Carnegie Mellon
Software Engineering Institute

news @ sei interactive

Volume 6 | Number 2 | Second Quarter 2003

In This Issue

columns

Features

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

news @ sei interactive

From the Director

These will be my last words to you from the pages of *news@sei*. Later this summer, I will become a vice president of Georgia Institute of Technology and the director of the Georgia Tech Research Institute. I will also be a professor in the School of Industrial and Systems Engineering. While this is a tremendous professional opportunity for me, I will miss the opportunity I have had to lead such a magnificent organization and to serve you.

I am very proud of the SEI and all we have accomplished during the past seven years. These accomplishments (such as creation and adoption of the CMMI, outstanding successes with the TSP, continued impact and leadership of the CERT/CC, the introduction of the product line framework, and many others that have been chronicled in *news@sei*) were only possible because of the willingness of the community to partner with us and to share your ideas and insights. I want to take this opportunity to thank you for your support of the SEI, but more importantly for your partnering with us to advance the practices of software engineering and cyber security.

Carnegie Mellon Provost Mark Kamlet is chairing the search committee. Until the search committee finds a new director, the SEI will be in the capable hands of Angel Jordan, SEI founding father, emeritus professor and former provost of Carnegie Mellon University, and member of the National Academy of Engineering. Angel is thoroughly committed to guiding the SEI in its core objectives: accelerating the introduction and widespread use of high-payoff software engineering practices and technology, maintaining a long-term competency in software engineering and technology transition, enabling government and industry organizations to make measured improvements in their software engineering practices, and fostering the adoption and sustained use of standards of excellence for software engineering practice. Angel knows and respects the current SEI leadership team. So with the continued outstanding support of Clyde Chittister (SEI Chief Operating Officer) and the current leaders of the SEI's administrative units, the SEI will be in great hands.

I have encouraged my SEI colleagues to stay focused on the SEI strategy and to always seek to improve, remembering that improvement requires change. I believe that a change in leadership at this time is a healthy change and it will help the SEI accomplish even more in the future. The SEI's new director will likely have a different style than I have, but I'm sure will be as passionate about the SEI and its service to the community as am I.

Again, thank you for your support these past seven years. I hope our paths cross again!

Stephen E. Cross
SEI Director and CEO

The Architect

Introducing Predictable Assembly from Certifiable Components (PACC)

Kurt Wallnau

In September 2002 the SEI launched a new initiative, Predictable Assembly from Certifiable Components (PACC). This new initiative is developing technology and methods that enable software engineers to predict the runtime behavior of assemblies of software components from the properties of those components. This requires that the properties of the components are rigorously defined and trusted and can be certified by independent third parties.

PACC builds on past and, in some cases, ongoing SEI research in software architecture and in the use of commercial off-the-shelf (COTS) software. The work reflects clear industry trends toward greater use of software-component technology and increasing concern with the quality of software systems.

Nothing serves so well as an illustration, so that's where I'll start. Then I'll introduce the key principles and technologies underlying our approach, which is called prediction-enabled component technology (PECT). I'll wrap up by describing what we're currently doing, open challenges, and next steps.

A Simple Illustration

In PACC our concern is to predict the runtime behavior of assemblies of components. By runtime behavior we mean any behavior of a system that is directly or indirectly observable on the executing system. For convenience, we refer to any such observable behavior as a *runtime property*.

In this illustration I'll use execution latency, i.e., the time it takes an assembly to perform a task, as the runtime property we wish to predict (our work on automatic verification of reliability properties through model checking will be the subject of a future article). The illustration is drawn from a proof of feasibility of PACC for power transmission and distribution (see ***Predictable Assembly of Substation Automation Systems: An Experiment Report*** for details).

A power substation serves several purposes, among which is protection and control of primary equipment, such as transformers, circuit breakers, and switches. Our task is to develop, from software components, a controller for a high-voltage switch. One function of the controller is to provide an interface that allows operators to manually open and

close the switch. We wish to predict the time it takes a controller to process operator requests, and the time it takes the controller to report on the change in switch status.

The illustration in Figure 1 presents the gestalt of the software engineering task in terms of PACC. We assume that a set of software components already exists, and that the service time of these components (defined as the time it takes the component to do its work, assuming no blocking or preemption) has been obtained (1 in the figure). The software engineer selects a set of candidate components, and composes their specifications to produce a model of the controller assembly, which is analyzed and from which latency is predicted (2 in the figure). If the predicted latency satisfies requirements, the components (rather than their specifications) are composed and the resulting assembly is deployed. Predictions are only predictions if there is a possibility that they are wrong, so some validation is required of the deployed assembly (3 in the figure).

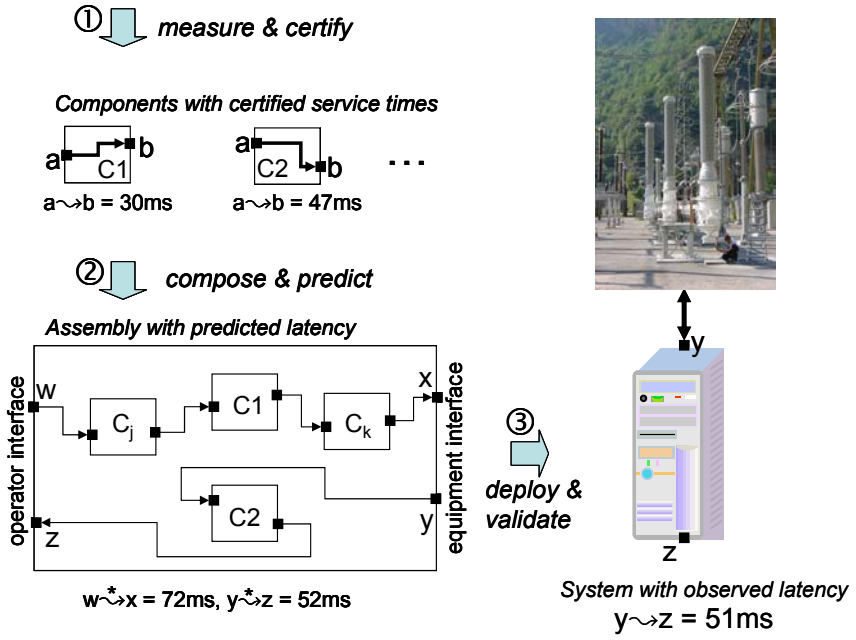

Figure 1: A Predictable Substation Assembly

This much might have been guessed from the name of the PACC initiative. However, technology being developed by the SEI aims to increase the level of automation in the assembly, prediction, and composition processes, and to provide an objective and quantified basis for trusting component properties and the predictions that are based on these properties. In particular, using this example:

Latency prediction for user-selected controller operations (e.g., from arrival of an operator request on *w* until the switch is signaled on *x* in Figure 1) is computed automatically from assembly specifications.

The analysis model used to make latency predictions defines precisely what runtime properties of components must be known and how these properties are specified and obtained. Thus, the properties of components that must be trusted are precisely those that enable predictions of assembly runtime behavior.

The assumptions underlying the analysis model about how components interact with their environment and with each other are made explicit. Assemblies are well formed if they satisfy these assumptions. Well-formedness is checked automatically—thus, assembly behavior is predictable by construction.

The accuracy and reliability of analysis-model predictions is objectively validated using statistically sound sampling and measurement. These qualities of predictions are certifiable properties of the analysis model itself, and are specified as a confidence interval for predictions—e.g., 9 out of 10 predictions will have an upper error bound of 3% with a 95% confidence.

We are concerned with more than just the timing properties of assemblies—e.g., reliability (an area of current work) and security (an area for future work). Therefore, the technology being developed by the SEI to enable PACC can be applied to many analysis models.

Prediction-Enabled Component Technology—Just the Basics

Our approach to achieving the above objectives is to use prediction-enabled component technology (PECT). Here I will limit the description of PECT to the simple core ideas. Readers interested in the more comprehensive treatment are referred to *Volume III: A Technology for Predictable Assembly from Certifiable Components*.

As its name suggests, a PECT is an enhanced component technology. What is a component technology? There is no answer to this question that won't provoke an argument, any more than there is a universally agreed-upon answer to the question What is a component? Nonetheless, there is growing agreement on the following rough definition:

- A software component is an implementation, ready to execute on some (possibly virtual) machine, with well-defined interfaces that enable third-party composition (roughly, integration with other components).

- A component technology is a component model and runtime environment where

    - the component model specifies what interfaces a component must provide, and how components are allowed to interact with one another and their runtime environment

    - the runtime environment is a container in which component behavior executes and in which components interact. The runtime environment may also provide useful services—persistence, transactions, etc.

Regular readers of this column will notice the similarity between this definition of component model and the usual definition of architectural style (or pattern) as a collection of *component types* and their allowable *patterns of interaction*. Even though they may differ in many respects, a component model and architectural style both specify invariants that must be satisfied by any instance of that model/style. These invariants are exactly those "well-formedness" rules that we impose on component assemblies to ensure that they can be analyzed, and therefore to ensure their predictability.

Seen in this light, a component technology can be thought of as an infrastructure for designing, developing, and deploying applications that adhere to a particular architectural style. The infrastructure does restrict the freedom of developers and designers, but in compensation it enforces design and implementation invariants that, in this case, ensure predictability. The tradeoff between restricted freedom and predictability has been seen before—in the development of strongly typed programming languages, now considered an essential element of modern software engineering practice. The long-awaited shift to a higher level of abstraction—from functions and classes to components—is underway.

Figure 2 provides a concise description of the structure of a PECT in the Unified Modeling Language (UML). Even readers not fully familiar with UML should be able to use this model to follow along, as there are only a few points to make that have not already been stated explicitly.

1. The *construction language* in Figure 2 could well be any architecture description language that supports primitive notions of component and connector. The

notation and tools we use (of our own making) will likely be replaced by the first usable UML 2.0 tools.

2. A PECT can and generally will support several analysis models, each of which is "packaged" in its own automated reasoning framework. An interpretation defines an automated translation from assemblies of components specified in the construction language to the reasoning framework.

3. PECT as a general concept does not depend on any component technology, but any PECT will. An abstract component technology specifies the invariants imposed on by a specific component and a collection of reasoning frameworks (each which may impose its own additional invariants).

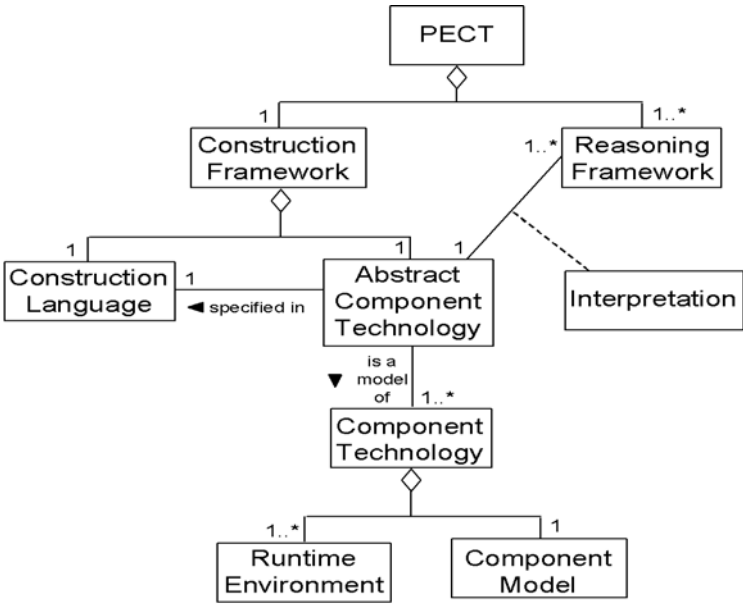

Figure 2: A UML Model of the Structure of a PECT

The SEI PACC initiative has developed several prototype demonstrations of PECT, and as a result we are gaining experience in the methods and technology infrastructure needed to achieve the levels of automation and trust we are seeking.

Status and Challenges

We are developing methods and tools that will enable the software industry to introduce predictable assembly from certifiable components into practice. Although the initiative is less than a year old, we are already working with an industrial sponsor, the ABB Group (Asea Brown Boveri, Ltd.), to demonstrate PECT feasibility in challenging industrial settings, currently in the domain of industrial robotics. Our current focus is on demonstrating PECT in incrementally more demanding and larger scale industry and DoD settings, and in documenting the methods we use to develop and validate PECTs. We are also broadening our repertoire of reasoning frameworks to include a variety of performance-analysis models, as well as automated verification (through model checking) of specific component and assembly-reliability claims.

Although we believe that we have demonstrated the potential of PECT, there are several challenges that must be met if the ideas are to find widespread use and acceptance:

Techniques for certifying, and labeling, component properties required by reasoning frameworks must be developed.

The business case for PECT must be established, since the development of PECTs requires substantial up-front investment.

The engineering methods and technology needed to build and use PECTS must be better understood, documented, and supported by commercial tools.

Although these are serious challenges, the needs addressed by PACC are real and immediate. Moreover, progress is being made, and not just at the SEI. Academic research and industrial practice are moving in the direction of predictable assembly, and the guaranteed component quality is demanded by the marketplace, by societal needs, and by our own quest to establish rigorous foundations for software engineering practice.

If PACC were to be summed up in a single sentence, it would be that it allows us to shift our focus from predicting the runtime properties of assemblies to building only assemblies whose properties we can predict.

For more information about PACC or to inquire about opportunities to collaborate with the SEI in this research, see the PACC Web site.

CMMI in Focus

Using CMMI® Appraisals in Acquisition—a Primer

Mike Phillips

When we began the CMMI project, we knew that we had to provide an appraisal method to accompany the models. Our initial choice was SCAMPI℠ V1.0 (Standard CMMI Appraisal Method for Process Improvement). But as we upgraded toward V1.1 models, our sponsor directed that we also specifically address the use of CMMI for source selection and contract monitoring, a function previously performed by the Software Capability Evaluation (SCE℠) or System Development Capability Evaluation (SDCE). SCAMPI V1.1 was significantly revised to enable that use. This column will address some of the ways that SCAMPI can be used with CMMI models for use in source selection and contract monitoring.

Establishing the Playing Field

Government contractors often receive a Request for Proposal (RFP) from a government program office that requires that candidate contractors achieve a particular CMM or CMMI maturity level to be considered a viable competitor for the contract. In some cases, proposing contractors are asked to provide evidence of process discipline. Frequently these are internal appraisals performed at the contractor's expense.

For the Capability Maturity Model for Software (SW-CMM), these appraisals are usually CMM-Based Appraisals for Internal Process Improvement (CBA IPIs). Some contractors have chosen to administer an internally (usually corporately) sponsored SCE, which provides similar information. For CMMI models, these appraisals use the Standard CMMI Appraisal Method for Process Improvement (SCAMPI). Some program offices prefer to send their own or other external teams to appraise the contractors' current development sites. As I mentioned above, these were called evaluations when the SW-CMM was the model, but SCAMPI has unified the appraisal methods into one that covers both internally sponsored assessments and externally sponsored evaluations.

An appraisal is used to confirm the progress a contractor has made in reaching higher levels of organizational maturity or process area capability. Several of a contractor's projects are chosen for appraisal. These projects are supposed to be representative of the contractor's full capabilities, not just the "pick of the litter." Typically, functional area experts from the contractor's other projects or non-project staff also are interviewed or provide data to ensure that the results gathered represent the entire contractor organization, not just the projects examined during the appraisal. The challenge for the appraisal team is to gain confidence that the goals of each applicable process area are being met across the organization—and that a deficiency in one project does not characterize the whole organization.

In many cases, the proposed development will require that the contractor create a new product team to work on the contract. Since a fully formed team may not have been created, the appraisal conducted to evaluate the contractor typically examines active project teams.

Because the appraisal precedes the creation of a new product team for the contract, the program office cannot have full confidence that the work to be performed under the proposal will be performed at the same maturity level as the appraisal suggests, because product teams are typically reformed and teamed differently with both internal and external partners. On the other hand, contractors that are continuing to improve their organizational maturity may well exceed the maturity levels documented in earlier appraisals.

Another way of stating this is that there is a risk that a new contract will stimulate both hiring and reorganization within the contractor organization. This may bring together organizational elements that weren't seen in the earlier appraisals. Further, many contracts for developing software-intensive systems demand teaming arrangements across multiple contractors. These requirements lessen the usefulness of predictive appraisals conducted on individual contractors because they are less directly applicable to the actual multi-contractor project team.

Thus, a preliminary appraisal may best be considered a way to establish the playing field among the contractors being considered—to set the minimum standards. If all proposing contractors are at maturity level 3, for example, the program office would have some degree of confidence that each of the contractors is able to predict cost and schedule realistically and that these contractors will be better able to work together than contractors that have not improved their development discipline to this level. Such a situation would not, however, give the program office confidence that development risks would be predictably addressed by the contractor team.

Reducing Development Risk

The original purpose of SCEs was to address risk to the program office. SCE teams were created by the program office to look at specific areas of concern. These areas of concern typically included a handful of process areas that were investigated at the contractor's site for a few days. A method similar to the SCE, the SDCE, involved asking the contractor questions that focused on specific areas of concern instead of maturity level ratings. In spite of the initial emphasis of these methods, maturity levels eventually became commonly used and familiar to the organizations using them. A maturity level focus displaced the risk focus of the two evaluation methods in many acquisition environments.

This emphasis may now be changing in some environments. For example, in the Department of Defense (DoD), the guidance from the Office of the Secretary of Defense/Acquisition, Technology, and Logistics (OSD/AT&L), which had required maturity level 3 for large DoD programs, no longer requires these programs to achieve these ratings. The new guidance still encourages these programs to pursue process improvement, but the maturity level expectation is no longer stated in policy guidance.

The program manager must still mitigate the risks of development of complex software-intensive systems. The CMMI Product Suite may be of assistance. Using a CMMI model and the SCAMPI method, the program office can conduct a baseline appraisal of the overall project and gain agreement with the contractors about the primary developmental risks to the success of the

contract. Risk mitigation then becomes a measure of program progress, and award fee mechanisms can be used to encourage and reward improvements that address the weaknesses identified in the baseline appraisal.

Contract monitoring appraisals, tailored to the areas of concern, can provide the confidence to the program office that needed improvements have been made by the contractors to ensure the success of the program. In one recent example, a baseline appraisal found 47 risk areas that could have potentially significant impact on program success. A year later, 41 of those risk areas had been mitigated by the contractors. The remaining six were still in progress, but the government and industry program managers agreed that this attention to process discipline was paying dividends on both sides.

Another approach briefed by an Army representative at the recent Software Technology Conference in Salt Lake City was similar. It allowed contractors to provide varied evidence of process improvement in their proposals. Maturity level documentation was not required. Instead, the contractors were encouraged to propose how the government-industry team would collaborate on continuing process improvement to both reduce risk and raise quality.

These real-world examples require the development of an acquisition strategy, RFPs, proposals, processes, and relationships that allow and encourage this sort of teamwork between the government and industry. Another factor in overall program success is the effectiveness of the teamwork across contractor teams from multiple companies.

Encouraging Multiple Contractor Teamwork

A frequent challenge these days occurs when multiple contractors, often with very different cultural roots, must work closely together. In these situations, separate appraisals of each contractor's capabilities may be misleading when applied to the overall program, as the teams have typically not been working together in the way they must after contract award. CMMI and SCAMPI offer some particular advantages in these situations. An appraisal of the entire team of contractors that focuses on a set of critical processes, or process areas, can be conducted. In some cases, there may be value in establishing a single process to be shared by multiple contractors; however, the primary value of such a focused appraisal is in knowing how the contractors must modify their processes to improve the effectiveness of integrating their work. Typically, the time spent in product integration and test is costly and time consuming. Early attention to process integration can pay big dividends by minimizing the time and effort required for product integration and test at the end of the product development life cycle.

There is a risk when contractors on a team vary in their level of process improvement achievement. A noteworthy example was a maturity level 5 contractor that needed subsystems provided by a maturity level 1 small company. The maturity level 5 contractor determined that the best way to ensure that these two contractors could work together effectively was to include the small company as a full team member for the contract. The small company's staff worked within the process architecture of the maturity level 5 contractor, and contributed its pieces of the system within a well-established framework. The small company gained exposure to the values of process discipline as a result of the mentoring teamwork. Consequently, all organizations involved benefited from this strategic approach. The program office was assured success, the

maturity level 5 contractor was able to meet its obligations with minimized risk, and the maturity level 1 contractor gained valuable experience.

Encouraging Government Contractor Teamwork

CMMI provides elements of the Software Acquisition CMM (SA-CMM) in process areas at maturity level 2 and maturity level 3. At maturity level 2, the Supplier Agreement Management (SAM) process area addresses acquisition issues. At maturity level 3, the Integrated Supplier Management (ISM) process area provides further guidance in acquisition activities. The integrated product and process development (IPPD) process areas, Integrated Project Management (IPM), Integrated Teaming (IT), and Organizational Environment for Integration (OEI), of course, provide a rich set of practices to encourage effective teamwork. There are great opportunities for shared value when implementing the Requirements Development (RD) and Requirements Management (REQM) process areas using a teaming approach.

Improving Internal Government Capabilities

The SEI has been asked by several program offices to help them improve the systems engineering provided within government acquisition. The SEI has supplemented the CMMI models with practices more directly applicable to an acquisition environment; these were extracted from the Software Acquisition CMM (SA-CMM) to provide the basis for investigation. Visits to program offices that lasted less than a week allowed team members to determine what capabilities were present in the workforce and where improved processes and/or training were required. While these visits share the basic data-gathering techniques of the more rigorous SCAMPI Class A, no attempt at determining a maturity level rating is seen as necessary or desirable.

Summary

As you may have gathered from the examples used in this article, much of the SEI's experience with contract monitoring has been with government organizations. However, these concepts are equally relevant in commercial industry. The goal is the same for government and industry—to ensure that the products they procure are developed and delivered by qualified contractors and result in quality integrated products.

CMMI models, coupled with a variety of appraisal approaches—from a full SCAMPI Class A through risk based appraisals to capability determinations—can be used in a variety of ways to meet the needs of the acquisition workforce, both in government and industry.

More information about this topic is available in the following documents, which are available on the SEI Web site:

CMMI models [http://www.sei.cmu.edu/cmmi/models/models.html]

SCAMPI V1.1 Method Definition Document
[http://www.sei.cmu.edu/publications/documents/01.reports/01hb001.html]

SCAMPI V1.1 Use in Supplier Selection and Contract Process Monitoring
[http://www.sei.cmu.edu/publications/documents/02.reports/02tn008.html]

SCAMPI V1.1 Method Implementation Guidance for Government Source Selection and Contract Process Monitoring [http://www.sei.cmu.edu/publications/documents/02.reports/02hb002.html]

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.

The COTS Spot

CMMI[SM] and BSCC

Robert C. Seacord

Introduction

Since the release of *Building Systems from Commercial Components* [Wallnau 01] in July of 2001, the design and engineering team from the SEI COTS-Based Systems Initiative has developed the collection of techniques described in that book into a more prescriptive COTS-based systems design process. This evolution has been partially described in this column and other forums and will soon be offered as part of an updated tutorial. However, we have not yet described the relationship between this work and another important SEI product, Capability Maturity Model® Integration (CMMI®) [CMMI 03]. In this column, I examine how this process, newly evolved from *Building Systems from Commercial Components* (BSCC) techniques, can be used to satisfy portions of the CMMI process model.

The CMMI model is illustrated in Figure 1. The process areas are clusters of related practices in an area that, when performed collectively, satisfy a set of goals in that area. CMMI Version 1.1 models identify 25 process areas, grouped into categories.

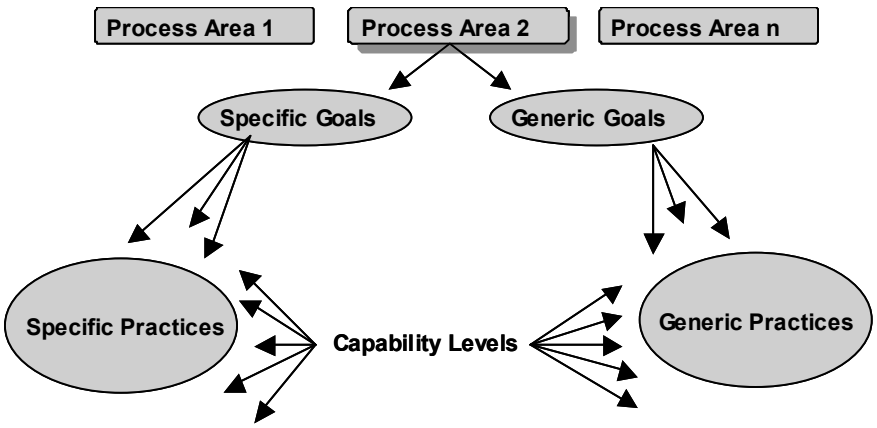

Figure 1: CMMI Model Components

Technical Solution

The Technical Solution practice area is defined in the Engineering process area. The purpose of Technical Solution is to design, develop, and implement solutions to requirements. Solutions, designs, and implementations encompass products, product components, and product-related life-cycle processes either singly or in combinations, as appropriate. The techniques and practices described in BSCC satisfy the following specific goals of the Technical Solution:

SG 1	Select Product-Component Solutions
	Product or product-component solutions are selected from alternative solutions.
SG 2	Develop the Design
	Product or product-component designs are developed.

The specific goal for selecting product-component solutions requires considering alternative solutions in advance of selecting a solution. This goal is supported in BSCC through the use of contingency planning, a technique for managing multiple design alternatives of unknown feasibility and determining how and when to apply resources to resolve critical unknowns. Risk/Misfit, another BSCC technique, is a decision aide used for determining the level of resources to be used in exploring the different design alternatives based on cost, schedule, performance, and risk. While Risk/Misfit itself may be better aligned with the Decision, Analysis, and Resolution (DAR) process area, its application here satisfies the specific goal of evaluating design alternatives against these criteria.

The specific goal of selecting product-component solutions in CMMI is supported by three specific practices:

SP 1.1 Develop detailed alternative solutions and selection criteria.
SP 1.2 Evolve operational concepts and scenarios.
SP 1.3 Select product-component solutions.

These specific practices are supported by BSCC processes and techniques as described in the following subsections.

Develop Detailed Alternative Solutions And Selection Criteria

Developing detailed alternative solutions is an essential concept of the Technical Solution process area. As already mentioned, BSCC encourages the development and management of multiple design alternatives through contingency management. The development of detailed design alternatives is supported in BSCC through the R^3 process and the use of model problems. R^3 stands for Risk analysis, Realize model problems, and Repair misfit. Risk analysis involves creating one or more model *blackboards* that illustrate a scenario through the design alternative containing a critical risk. If critical unknowns exist in the blackboard, model problems are realized to further assess the risk. Model problems are situated prototypes created to answer specific design questions. If specific risks still exist after creation of the model solution (to the model problem), it may still be possible to repair the misfit, either by adapting the design alternative or by changing the context (e.g., the requirements). Repair misfit is the final step in R^3.

Selection criteria are an inherent part of BSCC. The major selection criterion used in BSCC is design risk; for example, that the design does not satisfy stakeholder needs. Project-level design risk is assessed at the beginning of the process. Design alternatives are continually assessed against these risks throughout the R^3 process. The ability of each design alternative to address these risks is considered along with the costs and benefits of each approach in identifying a path for continued design-space exploration. Continued application of these criteria in the design process prevents unnecessary expenditure of resources on unfeasible solutions.

Evolve Operational Concepts And Scenarios

SP 1.2 evolves the operational concept, scenarios, and environments to describe the conditions, operating modes, and operating states specific to each product component. Operational concepts and scenarios are also a critical element of BSCC. Scenarios are developed and used in the creation of blackboards within R^3 to assess specific design risks.

Typical work products for this specific practice include product-component operational concepts, scenarios, and environments for all product-related life-cycle processes (e.g., operations, support, training, manufacturing, deployment, fielding, delivery, and disposal); timeline analyses of product-component interactions; and use cases. BSCC includes work products such as ensemble descriptions and blackboards that emphasize timeline analyses of product-component interactions. Use cases are used in BSCC to discover critical unknowns in a design alternative and to explore design risk in the R^3 process.

Select Product-Component Solutions

The final specific practice for SG 1 selects the product-component solutions that best satisfy the criteria established. Since integration problems are a major source of risk in COTS-based systems development, COTS components must be selected in the context of a feasible design solution. Consequently, evaluation in BSCC is an inherent part of the design process. Successful execution of BSCC results in the selection of a feasible design alternative that consists of a collection of components and design decisions.

Summary

The application of BSCC satisfies the specific goal of developing detailed alternative solutions and selection criteria in the Technical Solution process area for COTS-based system development. Both CMMI models and BSCC emphasize the need to evaluate multiple design alternatives relative to cost, schedule, and risk, as well as achieving the required functionality and design qualities.

I will explore, in a future column, where and to what degree BSCC satisfies the specific practices of SG 2 (developing the design).

References

[CMMI 03] Capability Maturity Model® Integration (CMMI[SM]), Version 1.1 CMMI[SM] for Systems Engineering, Software Engineering, Integrated Product and Process Development, and Supplier Sourcing (CMMI-SE/SW/IPPD/SS, V1.1) Staged Representation CMU/SEI-2002-TR-012/ESC-TR-2002-012, March 2002.

[Wallnau 01] Wallnau, Kurt; Hissam, Scott; and Seacord, Robert. *Building Systems from Commercial Components*. New York, NY: Addison-Wesley, 2001.

About the Author

Robert C. Seacord is a senior member of the technical staff at the SEI and currently leads a team researching software sustainment. He is coauthor of 2 SEI Series books, *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices* and *Building Systems from Commercial Components* as well as more than 40 papers on component-based software engineering, Web-based system design, legacy system modernization, component repositories and search engines, security, and user interface design and development. He has over 20 years of development experience and was previously a technical staff member at the X Consortium and IBM.

Security Matters

Use Care When Downloading and Installing Programs

Larry Rogers

When you buy an appliance, you give little thought to it doing you or your house any harm. Why? Because there are organizations like Underwriters Laboratories that set standards and certify products. When you see a certifier's label, you have more confidence that a product will be safer than a competing product that does not carry the same label. You're willing to accept the risk because you believe the product has met some standards and has been certified by a respected authority.

Unfortunately, the Internet is not the same. There are neither standards nor many certification organizations. Anyone who writes a program can distribute it through any means available, such as through the Web or by sending you a copy. Speaking of that, have you ever received a CD-ROM in the mail? How do you know that it contains what the label says? The answer is: you don't know. More importantly, it's difficult to know.

No matter how you acquire a program, it runs on your computer at the mercy of the program's author. Anything, any operation, any task that you can do, this program can also do. If you're allowed to remove any file, the program can too. If you can send email, the program can too. If you can install or remove a program, the program can too. Anything you can do, an intruder can do also, through the program you've just installed and run.

Sometimes there's no explanation of what a program is supposed to do or what it actually does. There may be no user's guide. There may be no way to contact the author. You're on your own, trying to weigh a program's benefits against the risk of the harm that it might cause.

What's the problem you're trying to solve here? You are trying to determine if the program you've just found satisfies your needs without causing harm to your computer and ultimately the information you have on the computer. How do you decide if a program is what it says it is? How do you gauge the risk to you and your computer by running this program?

You address these same risk issues when you purchase an appliance; you may just not have realized that's what you were doing. When you make that purchase, you buy from either a local store you know or a national chain with an established reputation. If there's a problem with your purchase, you can take it back to the store and exchange it or get your money back. If it causes you harm, you can seek relief through the legal system. The

reputation of the merchant, the refund/return policy, and the availability of the legal system reduce your risk to a point where you make the purchase.

Apply these same practices when you buy a program. You should

- Learn as much as you can about the product and what it does before you purchase it.

- Understand the refund/return policy before you make your purchase.

- Buy from a local store that you already know or a national chain with an established reputation.

Presently, it is not as clear what the legal system's role is for a program that causes harm or does not work as advertised. In the meantime, the LUB practices are a good first step.

But what about all those free programs available on the Internet? There is a multitude of free programs available for all types of systems, with more available each day. The challenge is to decide which programs deserve your confidence and are, therefore, worth the risk of installing and running on your home computer.

So how do you decide if a program is worth it? To decide if you should install and run a program on your home computer, follow these steps:

1. The Do test: What does the program do? You should be able to read a clear description of what the program does. This description could be on the Web site where you can download it or on the CD-ROM you use to install it. You need to realize that that if the program was written with malicious intent, the author/intruder isn't going to tell you that the program will harm your system. He or she will probably try to mislead you. So, learn what you can, but consider the source and consider whether you can trust that information.

2. The Changes test: What files are installed and what other changes are made on your system when you install and run the program? Again, to do this test, you may have to ask the author how the program changes your system. Consider the source.

3. The Author test: Who is the author? (Can you use email, telephone, letter, or some other means to contact him or her?) Once you get this information, use it to try to contact the author to verify that the contact information works. Your

interactions with the author may give you more clues about the program and its potential effects on your computer and you.

4. The Learn test: Has anybody else used this program, and what can you learn from him or her? Try some Internet searches using your Web browser. Somebody has probably used this program before you, so learn what you can before you install it.

If you can't determine these things – the DCAL tests for short – about the program you'd like to install, then strongly consider whether it's worth the risk. Only you can decide what's best. Whatever you do, be prepared to rebuild your computer from scratch in case the program goes awry and destroys it. "**Make Backups of Important Files and Folders**" in ***Home Computer Security*** tells you how to make a copy of your important information so you'll have it if you need it.

Your anti-virus program prevents some of the problems caused by downloading and installing programs. However, you need to remember that there's a lag between recognizing a virus and when your computer also knows about it. Even if that nifty program you've just downloaded doesn't contain a virus, it may behave in an unexpected way. You should continue to exercise care and do your homework when downloading, installing, and running new programs.

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

What's the Difference Between Product Line Scope and Product Line Requirements?

Paul Clements

Setting the proper scope of the product line is a step of fundamental importance. Make it too large and the core assets will have to be too hopelessly generic to ever work; make it too small, and the market demand for your product suite will be too small to recover the up-front investment. Make it the right size but encompassing the wrong systems for your market and you'll have no customers.

What does the representation of a product line scope look like, exactly? That is, how do you write one down? It can be very vague, or very precise—any statement that will let a decision-maker decide whether a proposed product is in or out will do. In practice, to make such a decision requires a fairly precise statement of what the "in" systems will all have in common. One product line scope definition that we once saw talked about systems for special-operations helicopters. These systems were supposed to do three things: "aviate" (that is, fly), "navigate" (move from point A to point B), and "communicate" (talk to other systems). While this description ruled out, say, software for toasters, it is clearly insufficient to tell whether the software for another kind of aircraft would be in or out of the product line. Something more detailed was needed. During the next round, some specific behaviors and features were articulated, and this began to position the product line squarely in the realm of special-operations helicopters.

This is usually how it goes. Scope starts out as a vague description of a set of systems by naming some functions they provide. After that, then it's refined to be written in terms of the products' observable behaviors and their exhibited quality attributes such as performance or security.

But that is also how writing down requirements for a system goes. So why are the scope and the requirements covered in two separate practice areas? Aren't they in fact the same activity? The answer is "Theoretically, yes." But "theoretically" is often a euphemism for "not really."

In part a of Figure 1, the rectangle represents every possible software system that ever has been, ever will be, or ever could be built. This is the starting point, albeit a not very useful one, for determining the scope of a product line. Typically, a product line manager is able to start describing systems that are definitely outside the product line (toasters) and a few products that are definitely in (a small list of specific special-operations helicopters). Part b of Figure 1 shows the system space divided into three parts: systems that are out (mottled), systems that are in (white), and systems we aren't sure about yet (black). The process of defining the product line scope is the process of narrowing down the "not sure about" space by carefully defining more of the "out" and "in" spaces, until conceptually it resembles part c.

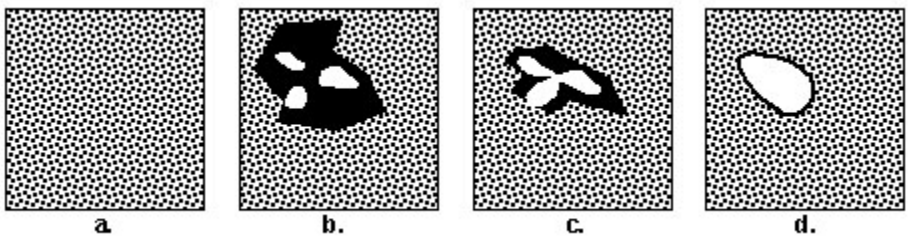

Figure 1: The Evolution of a Product Line Scope

Now think about requirements. A good requirements specification should tell us everything that must be true about a system for it to be acceptable, and no more. The "no more" part means that many systems could be built to satisfy a given requirements specification, since a statement of requirements does not and should not mandate a particular implementation. Requirements for a product line specify things that are true about every system in the family, and specify a set of allowable variations exhibited by individual systems. Given any system, we should be able to compare it to the requirements specification and see whether or not it conforms—that is, whether it satisfies all of the common requirements and an allowable combination of the variable requirements. It either does or it doesn't: There is no in between. Part d of Figure 1 is the visual rendition of this situation, which is just part c pursued to the point where the "not sure about" space has been squeezed to nothingness. This is why we said that requirements engineering was theoretically akin to product line scoping—a completely precise scope is, in fact, a requirements specification for the product line.

At least theoretically. Often, the scope includes aspects of the system that would not appear in even the most complete requirements specification, aspects related to business goals or construction constraints. For example, "any reasonable system commissioned by our most important customer" might appear in a scope definition as something you're willing to build, but not in any requirements spec because it's not a testable condition of the software.

In addition, the scope and the requirements are defined at different times. The scope is defined early enough so that a business case can be built and used to see if the product line is economically viable. The requirements are specified as a prelude to actual development. The two products have different consumers. The scope is written for people like marketers, who need to see what they will be asked to sell but do not require full statements of product behavior. The requirements spec is used by the architect and the developers of core assets and products who *do* need to know exact behavior. (The scope definition is also used by the architect to begin planning architectural means to provide the commonality and variability defined there.) Finally, there is no mandate for the scope to be completely precise—the situation in part c of Figure 1 is just fine. The vast majority of systems are ruled out, a useful number of systems are ruled in, and a class of systems remains on the cusp, meaning "If asked to build one of those, we'll think

about it." If, over time, you're asked to build a *lot* of those, it may be a sign that your scope missed the target a bit, and needs to be adjusted.

In fact, the situation in part c is *more* desirable for a scope than the one in part d. If you're asked to build a system that lies oh-so-close but just outside the anointed set of "in" systems, then "I'll think about it" is probably a better response than a flat "No." Rejecting it out of hand might cause you to miss a good business opportunity that you had not previously considered.

So if you thought that setting the scope and setting the product line requirements sounded like similar activities, you're right. They are. In practice, however, scoping and requirements engineering are done by different people, stop at different points, and are used for different purposes by different stakeholders.

Some Programming Principles: Products

Watts S. Humphrey

The Nature of Computer Programs

Computer programs are fundamentally different from other types of products. Software doesn't wear out, rot, or deteriorate. Once a program works and as long as its operating conditions remain the same, it will continue to work indefinitely. Software products can be reproduced for essentially nothing and distributed worldwide in seconds. From a business perspective, software is almost an ideal product. It is the most economical way to implement most logic and it is the only way to implement complex logic. As an intellectual product, software can be legally protected almost indefinitely and this makes software products potentially very profitable. These basic software characteristics provide great opportunities. However, they also present us with six major challenges. To properly address these challenges, we must substantially change the way we do our work and the methods we use to run our projects.

The First Challenge: Software Work Is Intellectual

Because software work is intellectual and our products are intangible, we cannot easily describe what we do or demonstrate our results. This makes it extremely difficult for non-software people to manage software groups or even to understand what we do. As more people get involved with and become knowledgeable about software, this will be less of a problem. However, today, few customers, managers, or executives have an intuitive feeling for the software business. The problem is that you can't touch, see, or feel software. When we tell them that the product is nearly ready to ship, they have no real appreciation for what we are saying. We can't take them out into the laboratory and show them the first software model being assembled, describe which parts must be added before final test, or show them parts being welded, machined, or painted.

The problem here is trust. Businesses run on trust, but many managers follow the maxim: "Trust but verify." Unfortunately, with software, management can only verify what we say by asking another software person who they hope is more trustworthy. Since we typically don't have any way to prove that what we say is true, management can only tell if we are telling an accurate story by relating what we currently tell them with what we have said before and how that turned out. Unfortunately, the abysmal history of most software operations is such that very few if any software developers or managers have any credibility with their more senior management.

The result is a critical challenge for the software community: we work in an environment where senior management doesn't really believe what we tell them. This means that, almost invariably, management will push for very aggressive development schedules in the hopes that we will deliver sooner than we otherwise would. To maintain this schedule pressure, management is not very receptive to pleas for more time. Furthermore, even when we break our humps and actually deliver a product on the requested schedule, management isn't very impressed. After all, that is only what we said we would do. Other groups do that all the time.

The Second Challenge: Software Is Not Manufactured

Because software can be reproduced automatically, no manufacturing process is required. This means that there is no need for a manufacturing release process and therefore that there is no external process to discipline our design work. When I used to run systems development projects, the hardware had to be manufactured in IBM's plants. Before I could get a cost estimate signed off or get a price and product forecast, I needed agreement from the manufacturing and service groups. This required that the product's design be released to manufacturing. In this release process, the manufacturing engineers reviewed the design in great detail and decided whether or not they could manufacture the product for the planned costs, in the volumes required, and on the agreed schedules. We also had to get service to agree that the spare parts plan was adequate, that the replacement rate was realistic, and that the service labor costs were appropriate.

As you might imagine, release-to-manufacturing meetings were extremely detailed and often took several days. The development engineers had to prove that their designs were complete enough to be manufactured and that the cost estimates were realistic and accurate. While these release meetings were grueling and not something that the development engineers enjoyed, once you passed one, you had a design that the manufacturing people could build and everybody knew precisely where your program stood.

Unfortunately, software development does not require such a release process. The consequence is that design completion is essentially arbitrary and we have no consistent or generally-accepted criteria that defines what a complete design must contain. The general result is poor software-design practices, incomplete designs, and poor-quality products. The challenge here is that, without any external forces that require us to use disciplined design practices, we must discipline ourselves. This is difficult in *any* field, but especially for software, since we have not yet learned how to reliably and consistently do disciplined software work.

The Third Challenge: The Major Software Activities Are Creative

Because software can be reproduced automatically, a traditional manufacturing process is not required and no manufacturing resources are needed. This is a major change from more traditional products where the manufacturing costs are often more than ten times the development expenses. In software, the principal resources are development and test. This mix imposes a new set of demands on management: they must now learn to manage large-scale intellectual work.

In the past, large-scale activities have generally concerned military operations or manufacturing processes. Typically, large numbers of people have been needed only for repetitive or routine activities like reproducing already-designed products. In software, large-scale efforts are often required to develop many of the products. In directing large numbers of people, management has typically resorted to autocratic methods like unilaterally establishing goals, setting and controlling the work processes, and managing with simplistic measures. The problem here is that large-scale intellectual work is quite different from any other kind of large-scale activity. Autocratic practices do not produce quality intellectual work and they are counterproductive for software. In fact, such practices often antagonize the very people whose creative energies are most needed.

To address this challenge, management must understand the problem and they must also get guidance on what to do and how to do it. While the proper management techniques are not obvious, they are not very complex or difficult. And once they are mastered, these management techniques can be enormously effective.[1]

The Fourth Challenge: Software Lives Forever

Because software essentially lives forever, product managers face an entirely new and unique set of strategic issues. For example, how can they continue to make money from essentially stable products, and what can they do to sustain a business in the face of rampant piracy? The problem is that immortal products that can be reproduced for essentially nothing will soon lose their unique nature and cease to be protectable assets. While this is not a severe problem when the software is frequently enhanced, once products stabilize, they will be exceedingly hard and often impossible to protect, at least for anything but very short periods.

This may not seem like a serious problem today, but it soon will be. A large but ultimately limited number of basic functions will be required to provide future users with

---

[1] See my book, *Winning with Software: an Executive Strategy*. Reading, Mass., Addison Wesley, 2002.

a stable, convenient, accessible, reliable, and secure computing capability. While such functional needs have evolved rapidly over the last 50 years, the rate of change will inevitably slow. This fact, coupled with the users' growing needs for safety, security, reliability, and stability, will require that the rate of change for many of our products be sharply reduced. This in turn will make it vastly more difficult to protect these products.

The reason that this is important to programmers is that if the uniqueness of our products cannot be protected, our organizations will be unable to make money from the products we produce. They will then no longer be able to pay us to develop these products.

Lest this prediction sound too dire, the programming business will not wither away. I am only suggesting that the part of the programming business that provides basic system facilities will almost certainly have to change. On the other hand, I cannot visualize a time when application programming will not be a critical and valuable part of the world economy. In fact, I believe that skilled application programming will become so important that the programming profession as we now know it will no longer exist: every professional will have to be a skilled application programmer.

The Fifth Challenge: Software Provides Product Uniqueness

Because software contains the principal logic for most products, it provides the essential uniqueness for those products. This means that the software's design is an essential product asset and that the key to maintaining a competitive product line is maintaining a skilled and capable software staff. This is an entirely new consideration for a management group that has viewed software as an expense to be limited, controlled, and even outsourced, rather than as an asset to be nurtured, protected, and grown. The pressure to limit software expenses is what caused IBM to lose control of the PC business. Management was unwilling to devote the modest resources needed to develop the initial PC software systems. This gave Bill Gates and Microsoft the opportunity to replace IBM as the leader of the software industry.

As software becomes a more important part of many products, whole industries are likely to lose control of their products' uniqueness. This control will be in the hands of the programmers in India or China or whoever else offered the lowest-cost bids for outsourcing the needed software work. In effect, these industries are paying their contractors to become experts on their products' most unique features. Ultimately, these contractors will very likely become their most dangerous competitors. Over time, these industries may well find themselves in a position much like IBM's in the PC business: manufacturing low-profit commodity-like hardware to run somebody else's high-margin software.

The Sixth Challenge: Software Quality is Critical

Because software controls an increasing number of the products we use in our daily lives, quality, safety, security, and privacy are becoming largely software issues. These increasing quality needs will put enormous pressure on software businesses and on software professionals. The reason is that software safety, security, and privacy are principally software design issues. Without a complete, fully-documented, and competently reviewed design, there is practically no way to ensure that software is safe, secure, or private. This is a problem of development discipline: since we don't have to release our products to a manufacturing or implementation group, there is no objective way to tell whether or not we have produced a complete and high-quality design.

This development discipline problem has several severe consequences. First, it has never been necessary for software people to define what a complete software design must contain. This means that most software engineers stop doing design work when they believe that they know enough to start writing code. However, unless they have learned how to produce complete and precise designs, most software engineers have only a vague idea of what a design should contain.

With the poor state of software design and the growing likelihood of serious incidents that are caused by poor-quality, insecure, or unsafe software, we can expect increased numbers of life-critical or business-critical catastrophes. It won't take many of these catastrophes to cause a public outcry and a political demand for professional software engineering standards. This will almost certainly lead to the mandatory certification of qualified software engineers. Then, the challenge for us will be to determine what is required to be a qualified software engineer and how such qualification can be measured.

Conclusions

Since we have been living with all of these problems for many years, you might ask why we should worry about them now. The reason is that the size and scope of the software business has been growing while software engineering practices have not kept pace. As the scale and criticality of software work expands, the pressures on all of us will increase. Until we learn to consistently produce safe, secure, and high-quality software on predictable schedules, we will not be viewed as responsible professionals. As the world increasingly depends on our work, we must either learn how to discipline our own practices or expect others to impose that discipline on us. Unfortunately, in the absence of agreed and demonstrably effective standards for sound software engineering practices, government-imposed disciplines will not likely be very helpful and they could even make our job much more difficult.

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, Bill Peterson, and Alan Willett.

In closing, an invitation to readers

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

Thanks for your attention and please stay tuned in.

Watts S. Humphrey

watts@sei.cmu.edu

About the Author

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

The views expressed in these articles are the author's only and do not represent directly or imply any official position or view of the Software Engineering Institute or Carnegie Mellon University. This article is intended to stimulate further discussion about this topic.

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

Documenting Software Architectures

Pennie Walters

Every day, in organizations around the world, thousands of dollars are invested to design and build software architectures, only to result later in architectures that can't be reproduced, repaired, analyzed, or even implemented due to poor documentation or a complete lack of it. In the past, no one seemed to know what makes good software architecture documentation. Though many books described the languages used to create it, none addressed its content or purpose.

Typically the question of how to document a software architecture was answered by identifying how *not* to do it, leaving documenting as a trial-and-error process—sometimes with costly consequences.

Researchers at the Software Engineering Institute tackle the issue of software architecture documentation in a new book titled *Documenting Software Architectures: Views and Beyond*.

This book, which was just awarded a Jolt Productivity Award from *Software Development* magazine, strives to answer the critical question, How do you document an architecture so that others can successfully use it, maintain it, and build a system from it?

What Is a Software Architecture?

A software architecture is the structure or structures of a system, comprising elements, their externally visible properties, and the relationships among them all. Documenting software architecture is important because it serves as the blueprint for a system and the project that develops that system. It defines work assignments and is the primary carrier of quality attributes (e.g., performance, reliability, security, and modifiability), the best artifact for early analysis, and the key to post-deployment maintenance and mining. So it makes sense that documenting an architecture is the crowning step to creating it.

Because software architectures are too complicated to be seen all at once, documenting them using views is a helpful technique. A view is a representation of some of the system's elements and the relationships associated with them. The most fundamental principle of architecture documentation comes from the concept of views: Documenting an architecture is a matter of documenting the relevant views and then adding documentation that applies to more than one view.

While some software architects prescribe using standard views for a particular type of system, *Documenting Software Architectures* recommends using an approach that incorporates input from the stakeholders (those who have a vested interest in the system, such as developers or subcontractors). Such input tells architects how the documentation will be used and what stakeholders expect from it. Ultimately, this input helps the architects decide which information should be part of a software architecture's documentation.

Knowing Which View to Use

Which views are relevant for a particular software architecture depends on who the stakeholders are and how they plan to use the documentation. Software architecture documentation has three primary uses: as a resource for educating infrastructure support personnel, new stakeholders, future architects, and others about the system; as a vehicle of communication among stakeholders; and as the basis for system analysis. For example, if future architects will use the documentation to gauge the impact of an expected change, a uses view (showing which modules are required for the proper operation of certain parts of a system) and a decomposition view (showing how a system's responsibilities are partitioned across modules) would be relevant. For nontrivial systems, one module view (showing elements that are units of implementation) is typically used, as well as one component-and-connector view (showing elements with runtime behavior and interaction), and one allocation view (showing how software structures are allocated to non-software structures).

Views of large software systems might contain hundreds or thousands of elements, so architects need to break the information presented into manageable chunks called view packets. A view packet is the smallest cohesive bundle of documentation that would be given to a stakeholder. It can show broad areas of the system at slight depth or small areas of the system at great depth. To orient the reader looking at a view packet, context diagrams are also included in the software architecture documentation. These diagrams depict the system or part of the system explained by each view packet.

Sound software architectural documentation should also show how views are related to each other through a combined view or a bridging document that relates elements and relationships.

Additional Documentation Components

Once software architecture is documented through views, it is necessary to add seven additional components to the documentation:

1. a documentation roadmap that tells how the documentation is organized, lists the views used and their elements, and provides scenarios for determining which parts of the documentation to consult
2. a view template that explains how each view is documented and organized
3. a system overview that provides context for new architects through an informal description of the system, including its purpose and functionality
4. mapping between views, which establishes useful correspondence between them
5. a directory, which shows where each element, relationship, and property is defined and used
6. an architecture glossary and acronym list
7. background information, design constraints, and rationale

These components are described in detail in *Documenting Software Architectures*.

Architecture Documentation Protects Your Investment

For nearly all systems, quality attributes are every bit as important as making sure that the software performs its expected functions. Architecture is the means by which these quality attributes are built into a system; in documentation, the formula for achieving them is put down

on paper. Using *Documenting Software Architectures* as your guide, you can create sound software architecture documentation that will protect the investment made during hours of design and decision making, and ultimately protect the intellectual property of your organization.

For more information, contact—

Paul Clements
Phone
512-453-1471
Email
clements@sei.cmu.edu
World Wide Web
http://www.sei.cmu.edu/ata/products_services/dsa_book.html?si

Second International Conference on COTS-Based Software Systems

Bill Anderson and Anatol Kark

Held in Ottawa, Canada's beautiful capital, the International Conference on Commercial Off-the-Shelf-Based Software Systems (ICCBSS) provided a forum for attendees to exchange ideas about current best practices and promising research directions for creating and maintaining systems that incorporate commercial off-the-shelf (COTS) software products.

Brian Lillico, a director for the Canadian Public Works and Government Services, summarized his team's impressions. "[Our] team especially enjoyed seminars where project leaders spoke about their project experiences and some of their lessons learned. Overall, we think [ICCBSS is] on the right track and does address the growing need for this type of information. I look forward to attending the 2004 session."

ICCBSS, pronounced "ice cubes" even before this year's subzero temperatures in Canada's winter wonderland, is truly international in scope:

- Presentations from Europe, Australia, and the Americas highlighted the state of the practice and introduced research in techniques for managing and engineering COTS-based systems.
- More than 100 practitioners from all over the world attended, representing government, military, commercial, and academic interests.
- ICCBSS is jointly sponsored by the National Research Council Canada, the Software Engineering Institute, the USC Center for Software Engineering, and the European Software Institute.

The conference was opened by Dr. Victor Basili, professor of computer science at the University of Maryland and executive director of the Fraunhofer Center. He provided a proposal for formalizing knowledge about COTS-based system development. Dr. Basili described a refinement of the definition of COTS to distinguish "easy" from "hard" COTS products. He introduced the concept of patterns of COTS-based systems to support more rigorous analysis of the impact of COTS products upon system development.

Presenters were selected on the basis of refereed papers published by Springer Verlag as Volume 2580 of *Lecture Notes in Computer Science*. These papers addressed COTS quality, architecture, and project-management topics including:

- ISO/IEC-based quality models and classification techniques, protective wrappers (software that integrates a COTS product into a system to provide some quality protection), and an evaluation framework that allows detailed software evaluation while protecting the vendor's intellectual property rights
- definition and substantiation of architecture conflicts that produce interoperability problems, a decision model to facilitate communication, recommendations for designing secure systems, practical experience in integrating COTS in safety-critical systems, and techniques for embedding executable specifications in software-component interfaces
- mechanisms to help organizations avoid inadequate practices and monitor project performance, modifications to COTS-based-systems cost models to address security concerns, and techniques used to help small manufacturing enterprises apply COTS solutions

One excellent paper[1] reported on an experimental application of i*, an agent-based approach for selecting multiple, interdependent software components to meet complex system requirements. Of particular interest to participants was a panel discussion in which representatives from Oracle, Peoplesoft, and Opentext challenged the community to become better customers, realize that COTS vendors are not custom-system developers, and leverage COTS software by being open to process modification.

Half-day tutorials enabled participants to examine topics in greater detail:

- functional fit analysis techniques to select the best solution and determine the degree of enhancement work required to meet customer requirements
- U.S. Air Force use of an engineering and management process to select, field, and support COTS products, other existing components, or custom components in complex environments
- matching component capability to system need through the application of marketing principles and practices

In the closing keynote, Dr. Robert Balzer, chief technical officer at Teknowledge Corporation, challenged software developers to be more creative in leveraging commercial offerings to their advantage. He demonstrated an integration architecture that used commonly available COTS products to meet specific end-user needs.

The conference provided a lively forum for exchanging experiences, ideas, and formal research. COTS practitioners put new challenges in front of the research community, from acquisition and economic models to development, integration and testing techniques. COTS researchers proposed new techniques and tools that are ready for practical application. These challenges and emerging pragmatic solutions will be explored at the Third ICCBSS, "Matching Solutions to Problems," which will be held in Redondo Beach, CA, Feb. 1-4, 2004.

The conference encourages your participation through conference attendance and submission of papers, presentations, tutorials, panels, or posters. Further information, including information about how to order the proceedings of past and upcoming conferences, is available at the conference Web site:
http://www.iccbss.org.

For more information, contact—

Customer Relations
Phone
412-268-5800
Email
customer-relations@sei.cmu.edu
World Wide Web
http://www.iccbss.org

---

[1] Franch, X. and Maiden, N.A.M. "Modeling Component Dependencies to Inform Their Selection," 81-91. *Proceedings of the COTS-Based Software Systems Second International Conference (ICCBSS 2003)* in *Lecture Notes in Computer Science 2580*. Ottawa, Canada, February 2003. Heidelberg, Germany: Springer-Verlag, 2003.

SEPG℠ 2003 Attendance Tops 1,500

Pamela Curtis

The 15th Software Engineering Process Group (SEPG) Conference (SEPG 2003) was held in Boston on February 24-27, 2003. More than 1,550 people attended, from defense and civil agencies, defense and commercial industry, and academic institutions.

Nearly three-fourths of attendees rated the SEPG keynote presentations as good to excellent compared to keynotes they'd heard at similar conferences. In the keynotes:

- Tom Davenport, director of the Accenture Institute for Strategic Change and a well-known author on business process reengineering, knowledge management, and enterprise systems, presented "Six Ways to Make Knowledge Work Better."

  "Process shouldn't cover everything," Davenport said. "Leave less routine tasks unstructured to allow for creativity. Avoid over-engineering."
- Allan Woods, vice chairman and chief information officer of Mellon Financial Corporation, spoke on "Execution."

  "Inertia is part of the natural law," Woods said. "Execution challenges us to get out of the ruts, to challenge conventional wisdom."
- Bill Hancock, vice president and chief security officer of Exodus, a cable and wireless service where he is responsible for global security for one of the world's largest hosting companies and IP networks, presented "Security Issues and Programming Fears."

  "A lot of code out there—including nuclear reactor subsystems—hasn't got a shred of security in it, because of limited resources," Hancock said.

The keynote presentations and speaker biographies are available on the SEI Web site.[1]

SEPG 2003 included increased numbers of presentations by users and adopters of Capability Maturity Model® Integration (CMMI®) models and Team Software Process℠ (TSP℠) from the United States, Europe, and Asia, evidence of the growing impact of these SEI technologies in the global community of software engineers. On display in the exhibit hall was a new book in the SEI Series in Software Engineering, *CMMI: Guidelines for Process Integration and Product Improvement*, written by SEI staff members Mary Beth Chrissis, Mike Konrad, and Sandy Shrum. The book, which generated much interest at the conference, is the definitive source for CMMI model information.

First SEPG News Conference

SEPG 2003 also featured a first-ever news conference, which focused on the relationship

---

[1] 1 *http://www.sei.cmu.edu/sepg/keynotes.htm?ns*

between security and software quality. The news conference began with a panel discussion led by SEI Director Steve Cross; Rich Pethia, director of the SEI's Networked Systems Survivability Program; Watts Humphrey, SEI fellow and creator of TSP; and Carol Grojean, a senior program manager with Microsoft. Articles resulting from the news conference have appeared in *eWeek*,[1] *Application Development Trends*,[2] and *CIO Magazine*.[3]

Pethia described the seriousness of the software quality problem: there were 40,000 new reports of software vulnerabilities in 2002, and the same types of vulnerabilities are reported year after year. "It takes a half a man year for a system administrator just to read all the new vulnerability reports," Pethia stated. "Applying patches for all of them is impossible. We need an order-of-magnitude decrease in security flaws in released software." Achieving this, Humphrey explained, will require changing the practices of software engineers, a primary goal of TSP. He described how the TSP team is working with staff from the SEI's Survivable Systems initiative to identify common security problems in software and then add practices to TSP that will help engineers avoid injecting such defects into the software they develop.

Grojean presented compelling data about the positive impact of TSP on cost, schedule, and quality for the project that she leads. For example, based on the projected reduction in the number of defects in the code, Grojean's team expects a 94% reduction in the cost of post-production fixes. Grojean said her team's use of TSP "gives management increased confidence in what we deliver, since our emphasis has been on quality from the beginning."

SEPG 2004

The 16th SEPG Conference will be held on March 8-11, 2004, at the Marriott World Center in Orlando, Florida.

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?ns

---

[1] http://www.eweek.com/article2/0,3959,922974,00.asp

[2] http://www.adtmag.com/article.asp?id=7358, http://www.adtmag.com/article.asp?id=7378

[3] http://www2.cio.com/research/security/edit/a02272003.html

New CERT® Certification to Train Computer Security Incident Handlers

Eric Hayes

Most organizations today depend on networked computer systems as an integral part of their businesses. As applications become more complex and services become increasingly integrated, protecting network security and recovering from computer security incidents has become a business-critical component of an organization's IT security plan. Creation of computer security incident response teams (CSIRTs) is an organizational best practice for protecting information assets and ensuring that an organization's mission survives. New laws and regulations also require organizations to identify and implement response capabilities—in some cases mandating that such incident response teams be formalized.

The CERT Coordination Center (CERT/CC) has been in the business of Internet security since 1988, handling incidents and helping other organizations create incident response teams. To help ensure a supply of qualified personnel to meet the growing demand for organizational incident response teams, the CERT/CC has created a program to train and certify individuals as CERT-Certified Computer Security Incident Handlers. CERT certification provides a tangible recognition of skills from the Internet's first and best-known computer security incident response team.

Certified incident handlers will be trained to handle diverse aspects of incident response and team leadership, ranging from applying operational concepts to managing a team to using technical expertise to prepare for, detect, analyze, and respond to security events. This range of knowledge ensures that CERT-certified incident handlers are well prepared to recognize and respond to security risks and threats.

The Curriculum

The certification requires individuals to take four core courses from the SEI or from an SEI transition partner (an organization licensed to provide SEI courses):

1. Creating a Computer Security Incident Response Team (CSIRT) (one day)
2. Information Security for Technical Staff (five days)
3. Managing CSIRTs (three days) or Fundamentals of Incident Handling (five days)
4. Advanced Incident Handling (five days)

One elective course in computer forensics, intrusion detection and analysis, or security audits and assessments is also required. This requirement is met by completing a course at a university or college accredited by the Accreditation Board for Engineering and Technology (ABET) or by completing five continuing education units from a recognized security training organization.

Once the prerequisites and course curricula have been completed, a final requirement for certification is to pass an SEI-administered written exam. The certificate is valid for three years; renewals require additional continuing education units and experience in the field.

Qualifications for Applicants

Applicants need to have at least three years of technical or managerial experience in incident handling (IH). After applicants meet this basic prerequisite, they must submit a letter of recommendation from their current or previous manager in support of their application.

The program welcomes incident handlers, CSIRT managers, system and network administrators with IH experience, and IH trainers and educators, as well as those with some technical training who want to enter the IH field.

Benefits of Certification

The certification should help computer-security professionals in their careers by demonstrating that they have achieved a high level of expertise. Organizations that hire CERT-Certified Computer Security Incident Handlers will benefit by having employees who are able to

- identify the benefits, challenges, and operational requirements needed to successfully create a structured incident handling or management team
- successfully participate as leaders or members of CSIRTs
- describe the information security tenets of confidentiality, availability, and integrity, and apply these concepts to the protection of information and information assets in an enterprise using a variety of technical and procedural solutions
- recognize and identify organizational risks and threats
- recommend and implement best practices for incident handling functions, computer security solutions, and mitigation strategies to reduce risks and counter threats across the enterprise
- demonstrate technical expertise in analyzing incident data and identifying response strategies

The SEI invites all qualified applicants to consider the CERT-Certified Computer Security Incident Handler certification program to enhance their computer security careers.

For a directory of organizations that teach SEI courses, see the SEI transition partner Web site:
http://www.sei.cmu.edu
/collaborating/partners/?ns

For more information, contact—

Kimberly Lang
Phone
412-268-9564
Email
training-info@cert.org
World Wide Web
http://www.cert.org/csirts

Improving Workforce Capabilities with the People Capability Maturity Model®

Sally Miller, Bill Curtis, and William Hefley

Authors' note:
A more in-depth version of this article can be found in the April 2003 issue of CrossTalk *(http://www.stsc.hill.af.mil /crosstalk/2003/04/index.html).*

The People Capability Maturity Model (People CMM®) is a framework that guides organizations in improving their processes for managing and developing their workforces. Based on the best current practices in fields such as human resources, knowledge management, and organizational development, the People CMM helps organizations characterize the maturity of their workforce practices, establish a program of continuous workforce development, set priorities for improvement actions, integrate workforce development with process improvement, and establish a culture of excellence.

Like other staged Capability Maturity Models developed at the SEI, the People CMM consists of maturity levels that establish successive foundations for continuously improving workforce competencies. These range from the Initial Maturity Level (Level 1), where workforce practices are performed inconsistently or ritualistically and frequently fail to achieve their intended purpose, to the Optimizing Maturity Level (Level 5), where everyone in the organization is focused on continuously improving their capability and the organization's workforce practices. The architecture of the People CMM, Version 2, is shown in Figure 1.

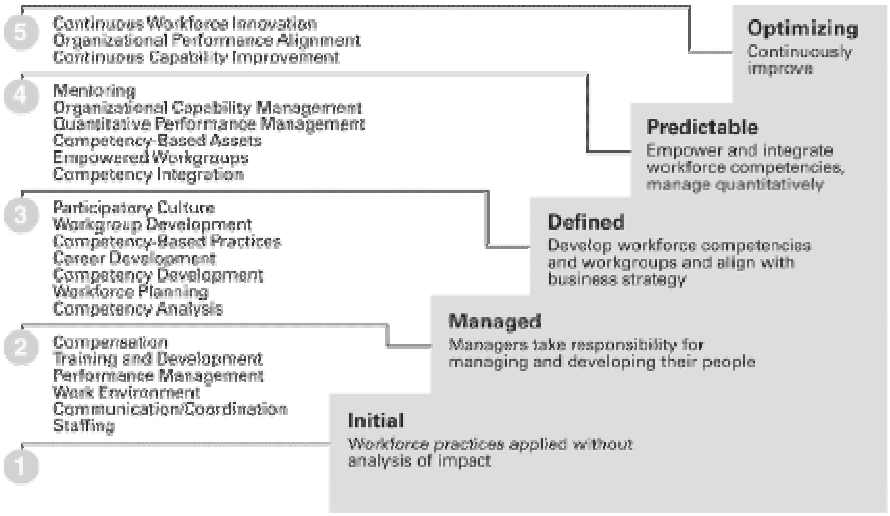

Figure 1: People CMM Version 2, Architecture and Process Areas

Adoption of the People CMM

The People CMM was originally released in 1995. In 2002, Version 2 of the People CMM was released to add enhancements learned from seven years of implementation experience and to integrate the model better with Capability Maturity Model Integration (CMMI®) and its Integrated Product and Process Development (IPPD) extensions. Version 2 now initiates process-driven Workgroup Development at Level 3. This is consistent with the placement of team-building activities at Level 3 of CMMI-IPPD. Also, a specific institutionalization goal was added to each process area to better align the goal structure with that used in CMMI. These improvements make it easier for organizations to integrate People CMM improvements with CMMI-based improvements.

Early adoption of the People CMM occurred primarily in organizations that had already adopted the Capability Maturity Model for Software (SW-CMM). Among the earliest adopters were aerospace companies such as The Boeing Company, Lockheed Martin Corporation, and GDE Systems (now BAE Systems). More recently, government agencies such as the Federal Emergency Management Agency have adopted the People CMM to address the government's objective of raising the performance and capability of the federal workforce.

Intel Information Technology, which supports the computing needs of more than 80,000 employees in some 70 sites worldwide, reported in 2003 on its implementation of the People CMM. "After investigating several different ideas, we decided the People CMM was the most appropriate for our objectives of developing a world-class workforce and organizational capabilities for IT by strategically shaping our future workforce and influencing our partners and industry. The People CMM assessment conducted in the third quarter of every year provides IT with a strategic road map for implementing areas for improvement."

Benefits Achieved

The benefits of implementing the People CMM differ by the maturity level attained. Organizations achieving the People CMM Level 2 uniformly report increases in workforce morale and reductions in voluntary turnover (see Figure 2). These results are not surprising, since years of research have shown that one of the best predictors of voluntary turnover is employees' relationship with their supervisors. The primary change at Level 2 is to help unit managers to develop repeatable practices based on committed work for managing the people who report to them and to ensure that the skill needs of their units are met.

Company	Initial Turnover	Level 2 Turnover
Boeing BRS	1998 7%	1999 5%
Novo Nordisk	1996 12%	2000 8%
GDE Systems	1996 7.8%	1998 7.1%

Figure 2: Annualized Voluntary Turnover

Organizations that achieve Level 3 experience productivity gains associated with developing the workforce competencies required to conduct their business activities. For example, Infosys (see Figure 3) reported a significant correlation of the level of competency among the members of software development groups at Infosys with the project's cost of quality (rework).

That is, the more competent the members of a development team are in the knowledge and skills related to the technology and application on an effort, the less rework groups will experience. At Level 4, an organization begins to achieve what W. Edwards Deming referred to as profound knowledge about the impact of its workforce practices on its workforce capability and on the performance of its business processes. This knowledge enables management to make strategic decisions regarding future investments in workforce practices.

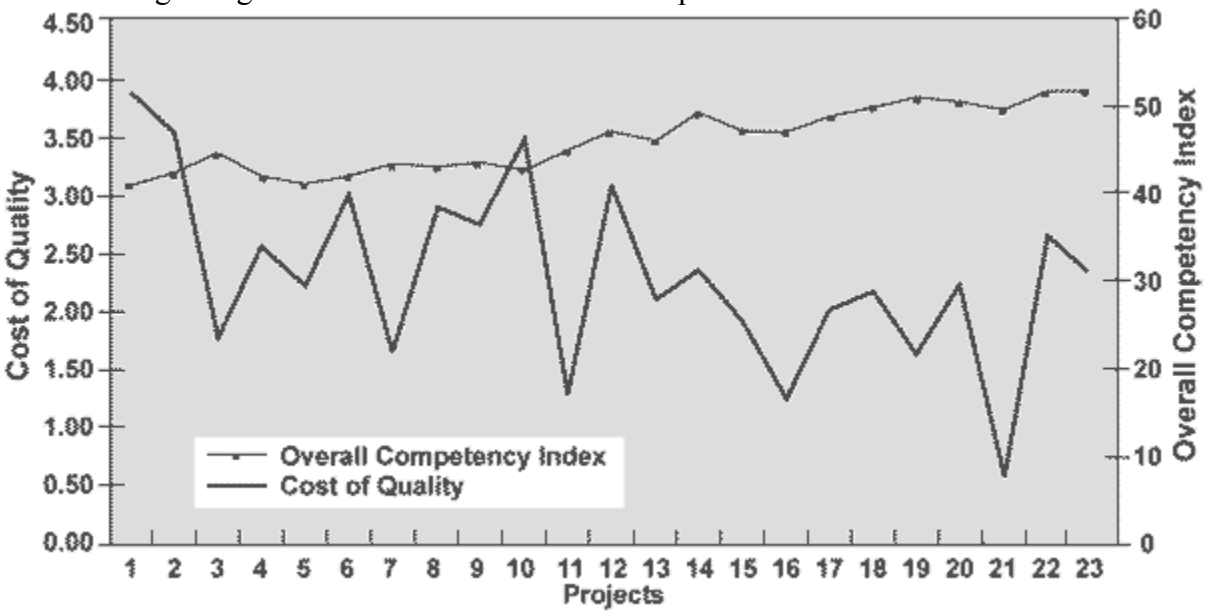

Figure 3: Correlation of Competencies with Cost of Quality at Infosys

Lessons Learned in Applying the People CMM

The CMMs that have been integrated into CMMI all concern behavior performed in or on behalf of projects, whereas the People CMM concerns behavior performed throughout the organization.

Consequently, People CMM-based improvement programs should be conducted as part of an overall organizational improvement strategy. A program based on the People CMM should not be treated as a human resources initiative. Rather, it should be presented as a program for operational management to improve the capability of its workforce. Professionals in human resources, training, organizational development, and related disciplines can assist operational managers in improving their workforce practices after an assessment. Nevertheless, the responsibility for ensuring that an organization has a workforce capable of performing current and future work lies primarily with operational management. The People CMM supplies the roadmap that operational management can use to develop the workforce needed to meet their strategic business needs.

Obtaining the People CMM

The People CMM is available as both a technical report from the SEI and as a book, *The People Capability Maturity Model®: Guidelines for Improving the Workforce*, published by Addison-Wesley. For more information, see the People CMM Web site.

For more information, contact—

Customer Relations
Phone
412-268-5800
Email
customer-relations@sei.cmu.edu
World Wide Web
http://www.sei.cmu.edu/cmm-p/?ns