was "Enterprise Process Improvement: Better Products, Dependable Services, Cultures of Excellence," and the presentations, tutorials, and panel discussions chosen for this year's conference all helped to advance this theme.

The conference attendees came from industry, defense agencies, and academic institutions, but all shared an interest in process improvement. A total of seven tracks were made available to this year's attendees: two dealing with process improvement, two with Capability Maturity Model Integration® (CMMI®), a special topics track, a Team Software Process[SM]/Personal Software Process[SM] (TSP[SM]/PSP[SM]) track, and a measurement and analysis track. SEPG attendees could select from more than 90 presentations and tutorials, which had been reviewed and recommended by 94 volunteers from the process community.

The keynote speakers at SEPG also came from diverse backgrounds, but each speaker emphasized his profession's commitment to process improvement in his keynote presentation. Dr. Jay Apt from Carnegie Mellon University's Electricity Industry Center discussed a failed space mission in "Ariane 501: Software Engineering Failure or Systems Engineering Failure?" Mr. Kevin Carroll, from the U.S. Army's Program Executive Office Enterprise Information Systems presented the unique process challenges in his organization and explained how process improvement has helped.

The conference location figured heavily in Dr. Thomas McCann's presentation. In "Behind the Magic: Science, Technology, and Process," McCann, the senior vice president for engineering at Walt Disney Imagineering, discussed process in his corporation and reminded attendees that process improvement happens in every organization. Dr. Thomas Wagner, from Robert Bosch, GmbH, continued this theme with his presentation, "Bringing Software on the Road," which discussed the auto industry and process.

"I think that people enjoyed being in Orlando and what it had to offer, especially to families," said Bill Peterson, the head of the SEI's Software Engineering Process Management Program. "I was pleased to see an emphasis on products and services, not just products and services delivered by the SEI, but by our SEI Partners. The tracks were more focused on CMMI and less on the Software CMM, and more on TSP but also measurement and analysis."

Dr. Caroline Graettinger, this year's program chair, agreed. "As process technologies have evolved over the years, so has the SEPG conference evolved. We are moving to encompass a broader set of process topics that pertain to the entire software-intensive enterprise."

Other highlights of this year's conference included the delivery of the first Japanese translation of CMMI. Representatives from IBM Japan and Japan's Information-Technology Promotion Agency were in Orlando to present Peterson with a printed copy of the model, which is available on the SEI's Web site at *http://www.sei.cmu.edu/cmmi/translations*. This is the first translation of the CMMI, but other translations are on the way, including a Traditional Chinese translation of the CMMI models.

Many SEI members came to Orlando and participated in new member activities at SEPG. In lieu of a special breakfast, this year's Member Assembly was converted into a luncheon, complete with guest speakers and an awards ceremony. Members also received access to a member lounge at the conference, which included daily breakfast.

"The conference was great for a number of reasons," said Julia O'Rorke, the SEPG 2004 conference manager. "One was the location, and another was the program, which was focused on enterprise process improvement related to systems and software engineering. We designed a program that was accessible to a lot of professionals, and I think they appreciated that."

Peterson supported this idea. "Generally, every year, the conference gets better. More attendees is one measure, but also I think stronger name recognition, stronger presentation content, and better understanding by people who've actually done process improvement using the newer products such as TSP, CMMI, and Six Sigma also contribute. I think that it gets better every year because more people have more experience with process improvement products and want to share those experiences and talk about them."

Over 80% of SEPG 2004 attendees gave the conference a rating of "excellent" when asked about their overall satisfaction with the conference, and almost half of all attendees surveyed said that they would be attending next year's SEPG conference. Attendees' comments were overwhelmingly positive, and reflected the conference effort to make process improvement accessible for everyone. "Being relatively new to process improvement, there was a lot of good information to absorb," remarked James Hastings, of Aspen Systems Corporation in Rockville, MD. Other attendees commented on the "diversity in complexity of levels" at the conference. "The presentations and tutorials were generally excellent," commented Lyn Elliott Dillinger of Process Improvement Associates. "I'm going home with really good ideas."

For additional information on SEPG, including copies of presentations given at the conference, the SEPG 2004 CD-ROM is available at http://www.sei.cmu.edu/sepg for $50. Information about SEPG 2005, which will be held in Seattle, Washington, is available at the Web site. The conference theme will be "Deliver Winning Products Through Process Improvement."

For more information, contact—

Customer Relations

Phone
412-268-5800

Email
customer-relations@sei.cmu.edu

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