# FAQs Part 3: Exploring the Issues More Deeply

NEWS AT SEI

Author

Paul C. Clements

This library item is related to the following area(s) of work:

Software Product Lines

This article was originally published in News at SEI on: January 1, 2006

Leveraging core assets (software, designs, documentation, test artifacts, budgets and schedules, tools, and more) across a family of systems—instead of building each system separately—enables organizations to dramatically increase quality and reduce cost and time to market. But what kinds of organizations are most successful at building software product lines, and must organizations pursue process improvement to initiate product line practice? This is the third in a series of columns to answer some of the questions most frequently asked by organizations considering a product line approach.

*"Do successful software product line organizations have certain traits in common?"*

They do indeed. The most successful ones are comfortable with process discipline, have a strong and tireless champion for the approach in a position of leadership, and have lengthy and broad experience in the application areas covered by the product line. Less tangible but just as observable is one other trait, something we've nicknamed "the *e pluribus unum effect*." *E pluribus unum* is Latin, of course, for "out of many, one." All of the most successful product line organizations we've observed regard their primary missions as building and maintaining the product line (singular) or equivalently, a production capability. Organizations that have still not fully embraced the product line approach tend to describe themselves as turning out products (plural). The difference in mindset is subtle but significant. Companies with the singular outlook realize that they have built a product capability that transcends any one product, or even any group of products. To them, turning out products is easy.

*"What's the difference between domain analysis and requirements analysis?"*

Related Links

**Training**

Software Product Lines - eLearning

Software Product Lines

See more related courses »

(< 5 minute) survey.

Domain analysis tends to be broader in scope than requirements analysis. Requirements analysis usually focuses on a specific application while domain analysis looks at the broader requirements for a family of applications. For example, domain analysis would emphasize commonality and variability across a family of products whereas requirements analysis would not. In a product line, however, this distinction is not so sharply drawn, since requirements analysis is often performed for the set of products as a whole during the scoping exercise.

*"What is the relationship between process improvement and product line practice?"*

Process improvement is broader in scope. Product line practice focuses on how to use a common set of core assets to modify, assemble, instantiate, or generate multiple products that are referred to as a product line. The focus is on improvement in product management. Process improvement applies to software engineering in general. While there is overlap between process improvement and product line practice in key practice areas, product line practices provide guidance for one particular approach to software development. However, experience shows that an organization with poorly defined software processes will not fare well in the transition to product line practice. Consequently, organizations need to apply product line practice and process improvement concurrently. As organizations improve their processes, they often enjoy productivity increases. Organizations with higher process maturity can continue to make productivity increases by turning their attention to product line practices.

*"Must all products in the software product line share the same architecture? If my products have different architectures but make heavy use of other shared assets, isn't that a software product line?"*

In theory, yes. However, to date, in all of the software product lines we've studied, all products share either the same architecture or slight variations of the same architecture (e.g., by replacing one component with a similar one, by instantiating a multiple component a different number of times, or by exhibiting some subset of the overall architecture).

### Next column: Product lines in the context of acquisition

### About the Author

Paul Clements is a senior member of the technical staff at the SEI, where he has worked for 10 years leading or co-leading projects in software product line engineering and software architecture design, documentation, and analysis. Clements is the co-author of three practitioner-oriented books about software architecture: *Software Architecture in Practice* (1998; second edition, 2003), *Evaluating Software Architectures: Methods and Case Studies* (2001), and *Documenting Software Architectures: View and Beyond* (2002). He also co-wrote *Software Product Lines: Practices and Patterns* (2001), and was co-author and editor of *Constructing Superior Software* (1999). In addition, Clements has also written dozens of papers in software engineering reflecting his long-standing interest in the design and specification of challenging software systems. He received a BS in mathematical sciences in 1977 and an MS in computer science in 1980, both from the University of North Carolina at Chapel Hill. He received a PhD in computer sciences from the University of Texas at Austin in 1994.

The views expressed in this article 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.

## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# Three Perspectives Required of Service-Oriented Architectures

NEWS AT SEI

Authors

Grace Lewis

Dennis B. Smith

This library item is related to the following area(s) of work:

Service-Oriented Architecture

This article was originally published in News at SEI on: January 1, 2005

Related Links

**Training**

Migrating Legacy Systems to SOA Environments - eLearning

Service-Oriented Architecture: Best Practices for Successful Adoption

See more related courses »

A previous column described the role of service-oriented architectures (SOAs) in a modern information technology environment. With the advent of universal Internet availability, many organizations have leveraged the value of their legacy systems by exposing all or parts of them as services. A service is a coarse-grained, discoverable, and self-contained software entity that interacts with applications and other services through a loosely coupled, often asynchronous, message-based communication model [Lewis 05]. A collection of services with well-defined interfaces and shared communications model is called a service-oriented architecture (SOA). A system or application is designed and implemented using functionality from these services. The characteristics of SOAs (e.g., loose coupling, published interfaces, standard communication model) offer the promise of enabling existing legacy systems to expose their functionality as services, presumably without making significant changes to the legacy systems.

However, constructing services from existing systems to obtain the benefits of an SOA is neither easy nor automatic. There are no effective published strategies for building end-to-end systems[1] based on SOAs; there are no approaches for understanding end-to-end quality of service (QoS); the technologies that SOAs are based on are still immature, and it is not clear what works and what does not work [Ma 05, Manes 05]. This column distinguishes three different perspectives that should be addressed in developing an effective SOA or in developing a component of an SOA-based system.

For each perspective, it identifies the issues, tasks, and risks and outlines a set of research issues.

### Development Perspectives in SOAs

The three different development perspectives are illustrated in Figure 1.

1. Middleware or infrastructure providers (shown in the middle level of Figure 1) must identify the network and communications protocols and standards to be employed. They must also determine what additional SOA infrastructure capabilities are necessary and provide them as common services (e.g., service registry, service-orchestration mechanisms). Infrastructure providers must allow the discovery of services and the data exchange between application and services.
2. Application developers (shown at the top level of Figure 1) must locate or select appropriate services to be used by the application and develop application-specific code to invoke the selected services. They are concerned with whether services invoked by the application meet a full range of capability, QoS, and efficiency-of-use expectations.
3. Service providers (shown at the bottom level of Figure 1) must identify a needed service, identify legacy code that potentially can satisfy the needed service, develop the appropriate interfaces, and finally modify the legacy code and/or develop new code to provide the service so that it has useful capability to the widest range of applications possible.

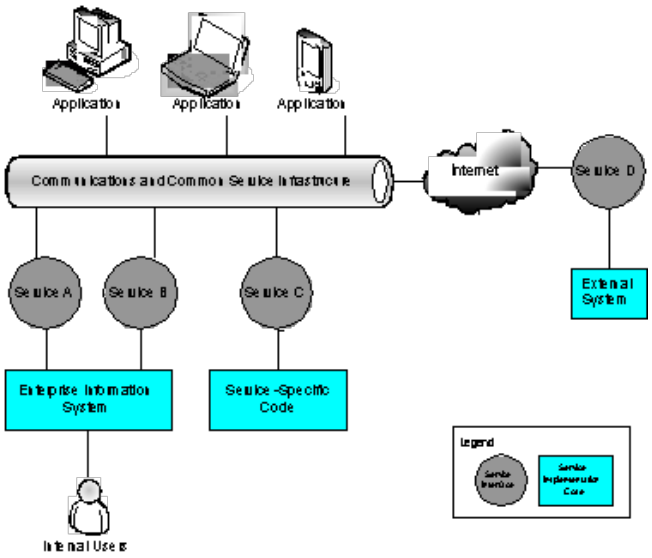



*Figure 1: High-Level View of an SOA*

The focus, tasks, and risks that must be addressed by each perspective are outlined in the following sections.

#### 1. Middleware or Infrastructure Providers

Developers of SOA infrastructure or middleware must provide a stable infrastructure for the discovery and invocation of services. Specific challenges include

- development of a set of common infrastructure services for discovery, communication, security, etc.
- identification and development of binding mechanisms to satisfy the largest set of potential service users

Based on these challenges, the tasks for infrastructure developers are

- development of a set of common infrastructure services for discovery, communication, security, etc.
- provision of tools for application and service developers

Infrastructure developers face some risks:

- Constraints of the infrastructure may limit the availability of services and applications that use it.
- The effort for development, support, and training for the use of tools and infrastructure may be underestimated.
- Delays in schedule will affect the schedule of applications and services that depend on the infrastructure.
- Changes in the infrastructure will affect the applications and services that use it.

### 2. Application Developers

Developers of applications that use services must either discover and connect to services statically at design time or build the applications so that services can be located and invoked dynamically at run time. The semantics of the information being exchanged are always a challenge, as is the question of what to do when a service is no longer available in the case of static binding. Specific challenges for application developers include

- identification of the right services
- understanding of the semantics of the information being exchanged
- determination of rules to follow when services are no longer available (in the case of static binding)
- creation of an architecture that is stable enough to accommodate changes in services that are often outside of the control of the organization

Tasks for developers of applications that use services include

- understanding the SOA infrastructure: bindings, messaging technologies, communication protocols, service-description languages, and discovery services
- discovering services to be incorporated into applications[2]
- retrieving service-description documentation—description, parameters, bindings, transport protocol
- invoking the identified services in applications
  - composition
  - service request
  - error handling
  - availability handling

Application developers, therefore, face these risks:

- Available services might not meet functional and non-functional application requirements.
- Services may change or disappear without notification.
- With proprietary SOAs, there might be dependencies on tools and programs provided by the infrastructure developers that require training and that may conflict with the development and deployment environments.

### 3. Service Providers

Developers of services must describe and develop services that applications can easily locate and use with acceptable QoS. If there are existing systems that can provide service functionality, developers must focus on the development of proper interfaces, the extraction of information from the systems, and the wrapping of this information according to the requirements of the SOA infrastructure. If services are non-existent, developers must focus on the service-specific code that must be written or reused from legacy systems. In this case, there are additional issues of service initialization. If code is reused, developers must analyze the feasibility of migration from legacy to target. Specific challenges for service providers include

- mapping of service requirements to component capabilities, often having to anticipate requirements because the complete set of potential service users is not

known

- description and granularity of services with acceptable QoS
- development of new service-specific code and wrappers to existing code
- determination of migration feasibility of potential services from legacy applications, if required

Tasks for service developers, therefore, include

- gathering requirements from potential service users: Who would use the services and how would they use them?
- understanding the SOA infrastructure: bindings, messaging technologies, communication protocols, service-description languages, and discovery services
- developing code that receives the service request, translates it into calls into existing systems, and produces a response
- describing and publishing the service
- developing service-initialization code and operational procedures

Risks faced by service providers are these:

- Developed services may not be used because they do not meet functional and/or non-functional requirements.
- The effort to translate legacy data types into data types that are allowed by the SOA infrastructure can be greater than expected, especially in the case of complex data types such as audio, video, and graphics.
- If dealing with proprietary SOAs,
  - there may be multiple constraints imposed on developed services; and
  - there might be dependencies on tools and programs provided by the infrastructure developers that require training and that may conflict with the development and deployment environments.

### Why End-to-End SOA Engineering is Important

In a system-of-systems (SoS) context, it is common for these three types of components to be developed independently by separate organizations. This, in fact, is the idea behind SOAs: loose coupling and separation of concerns. Nonetheless, decisions made locally by any one of these development groups can have an effect on the other groups and can potentially have a global effect on the SoS:

- The granularity of service interfaces can affect the end-to-end performance of an SoS because services are executed across a network as an exchange of a service request and a service response. If service interfaces are too coarse-grained, clients will receive more data than they need in their response message. If service interfaces are too fine-grained, clients will have to make multiple trips to the service to get all the data they need.
- If developers of the SOA infrastructure offer only an asynchronous communication mechanism, system developers requiring high performance and availability will encounter problems.
- If service developers do not gather functionality and QoS needs from potential users of services, they might develop and deploy services that are never used.
- If service interfaces are constantly changing and there is no formal mechanism for communicating these changes, users of the SOA-based system might find that certain system functionality is no longer available because the system was not able to anticipate and accommodate these changes.

### Conclusions and Next Steps

Most literature and analyses related to SOAs focus on only one of the development perspectives and tend to assume an ideal environment in which there are no conflicts among the three perspectives. The SEI is beginning to develop a research agenda to

examine each of the perspectives—application developer, infrastructure developer, and service provider—and to answer questions such as how to select the appropriate services and infrastructure for the organization's system goals, how to determine the QoS delivered by a system when some of its components are discovered and composed at runtime, and how to build services that can be used in a wide range of applications. All of these three perspectives require awareness of the needs and challenges of the others so they can contribute overall to the quality of the SOA-based system. Addressing these issues can ultimately lead to a disciplined approach to building end-to-end systems based on SOAs. Such an approach must focus on an interrelated set of critical issues, including

- QoS in a system where service discovery, composition, and invocation play a major role
- insights on paths and limitations of technologies that enable SOAs
- understanding critical functional and QoS needs of potential users of services
- service-level agreements in a dynamic environment

## References

[Lewis 05]
Lewis, Grace and Wrage, Lutz. *Approaches to Constructive Interoperability* (CMU/SEI-2004-TR-020). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

[Ma 05]
Ma, Kevin. "Web Services: What's Real and What's Not." *IT Professional 7*, 2 (March-April 2005).

[Manes 05]
Manes, Anne. *VantagePoint 2005-2006 SOA Reality Check*. Midvale, UT: Burton Group, 2005.

1 By end-to-end, we mean the complete pathway through applications, the communication infrastructure and network, and the actual services to perform a specific task.

2 Dynamic discovery, composition, and invocation are still extremely limited with existing technologies.

## About the Authors

Grace Lewis is a senior member of technical staff at the Software Engineering Institute (SEI) of Carnegie Mellon University (CMU), where she is currently working in the areas of constructive interoperability, COTS-based systems, modernization of legacy systems, enterprise information systems, and model-driven architecture. Her latest publications include several reports published by Carnegie Mellon on these subjects and a book in the SEI Software Engineering Series. Grace has more than 15 years of experience in software engineering. She is also a member of the technical faculty for the Master's in Software Engineering program at CMU.

Dennis Smith is the lead for the SEI Integration of Software Intensive Systems (ISIS) Initiative. This initiative focuses on addressing issues of interoperability and integration in large-scale systems and systems of systems. Earlier, he was the technical lead in the effort for migrating legacy systems to product lines. In this role he developed the method Options Analysis for Reengineering (OARS) to support reuse decision-making. Smith has also been the project leader for the CASE environments project. This project examined the underlying issues of CASE integration, process support for environments and the adoption of technology. Smith has published a wide variety of articles and technical reports, and has given talks and keynotes at a number of conferences and workshops. He has an MA and PhD from Princeton University, and a BA from Columbia University.

topic.



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# Three Perspectives of Service-Oriented Architectures

NEWS AT SEI

Authors

Grace Lewis

Dennis B. Smith

This library item is related to the following area(s) of work:

Performance and Dependability

This article was originally published in News at SEI on: January 1, 2006

Related Links

**Training**

Software Architecture Design and Analysis

Modeling System Architectures Using the Architecture Analysis and Design Language (AADL)

See more related courses »

A previous column described the role of service-oriented architectures (SOAs) in a modern information technology environment. With the advent of universal Internet availability, many organizations have leveraged the value of their legacy systems by exposing all or parts of them as services. A service is a coarse-grained, discoverable, and self-contained software entity that interacts with applications and other services through a loosely coupled, often asynchronous, message-based communication model [Lewis 05]. A collection of services with well-defined interfaces and shared communications model is called a service-oriented architecture (SOA). A system or application is designed and implemented using functionality from these services. The characteristics of SOAs (e.g., loose coupling, published interfaces, standard communication model) offer the promise of enabling existing legacy systems to expose their functionality as services, presumably without making significant changes to the legacy systems.

However, constructing services from existing systems to obtain the benefits of an SOA is neither easy nor automatic. There are no effective published strategies for building end-to-end systems[1] based on SOAs; there are no approaches for understanding end-to-end quality of service (QoS); the technologies that SOAs are based on are still immature, and it is not clear what works and what does not work [Ma 05, Manes 05]. This column distinguishes three different perspectives that should be addressed in developing an effective SOA or in developing a component of an SOA-based system.

For each perspective, it identifies the issues, tasks, and risks and outlines a set of research issues.

(< 5 minute) survey.

### Development Perspectives in SOAs

The three different development perspectives are illustrated in Figure 1.

1. Middleware or infrastructure providers (shown in the middle level of Figure 1) must identify the network and communications protocols and standards to be employed. They must also determine what additional SOA infrastructure capabilities are necessary and provide them as common services (e.g., service registry, service-orchestration mechanisms). Infrastructure providers must allow the discovery of services and the data exchange between application and services.
2. Application developers (shown at the top level of Figure 1) must locate or select appropriate services to be used by the application and develop application-specific code to invoke the selected services. They are concerned with whether services invoked by the application meet a full range of capability, QoS, and efficiency-of-use expectations.
3. Service providers (shown at the bottom level of Figure 1) must identify a needed service, identify legacy code that potentially can satisfy the needed service, develop the appropriate interfaces, and finally modify the legacy code and/or develop new code to provide the service so that it has useful capability to the widest range of applications possible.

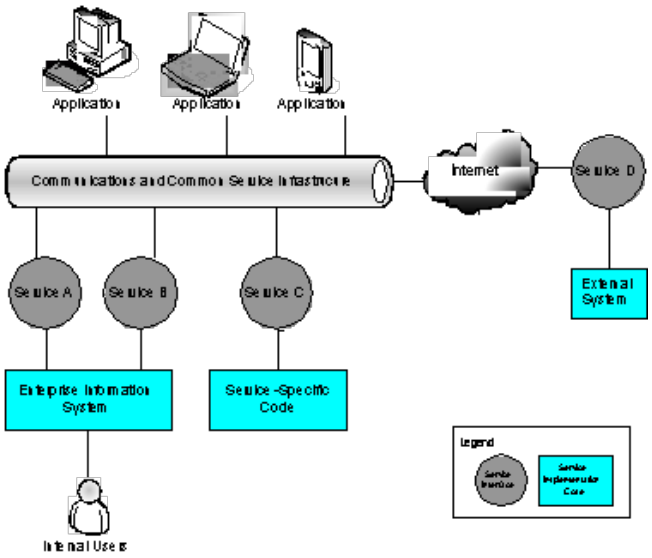



*Figure 1: High-Level View of an SOA*

The focus, tasks, and risks that must be addressed by each perspective are outlined in the following sections.

#### *1. Middleware or Infrastructure Providers*

Developers of SOA infrastructure or middleware must provide a stable infrastructure for the discovery and invocation of services. Specific challenges include

- development of a set of common infrastructure services for discovery, communication, security, etc.
- identification and development of binding mechanisms to satisfy the largest set of potential service users

Based on these challenges, the tasks for infrastructure developers are

- development of a set of common infrastructure services for discovery, communication, security, etc.
- provision of tools for application and service developers

Infrastructure developers face some risks:

- Constraints of the infrastructure may limit the availability of services and applications that use it.
- The effort for development, support, and training for the use of tools and infrastructure may be underestimated.
- Delays in schedule will affect the schedule of applications and services that depend on the infrastructure.
- Changes in the infrastructure will affect the applications and services that use it.

#### *2. Application Developers*

Developers of applications that use services must either discover and connect to services statically at design time or build the applications so that services can be located and invoked dynamically at run time. The semantics of the information being exchanged are always a challenge, as is the question of what to do when a service is no longer available in the case of static binding. Specific challenges for application developers include

- identification of the right services
- understanding of the semantics of the information being exchanged
- determination of rules to follow when services are no longer available (in the case of static binding)
- creation of an architecture that is stable enough to accommodate changes in services that are often outside of the control of the organization

Tasks for developers of applications that use services include

- understanding the SOA infrastructure: bindings, messaging technologies, communication protocols, service-description languages, and discovery services
- discovering services to be incorporated into applications[2]
- retrieving service-description documentation—description, parameters, bindings, transport protocol
- invoking the identified services in applications
  - composition
  - service request
  - error handling
  - availability handling

Application developers, therefore, face these risks:

- Available services might not meet functional and non-functional application requirements.
- Services may change or disappear without notification.
- With proprietary SOAs, there might be dependencies on tools and programs provided by the infrastructure developers that require training and that may conflict with the development and deployment environments.

#### *3. Service Providers*

Developers of services must describe and develop services that applications can easily locate and use with acceptable QoS. If there are existing systems that can provide service functionality, developers must focus on the development of proper interfaces, the extraction of information from the systems, and the wrapping of this information according to the requirements of the SOA infrastructure. If services are non-existent, developers must focus on the service-specific code that must be written or reused from legacy systems. In this case, there are additional issues of service initialization. If code is reused, developers must analyze the feasibility of migration from legacy to target. Specific challenges for service providers include

- mapping of service requirements to component capabilities, often having to anticipate requirements because the complete set of potential service users is not known

- description and granularity of services with acceptable QoS
- development of new service-specific code and wrappers to existing code
- determination of migration feasibility of potential services from legacy applications, if required

Tasks for service developers, therefore, include

- gathering requirements from potential service users: Who would use the services and how would they use them?
- understanding the SOA infrastructure: bindings, messaging technologies, communication protocols, service-description languages, and discovery services
- developing code that receives the service request, translates it into calls into existing systems, and produces a response
- describing and publishing the service
- developing service-initialization code and operational procedures

Risks faced by service providers are these:

- Developed services may not be used because they do not meet functional and/or non-functional requirements.
- The effort to translate legacy data types into data types that are allowed by the SOA infrastructure can be greater than expected, especially in the case of complex data types such as audio, video, and graphics.
- If dealing with proprietary SOAs,
  - there may be multiple constraints imposed on developed services; and
  - there might be dependencies on tools and programs provided by the infrastructure developers that require training and that may conflict with the development and deployment environments.

### Why End-to-End SOA Engineering is Important

In a system-of-systems (SoS) context, it is common for these three types of components to be developed independently by separate organizations. This, in fact, is the idea behind SOAs: loose coupling and separation of concerns. Nonetheless, decisions made locally by any one of these development groups can have an effect on the other groups and can potentially have a global effect on the SoS:

- The granularity of service interfaces can affect the end-to-end performance of an SoS because services are executed across a network as an exchange of a service request and a service response. If service interfaces are too coarse-grained, clients will receive more data than they need in their response message. If service interfaces are too fine-grained, clients will have to make multiple trips to the service to get all the data they need.
- If developers of the SOA infrastructure offer only an asynchronous communication mechanism, system developers requiring high performance and availability will encounter problems.
- If service developers do not gather functionality and QoS needs from potential users of services, they might develop and deploy services that are never used.
- If service interfaces are constantly changing and there is no formal mechanism for communicating these changes, users of the SOA-based system might find that certain system functionality is no longer available because the system was not able to anticipate and accommodate these changes.

### Conclusions and Next Steps

Most literature and analyses related to SOAs focus on only one of the development perspectives and tend to assume an ideal environment in which there are no conflicts among the three perspectives. The SEI is beginning to develop a research agenda to examine each of the perspectives—application developer, infrastructure developer, and

service provider—and to answer questions such as how to select the appropriate services and infrastructure for the organization's system goals, how to determine the QoS delivered by a system when some of its components are discovered and composed at runtime, and how to build services that can be used in a wide range of applications. All of these three perspectives require awareness of the needs and challenges of the others so they can contribute overall to the quality of the SOA-based system. Addressing these issues can ultimately lead to a disciplined approach to building end-to-end systems based on SOAs. Such an approach must focus on an interrelated set of critical issues, including

- QoS in a system where service discovery, composition, and invocation play a major role
- insights on paths and limitations of technologies that enable SOAs
- understanding critical functional and QoS needs of potential users of services
- service-level agreements in a dynamic environment

## References

[Lewis 05]
Lewis, Grace and Wrage, Lutz. *Approaches to Constructive Interoperability* (CMU/SEI-2004-TR-020 ESC-TR-2004-020). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

[Ma 05]
Ma, Kevin. "Web Services: What's Real and What's Not." *IT Professional 7*, 2 (March-April 2005).

[Manes 05]
Manes, Anne. *VantagePoint 2005-2006 SOA Reality Check*. Midvale, UT: Burton Group, 2005.

1 By end-to-end, we mean the complete pathway through applications, the communication infrastructure and network, and the actual services to perform a specific task.

2 Dynamic discovery, composition, and invocation are still extremely limited with existing technologies.

## About the Authors

Grace Lewis is a senior member of technical staff at the Software Engineering Institute (SEI) of Carnegie Mellon University (CMU), where she is currently working in the areas of constructive interoperability, COTS-based systems, modernization of legacy systems, enterprise information systems, and model-driven architecture. Her latest publications include several reports published by Carnegie Mellon on these subjects and a book in the SEI Software Engineering Series. Grace has more than 15 years of experience in software engineering. She is also a member of the technical faculty for the Master's in Software Engineering program at CMU.

Dennis Smith is the lead for the SEI Integration of Software Intensive Systems (ISIS) Initiative. This initiative focuses on addressing issues of interoperability and integration in large-scale systems and systems of systems. Earlier, he was the technical lead in the effort for migrating legacy systems to product lines. In this role he developed the method Options Analysis for Reengineering (OARS) to support reuse decision-making. Smith has also been the project leader for the CASE environments project. This project examined the underlying issues of CASE integration, process support for environments and the adoption of technology. Smith has published a wide variety of articles and technical reports, and has given talks and keynotes at a number of conferences and workshops. He has an MA and PhD from Princeton University, and a BA from Columbia University.

Find Us Here

Share This Page

## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# CMMI V1.2: What's Changing? (Part 4)

NEWS AT SEI

Author

Mike Phillips

This library item is related to the following area(s) of work:

Process Improvement

CMMI

This article was originally published in News at SEI on: February 1, 2006

In earlier columns, I described most of the intended improvements considered for V1.2. But now I can describe the major changes the CMMI Development Team has enacted to simplify and clarify what many are using today in V1.1.

### Three Fewer Process Areas and a Simpler Model for V1.2

The team observed that more than 80% of the appraisals using V1.1 models did not choose Integrated Product and Process Development (IPPD) or Supplier Sourcing (SS), but instead used models that addressed only the systems engineering and software engineering disciplines. This was the overall level of usage despite the widespread use of team-based development and complex multi-company developments, which suggest that these organizations would benefit from the practices in the IPPD and SS additions. Consolidating the material within IPPD and SS would improve the use of their associated practices while simplifying the models.

Many change requests from CMMI users suggested combining Integrated Supplier Management (ISM) with Supplier Agreement Management (SAM). ISM was designed to include more proactive practices and monitoring activities that would result in process understanding across organizations, unlike the practices in a SAM. However, the significant overlap between ISM and SAM, was troubling enough to users that they submitted change requests. After careful analysis of the overlap, we strengthened the informative material in SAM to include ideas from ISM and added the following two specific practices that address the proactive monitoring of supplier progress formerly contained in ISM:



Related Links

**News**

SEI to Co-Sponsor 26th Annual IEEE Software Technology Conference

SEI Cosponsors Agile for Government Summit

**Training**

See more related courses »

**Events**

Team Software Process (TSP) Symposium 2014

Nov 3 - 6

- Monitor selected supplier processes.
- Evaluate selected supplier work products.

Since one specific practice in SAM, "Analyze COTS" was being refocused as informative material in both SAM and Technical Solution (TS), the end result of changes to SAM is that its number of specific practices has increased by one.

Since SAM is part of the base model and ISM was eliminated (its contents either incorporated into SAM or eliminated), the Supplier Sourcing addition no longer exists. Therefore, in V1.2 only IPPD can be optionally selected for the model used in a process improvement program.

When we considered consolidating and simplifying the IPPD material in CMMI, we first realized that some of the V1.1 IPPD material was not actually specific to the use of IPPD. Much of the material identified as being more general was distributed to other process areas as informative material. We then reviewed the IPPD-specific material to first simplify the text and to place it appropriately. In V1.1, the IPPD material consisted of two process areas, Integrated Teaming (IT) and Organizational Environment for Integration (OEI) as well as two goals in the Integrated Project Management (IPM) process area. All of this material was used only if the IPPD option was selected by the organization for its process improvement program.

For V1.2, we determined that the placement of IPPD material in the model could be simplified if we added a goal to Organizational Process Development (OPD) to address the organizational commitment to IPPD and consolidated the integrated team concepts in IT into IPM. This simpler approach has greatly reduced the number of IPPD-related practices-and process areas. IPPD will now be addressed by including one additional goal in OPD (for the organizational behaviors), and one additional goal in IPM (for the project behaviors). The two goals are

- Enable IPPD management (in OPD).
- Apply IPPD principles (in IPM).

### Hardware Engineering and Work Environment Coverage

A hardware engineering team looked for ways to assure that V1.2 adequately addressed hardware aspects sometimes perceived to be missing from earlier CMMI versions. Much of this work is now reflected in new examples in a number of process areas, sometimes within hardware amplifications and sometimes in notes. This addition of hardware examples also resulted in a reduction in the number of discipline-specific amplifications; we considered it better to combine the examples into notes rather than separating them along discipline lines. The result is six specific hardware amplifications (there were none in V1.1), four systems engineering amplifications (the same as in V1.1), and eight software engineering amplifications (there were 11 in V1.1).

A potential Work Environment process area was proposed, but further investigation and analysis revealed that we could cover the basics as we had in the past for data management by creating two practices that address the work environment. These two practices are contained in the same process areas that we used to address IPPD concepts-OPD and IPM. A new practice in OPD will focus organizational attention on effective work-environment standards, and IPM will focus on tailoring these standards to individual projects. These two specific practices are

- Establish work environment standards (in OPD).
- Establish the project's work environment (in IPM).

### "Not Applicable" Process Areas

With the release of V1.2, we will narrow the potential for maturity-level variability. In both V1.0 and V1.1, we took time to describe in Chapter 6 that process areas could be determined "not applicable" for organizational process improvement. One of the legacy models, the SW-CMM, had always allowed Software Subcontract Management (SSM) to be considered "not applicable." The CMMI equivalent, Supplier Agreement Management (SAM), was cited in the Chapter 6 discussion as an example in CMMI of a process area that could be declared "not applicable." However, the text did not limit

consideration of declaring a process area to be "not applicable" only to SAM.

The V1.2 model will no longer discuss "not applicable" status. Instead, that issue will be discussed only as part of the SCAMPI Method Definition Document. For SAM to be successfully declared "not applicable," the lead appraiser must agree. We will rely on the lead appraiser, with help from the appraisal team, to determine, before the appraisal onsite, the relevancy of SAM practices to the organizational unit being appraised. The appraisal disclosure statement will include a statement about the absence of suppliers needing management if SAM is determined to be "not applicable."

### Appraisal Validity Period

The CMMI Steering Group has determined that some sense of lifetime needed to be defined for CMMI appraisals. After extended discussions, the Steering Group determined that a three-year validity period, similar to that established for ISO 9000:2000, would be the most reasonable lifetime for CMMI appraisal results.

This new validity period for appraisal results will be phased in. First, after the release of V1.2, all appraisals, both V1.1 and V1.2, will be considered valid for three years from the date of their completion, as noted on the Appraisal Disclosure Statement. When two years have passed without new appraisals, the SEI will contact the sponsors to remind them that the appraisal results are valid for one more year.

There is a different approach for appraisals that have already been conducted. Here the planned availability of V1.2 causes a need for flexibility, to encourage a smooth transition to the improved version. Therefore, for existing appraisals older than three years, we will continue to recognize them for a full year after the planned summer release of V1.2. This approach is designed to allow time to plan and execute appraisals using the V1.2 product suite. Further, we will continue to recognize V1.1 appraisals through December 2007 in case the concerns about change are greater than what we currently expect.

Although we do not make SW-CMM appraisal results publicly available, we deemed it appropriate to establish a validity period for these as well. Since all recognized appraisals had to be completed by December 2005, we chose a single date-the same date on which V1.1 appraisals end, December 2007. This leaves CMM users with some flexibility as well, with more than a year and a half to make the transition to CMMI and to use either V1.1 or V1.2.

### Summary

We are entering the piloting period for V1.2 now with the updates I described above. Some of them may change slightly as a result of the feedback we receive from piloting. We will keep you up to date on progress as we move closer to the publication of V1.2. We now expect that we will be offering a clarified version of the model this summer-with changes many of you have told us would aid your adoption of the CMMI Product Suite.

In the next column, I'll be able to provide more detail on the changes for V1.2, as well as our initial impressions from piloting feedback.

### About the Author

As the director of special projects at the Software Engineering Institute, Mike Phillips leads 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, Ohio. In addition to his bachelor's degree in astronautical engineering from the U.S. Air Force Academy, Phillips has master's 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.

Share This Page

## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# How Much Security Is Enough?

NEWS AT SEI

Author

Julia H. Allen

This article was originally published in News at SEI on: February 1, 2006

CIOs, CSOs, and system administrators may dream about achieving a state of complete organizational security, but pragmatically they realize this is unrealistic and financially imprudent. However, it is feasible to adopt the concept of achieving adequate security at an enterprise level in response to the question, “How much security is enough?”

Achieving adequate security means more than complying with regulations or implementing commonly accepted best practices. Formulating the concept of adequate security helps define the benefit and optimized outcome for security investment. This formulation must occur in the context of identifying and managing the security risks to an organization’s mission and objectives.

One approach to defining an adequate or appropriate level of security is to compare and contrast it with a theoretical state of absolute security—an ideal condition where all security requirements for critical business processes and assets are satisfied (assuming that an organization has identified these as worthy investments).

So, in this context, how might we define and determine adequate security, recognizing that

- absolute security is not only impossible but highly undesirable from effectiveness, efficiency, risk/reward, and cost/benefit perspectives
- governing security at an adequate or appropriate level enables enterprise risk to be managed in a cost-effective manner

Determining adequate security is largely synonymous with determining and managing risk. Where possible, an organization implements controls that satisfy the security requirements for its critical business processes and assets. Where this is not possible, security risks to such processes and assets are identified, mitigated, and managed at a level of residual risk that is acceptable to the organization.

(< 5 minute) survey.

Determining adequate security depends on what an organization needs to protect and what it needs to prevent to support achievement of enterprise objectives. Consider the following questions from an enterprise perspective, not just an IT perspective:

- What needs to be protected? Why does it need to be protected? What happens if it is not protected?
- What potential adverse consequences need to be prevented? At what cost? How much disruption can we stand before we take action?
- How do we effectively manage the residual risk when protection and prevention actions are not taken?

Selecting protection and prevention actions based on risk helps determine how much to invest, where to invest it, and how fast.

### Defining Adequate Security

We define adequate security as

> *The condition where the protection strategies for an organization's critical assets and business processes are commensurate with the organization's risk appetite and risk tolerances.*

*Protection strategies* include principles, policies, procedures, processes, practices, and performance indicators and measures, all elements of an overall system of controls.

An *asset* is anything of value to an organization. Assets include information such as enterprise strategies and plans, product information, and customer information; technology such as hardware, software, and IT-based services; and supporting assets such as facilities and utilities. Critical assets are those that directly affect the ability of the organization to meet its objectives and fulfill its critical success factors [Caralli 04]. As stated earlier, assets also include items of significant yet largely intangible value, such as brand, image, and reputation.

A *process* is a systematic series of progressive and interdependent actions or steps by which a defined end result is obtained. Business processes create the products and services that an organization offers and can include customer relationship management, financial management and reporting, and management of relationships and contractual agreements with partners, suppliers, and contractors.

*Risk appetite* is defined by the Committee of Sponsoring Organizations of the Treadway Commission (COSO) as "... the amount of risk, on a broad level, an entity is willing to accept in pursuit of value (and its mission)." Risk appetite influences the entity's culture, operating style, strategies, resource allocation, and infrastructure [COSO 04]. Risk appetite is not a constant; it is influenced by and must adapt to changes in the environment.

Defining the organization's risk appetite is an executive responsibility. It is undertaken in conjunction with evaluating alternative business models in pursuit of the organization's goals and objectives. Management assesses the alternatives, sets objectives aligned with strategy, develops business processes to accomplish the plan, and manages any inherent risks. Risk appetite can be expressed as impact (potential consequences of a risk-based event), likelihood of a risk's occurrence, and associated mitigating actions [Carey 05]. For identified and evaluated risks, risk appetite could be defined as the residual risk the organization is willing to accept as the default condition of having implemented its set of risk-mitigation and monitoring processes [Taylor 04].

Risk tolerances are defined by COSO as "... the acceptable levels of variation relative to the achievement of objectives, [which] are often best measured in the same units as the related objectives" [COSO 04]. In defining acceptable levels of variation, risk tolerance defines and delineates the range of impact and corresponding risk to the organization. This is embodied in defining and using impact and risk-evaluation criteria, which can be expressed both qualitatively and quantitatively.

Risk tolerance could be defined as the residual risk the organization is willing to accept after implementing risk-mitigation and monitoring processes and controls. One way to implement this is to define high, medium, and low levels of residual risk. An example is a policy to conduct prioritized mitigation for high- and medium-level risks and to accept (monitor) low-level risks as the default condition.

With risk appetite and risk tolerances defined, how does the organization manage different levels of inherent and residual risk? How does an organization prioritize risks requiring mitigating actions? In quantitative terms, what "value at risk" is acceptable [Taylor 04]?

Consider the following example: A retailer decides to enter the e-commerce marketplace but has a low risk appetite relative to its relationship with existing customers, particularly with respect to fulfilling orders promptly and accurately. To protect these relationships, management allocates necessary resources (people, processes, technology) to ensure that (1) order-to-delivery response times meet or exceed defined targets and (2) order-fulfillment accuracy meets or exceeds defined criteria. Management is now conducting business online and has installed the resources needed to protect its reputation for timely and accurate fulfillment of customer orders. It has set a target for delivery within seven days of accepting orders and has guaranteed delivery within two weeks by a statement on its Web site. However, how much variation is management willing to tolerate with respect to delivery and order-accuracy targets? Is a five-day average variance around the delivery target too much? The level of variation relative to achievement of objectives is known as the risk tolerance [Taylor 04].

### Determining Adequate Security

With the benefit of this description, a useful way to address the question "How much security is enough?" is to first ask "What is our definition of adequate security?" by exploring the following more detailed questions:

1. What are the critical assets and business processes that support achieving our organizational goals? What are the organization's risk tolerances and risk appetite, in general and with respect to these assets and processes?
2. Under what conditions and with what likelihood are assets and processes at risk? What are the possible adverse consequences if a risk is realized? Do these risks fit within our risk appetite and risk tolerances?
3. In the cases where risks are beyond these thresholds, what actions do we need to take to mitigate and with what priority? Are we making conscious decisions to accept levels of risk exposure and then effectively managing residual risk? Have we considered mechanisms for sharing potential risk impact (for example, through insurance or with third parties)?
4. For those risks we are unwilling or unable to accept, what protection strategies do we need to put in place? What is the cost/benefit or return on investment of deploying these strategies?
5. How well are we managing our security state today? How well will we manage our security state 30 days, 6 months, and a year from now? Are we updating our understanding and definition of our security state as part of normal planning and review processes?

### Example

One of Acme, Inc.'s, critical assets is the customer-transaction database, which includes order history. This is used actively in targeted marketing and sales processes with exceptional results (repeat sales). It has taken three years of staff effort to build and populate this database at an estimated cost of USD $1 million. Ongoing operations and maintenance costs including the protection strategies described below are USD $200,000.

There are specific events, impacts, and consequences that Acme needs to prevent. Competitors regularly attempt to obtain access to this information or to obtain a copy of this information (high risk). Management is sensitive to the risk of disclosure by sales and marketing staff who are approached by competitors to share this information

for personal financial gain (medium risk). Third-party intruders have threatened to obtain access to and disclose this information on the Internet (low risk). While Acme believes it offers superior service, creating customer loyalty in the face of competitive pressure to switch, it places the value at risk at USD $10 million (risk appetite).

Security requirements for this asset include zero tolerance of unauthorized disclosure (violation of confidentiality), continuous validation of data integrity (by automated comparison with a trusted, securely stored version), and 99.999 percent availability (risk tolerances).

Protection strategies include

- principles enacted by policies and procedures that state these requirements and risk tolerances for this asset
- clear assignment of roles and responsibilities and periodic training for staff and managers involved in protecting this asset; financial incentives for those demonstrating innovative approaches to asset protection
- periodic training for staff having access to this asset; immediate removal of access and authorization for any staff member whose responsibilities no longer require a need for access, including any change in employment status such as termination
- an infrastructure architecture that fulfills these requirements, meets these risk tolerances, and implements effective controls (strong authentication, firewalls including ingress and egress filtering, enforcement of separation of duties, automated integrity checking, hot backups, etc.)
- review of all new and upgraded technologies that provide database support and in-house and remote access, to determine if any of these technologies introduce additional security risks or reduce existing risks. Review occurs before and after technology deployment.
- regular review and monitoring of relevant processes, and performance indicators and measures including financial performance and return on investment; regular review of new and emerging threats and evaluation of levels of risk
- regular audit of relevant controls and timely resolution of audit findings

A level of adequate security as defined here is constantly changing in response to business and risk environments and the variation in risk tolerance that management is willing to accept. Effectively achieving and sustaining adequate security based on this definition is a continuous process, not a final outcome. Thus processes to plan for, monitor, review, report, and update an organization's security state must be part of normal day-to-day business conduct, risk management, and governance. This includes documenting this state and the best anticipation of its evolution as part of strategic and operational plans.

## References

[Caralli 04]
Caralli, Richard. The Critical Success Factor Method: Establishing a Foundation for Enterprise Security Management (CMU/SEI-2004-TR-010). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004.

[Carey 05]
Carey, Mark. "Enterprise Risk Management: How To Jumpstart Your Implementation Efforts." International Risk Management Institute, 2005.

[COSO 04]
The Committee of Sponsoring Organizations of the Treadway Commission. "Enterprise Risk Management—Integrated Framework." September 2004. The executive summary is available online.

[Taylor 04]
Taylor, Jay. Review comments, December 2004.

## About the Author

Julia Allen is a senior member of the technical staff in the Networked Systems

Survivability Program at the Software Engineering Institute (SEI), a unit of Carnegie Mellon University in Pittsburgh, Pa. The CERT Coordination Center is also a part of this program.

Allen is engaged in developing and transitioning enterprise security frameworks and executive outreach programs in enterprise security and governance. Prior to this technical assignment, Allen served as acting director of the SEI for an interim period of six months as well as deputy director/chief operating officer for three years. Her degrees include a BSci in computer science (University of Michigan) and an MS in electrical engineering (University of Southern California). She is the author of *The CERT Guide to System and Network Security Practices* (Addison-Wesley, June 2001).

The views expressed in this article 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 views expressed in this article 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.



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# SEI to Open West Coast Office

NEWS AT SEI

Author

Susan Kushner

This article was originally published in News at SEI on: February 1, 2006

Extending its support of the United States Department of Defense (DoD), civil agencies, and industry, the Carnegie Mellon Software Engineering Institute (SEI) is opening an office in Southern California in 2006. Initially, SEI technical staff at the Southern California site will focus their expertise on the acquisition programs headquartered at the United States Air Force's Space and Missile Systems Center (SMC) at Los Angeles Air Force Base.

In 2005, the SEI supported acquisition efforts for more than 50 projects within the DoD and civil agencies. The principal SEI program that promotes the application of software and systems engineering expertise to the DoD is the Acquisition Support Program (ASP). ASP works with acquisition programs to help them achieve their objectives by applying software and systems engineering products and services in specific contexts.

## Revolutionizing the Acquisition of Software-Intensive Systems

Acquisition program managers are challenged not only to grasp practical business concerns, but also to understand topics as diverse as risk identification and mitigation, selection and integration of commercial off-the-shelf (COTS) components, process capability, program management, architecture, network survivability, interoperability, source selection, and contract monitoring. The SEI has spent almost two decades compiling a body of knowledge and developing solutions in these areas.

Founded in 2002, ASP helps DoD and other government acquirers make evolutionary and revolutionary improvements in the acquisition of software-intensive systems and provides opportunities to create, apply, and extend new technologies. According to ASP director Brian Gallagher, "The SEI is in the trenches with key acquisition programs to help them define innovative acquisition strategies, identify and mitigate risk, and structure acquisition programs to reduce conflict and establish mutually beneficial relationships among all stakeholders."

(< 5 minute) survey.

## Executing the SMC Mission

SMC is the Air Force's center of excellence for acquiring and developing military space systems. ASP works closely with SMC to fulfill its primary mission: "Deliver unrivaled space and missile systems to the joint warfighter and our nation."

To fulfill this mission, SMC manages over $60 billion in contracts each year and exercised an operating budget of $8.6 billion in 2005. Over 6,800 people are employed by the SMC worldwide, about 1000 of them civilians. SMC currently executes 28 highly visible programs. Among these are

- MILSATCOM (military satellite communications), the primary acquirer of satellite communications systems to satisfy warfighter needs including the
  - advanced extremely high frequency (AEHF) joint service satellite communications system that provides near-worldwide, secure, survivable, and jam-resistant communications for high-priority military ground, sea, and air assets; and the
  - Transformational Satellite Communication (TSAT) System to develop unprecedented satellite communications with Internet-like capability delivering increased situational awareness and targeting information to the warfighter
- NAVSTAR Global Positioning System (GPS) to provide three-dimensional position, velocity, and time data to field forces, civil, and commercial users
- Space Based Infrared System (SBIRS) providing the nation with the very best space-based surveillance capability.

Another SMC project that ASP assists is the Systems Engineering Revitalization (SER) initiative started in 2002. The goal of SER efforts is to apply world-class systems-engineering and program-management practices to SMC's systems acquisitions. Lt. Gen. (ret) Brian A. Arnold, former commander of SMC stated in a 2002 video message to his staff, "Simply put, we must remember that at the core of effective program and acquisition management are disciplined technical oversight and systems engineering. These are things that we can't expect our contractors to do for us. Rather, we must do them ourselves by becoming fully engaged with our contractors and value-added partners in the acquisition of military space systems."

ASP is helping SMC achieve its vision of producing innovative, affordable, operationally effective space systems by providing direct support to several of its acquisition programs. SEI technical staff lead and participate in multi-discipline teams to identify and solve software-acquisition problems using problem-solving research, instrumentation and measurement, and knowledge transfer. These activities include independent technical assessments; pilots; direct support; empirical research including case studies, workshops, and the preparation of lessons learned; presentations; guides; reports; and articles.

## Seeking Talent for Deployment in Southern California

Since 1984, the SEI has been identifying, developing, and advocating practices to improve all aspects of software. Its mission is to advance software engineering and related disciplines to ensure the development and operation of systems with predictable and improved cost, schedule, and quality. The SEI currently is seeking talented technical personnel to support its West Coast endeavors and, specifically, wants to fill several positions in 2006.

Staff members in the Southern California office will have an opportunity to understand government acquisition needs, apply new technologies, and establish capabilities to meet the needs of sponsoring organizations and the acquisition community.

Candidates applying for a position in ASP should have a bachelor's degree in computer science, information systems, systems engineering, software engineering, or acquisition management; experience; a proven track record in the development and acquisition of software-intensive systems; and detailed knowledge in at least one core competency of risk management, interoperability, architecture, process improvement, survivability, or

component-based design.

## For More Information

To learn more about the SEI's ASP, visit http://www.sei.cmu.edu/programs/acquisition-support. For more information about the SMC, visit http://www.losangeles.af.mil.



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# New CERT "Virtual Training Environment" Provides Online Information Security Education

NEWS AT SEI

Author

Eric Hayes

This article was originally published in News at SEI on: February 1, 2006

*Video, documentation, demonstrations, and lab exercises offer unique learning opportunities for information-security professionals.*

Today's networks are complex and globally connected. Educating those responsible for the security and survivability of these networks—and the computer systems they connect—is a major challenge. An even greater challenge is to provide this education in a way that is accessible and not tied to a certain place or time.

To meet these challenges, the CERT Program at the Carnegie Mellon Software Engineering Institute (SEI) has launched the CERT Virtual Training Environment (VTE). Available to anyone connected to the Internet, VTE employs technology to deliver security knowledge to information-security professionals.

CERT VTE provides quick access to training materials such as white papers, captured desktop screen demonstrations, recorded course lectures, and hands-on training labs for information technology professionals who cannot attend in-person training. VTE also provides trained personnel with a means to refresh and practice their knowledge and skills. Thus, network administrators and other IT security professionals can learn from experts with over 17 years of security knowledge and expertise without having to leave their desks.

## Different Content for Different Needs

VTE offers Knowledge in Depth for Defense in Depth (KD3) training material covering topics on information assurance, cyber forensics, and other IT-related topics in three different forms.

First, CERT VTE offers a free public-access Web "library" of content that includes documents, demonstrations of concept applications, and videotaped lectures. Users

who wish to learn the basics of computer security can visit the VTE library and select a subject they want to learn more about.

In the library, students can choose one or more of the following subject-matter groups: asset and risk management, information assurance policy and implementation, TCP/IP security, cryptography, host-system hardening, securing network infrastructure, firewalls and network security, intrusion detection, synchronization and logging, forensics and incident handling. The content for these 10 groups incorporates existing materials from CERT public course offerings, as well as other special material drawn from training and courses that are not offered to the public.

"The SEI's mission is to transition knowledge and technology," says Jim Wrubel, senior member of the technical staff and VTE team lead. "By providing free access to these documents, we are not only fulfilling that mission, but also raising the awareness that an informed and educated IT community will prevent security problems before they happen."

Next, VTE offers a paid subscription service, which contains lab environments that users can access, performing exercises that test the skills learned through use of the other VTE materials.

"These labs provide a unique training exercise—a chance to apply the knowledge described in the lectures, documents, and demos found in the public library. They provide a chance to try new skills in a risk-free environment," says Wrubel.

Finally, beginning in March 2006, information-security professionals will have the opportunity to complete full courses in this online environment. The online classroom mode will offer a guided path through course content, track and evaluate student progress, and give students the opportunity to interact with a course instructor via email and chat sessions. There will be a charge for each course taken, and students who complete these online courses will be eligible for continuing education units (CEUs), just like in-person SEI course attendees.

VTE will initially offer online versions of three courses that are now offered in live public and customer-site locations: Information Security for Technical Staff, Advanced Information Security for Technical Staff. More network-security and incident-handling courses will be added in the future. By the summer of 2006, special tracks will be available that will allow organizations to meet the training-compliance needs of directives such as DoD 8570 and the Federal Information Security Management Act (FISMA). VTE can also be used by organizations to host their own private content, to be delivered exclusively to their designated audiences.

(< 5 minute) survey.



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# Carnegie Mellon Software Engineering Institute Launches New PSP-Developer Certification

NEWS AT SEI

This article was originally published in News at SEI on: March 1, 2006

> *"PSP certification will assure potential employers that you are a professional who is capable of producing high-quality software for predictable costs and on committed schedules."*
> *—Watts Humphrey, SEI Fellow & National Medal of Technology Recipient*

The SEI officially launched its SEI-Certified PSP Developer Certification program in March at the 18th Annual Software Engineering Process Group (SEPG) conference in Nashville, Tenn. But, put away your pens, pencils, and papers because the exam, which will measure knowledge of the Personal Software Process (PSP) methodology, will soon be going online and going global.

The SEI has partnered with Kryterion, a secure online examination delivery and program management company, to enable interested candidates to take SEI certification exams at any of 250 locations worldwide. Kryterion will use the global network of Drake Authorized Testing Centers to enable candidates to take their exams at places and times that are convenient to them. Candidates will be able to register through the PSP-Developer Web site in early May.

"We're very excited to be offering this new service to professionals interested in affirming their commitment to disciplined process improvement," says Jeff Welch, SEI Certification Program manager. "As an organization with global partnerships, we saw the benefit of bringing the certification exams closer to the individual."

In today's workplace, managers need assurance that employees can meet the demand for timeliness, efficiency, and quality. Certification by the SEI—where the PSP was developed—provides an objective confirmation of PSP knowledge. The PSP Developer certification was developed in response to the growing number of software development professionals who found that incorporating the PSP methodology into their software development efforts consistently improved their ability to produce quality products on predictable schedules.

(< 5 minute) survey.

"The PSP Developer certification recognizes that accomplishment by software developers with a distinction that allows them to stand out among their peers," says James Over, senior member of the technical staff and TSP technical lead. "It also serves the software development community by providing a means to differentiate between individuals with the requisite knowledge of the PSP and those without this unique qualification."

The certification is based on the guiding principles of the PSP itself and developed in accordance with the testing guidelines set forth by the American Educational Research Association, the American Psychological Association, and the National Council on Measurement in Education. In addition, the SEI conducted alpha and beta testing to ensure that the high standards of the question writers and reviewers had been met. The pilot testing enabled the SEI to receive feedback, suggestions, and recommendations.

"Subject matter experts with diverse backgrounds were involved in each phase of the SEI-Certified PSP Developer exam," says Welch. "The experts were asked to review the questions for clarity, correctness, fairness, and appropriateness. All questions in the question bank underwent this same rigorous review process."

### Taking the Exam

Candidates sitting for the PSP Developer certification must successfully demonstrate understanding and comprehension of the competencies detailed in the *Personal Software Process (PSP) Body of Knowledge (BOK)*. The two-hour exam consists of 80 multiple-choice questions and covers the following competency areas described in the *PSP BOK*:

- foundational knowledge
- basic PSP concepts
- size measuring and estimating
- making and tracking project plans
- planning and tracking software quality
- software design
- process extensions

The *PSP BOK* provides a high-level comprehensive overview of the knowledge areas and competencies of PSP. It helps individual practitioners assess and improve their own skills, provides employers with an objective baseline for assessing the personal process skills and capabilities of their engineers and teams, and guides academic institutions that want to incorporate PSP into their software and other engineering courses or curricula.

Although not required for the PSP Developer certification, candidates are encouraged to take PSP public courses offered by the SEI or the SEI Partner Network.

### Improving Software Quality through PSP

Through the pioneering efforts of Watts S. Humphrey, National Medal of Technology laureate, the SEI has pioneered practices including PSP, the Capability Maturity Model Integration (CMMI), and Team Software Process (TSP) methodologies that are revolutionizing the way that organizations and people work and helping organizations efficiently create products that meet the needs of their customers.

Together, the PSP and TSP show individuals how to achieve goals and plan projects in detail with enough time to do the job right. SEI research confirms that with PSP and TSP, developers can produce superior products within budget and on schedule. Average product cost and schedule performance is typically within plus or minus 10 percent of planned commitments, and software quality is improved by 10 times.

### SEI Certifications and Certificates

In addition to the PSP Developer certification, the SEI offers CERT-Certified Computer Security Incident Handler certification and 12 certificate programs in the areas of software architecture, product lines, process management, and computer security. The

SEI is also working on future certifications in measurement and analysis and for PSP instructors and TSP coaches.



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# Initiative Advocates Building Security In from the Start

NEWS AT SEI

This article was originally published in News at SEI on: March 1, 2006

To improve efficiency, organizations today promote integration through networked systems. Yet when they do so, they increase the risk of intrusion and compromise if software is not secure. Integrating software and system security can mitigate the risk. The Build Security In (BSI) Software Assurance Initiative seeks to alter the way software is developed so that it is less vulnerable to attack and security is "built in" from the start.

Typical software development life-cycle models are not focused on creating secure systems and exhibit shortcomings when the goal is to develop systems with a high degree of assurance [Marmor-Squires 88]. If addressed at all, security is often relegated to a separate thread of project activity in which it is treated as an add-on property [Mead 01].

"Security considerations should not be treated separately from primary system-development tasks," says Nancy Mead, senior member of the technical staff at the SEI CERT Program and technical lead of the BSI initiative. "To develop systems with required functionality and performance that can also withstand failures and compromises, security should be integrated and treated the same as other system properties. Important requirements and design decisions and tradeoffs become more difficult when security is not integrated into the primary development life cycle."

Separate threads of activities are expensive and labor intensive, often resulting in duplicated effort in design and documentation. In addition, tools for supporting security engineering are often not integrated, and technologies that support security goals such as formal specification, architecture tradeoff methods, intrusion analysis, and security design patterns are not effectively applied into the development process. When security is treated separately, it becomes more difficult to adequately assess the risks and consequences of failure.

"For each life-cycle activity," says Mead, "security goals should be addressed and methods to ensure security should be incorporated."

One way of illustrating a life-cycle approach that incorporates security into each major

activity is shown in Figure 1. The emphasis is on artifacts, and the activities support production of the artifacts.

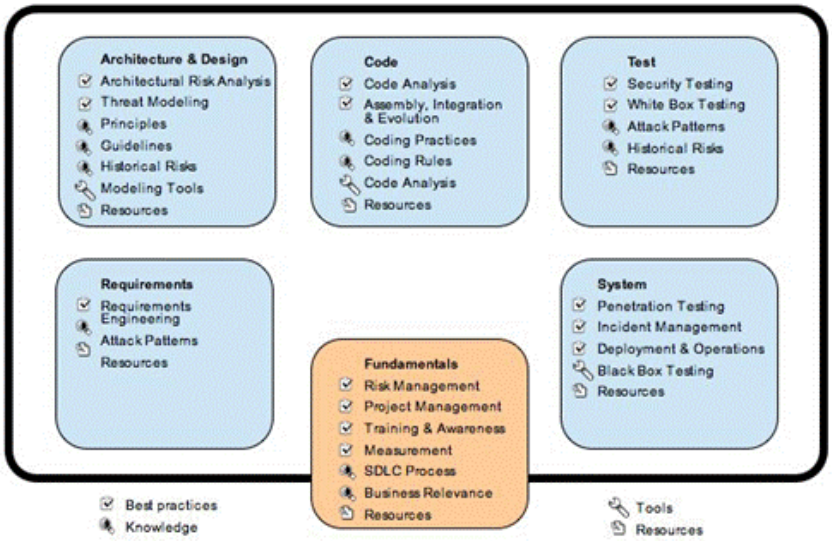



*Figure 1: Incorporating Security into Life-Cycle Activities*

## Building Security In

Many reported security incidents are the result of exploits against defects in the design or code of software, and many security efforts attempt retroactively to bolt on devices that make it more difficult for those defects to be exploited. But this approach does not address the root problem or threat.

BSI is a project of the National Cyber Security Division (NCSD) of the Department of Homeland Security (DHS). NCSD has sponsored development and collection of software-assurance and software-security information that will help software developers and architects create secure systems.

“We saw the need to create awareness of software security in a number of audiences,” says Joe Jarzombek, director for software assurance at NCSD. “New software developers, acquisition professionals, and managers, for example, sometimes don’t understand their roles in making software secure, and our initiative will eventually provide each of these groups and others with tools they can use to incorporate security considerations from the earliest stages of a project.”

As part of the initiative, BSI published an online content catalog, which is based on the principle that software security is fundamentally a software engineering problem and must be addressed in a systematic way throughout the software-development life cycle. The catalog contains or links to a broad range of information about best practices, tools, guidelines, rules, principles, and other knowledge to help organizations build secure and reliable software.

## BSI Content Catalog

The BSI catalog organizes material in categories of best practices, knowledge, and tools.

| Best Practices | Knowledge |
|---|---|
| architecture and analysis | attack patterns |
| assembly, integration and evaluation | business relevance |
| code analysis | coding practices |
| deployment and operations | coding rules |
| incident management | guidelines |
| measurement | historical risks |
| penetration testing | principles |

| | |
|---|---|
| project management<br>requirements engineering<br>risk management<br>security testing<br>threat modeling<br>training and awareness<br>white box testing | SDLC process<br>**Tools**<br>black box testing<br>modeling<br>source code analysis |
| | |

### Best Practices

A significant portion of the BSI site is devoted to best practices that can provide the biggest return considering current best thinking, available technology, and industry practice.

### Knowledge

Software defects with security ramifications—including implementation bugs and design flaws such as buffer overflows and inconsistent error handling—are likely to persist. The BSI team has identified recurring patterns of software defects that lead to vulnerabilities and has documented detailed instructions on how to produce software without these defects.

### Tools

The BSI site describes tools that can help eliminate defects before code is released. The site currently covers black-box testing and source-code analysis tools; a modeling-tools topic is slated for future development.

Users can approach the online content in several ways. For example, a software engineer might use the catalog to determine applicable security guidelines, while an architect might use it to determine how to design a Web-services application in a secure fashion, and a development team leader might use the information to justify software-assurance techniques to management by building a business case.

## Outreach and Further Development

To help ensure that this software assurance initiative is accepted and supported by software-development organizations, DHS NCSD involved participants from industry, academia, and government. A Software Technical Working Group (STWG) reviewed the catalog content and continues to develop future content for the site.

Outreach activity includes a series of NCSD workshops and working-group sessions at which participants can receive and share information about software-assurance resources. Feedback from users of the content catalog, both at workshops and online, will be used to further develop or modify the content.

The initiative also plans to continually add to and update content on the Web site. For example, Version 1.0of *Secure Software Assurance: A Guide to the Common Body of Knowledge* and *Security in the Software Lifecycle*, a developers guide, will be released soon and posted on the BSI Web site.

To learn more about the Build Security In initiative, see the Web site.

## References

[Marmor-Squires 88]
Marmor-Squires, A. B. and P. A. Rougeau. “Issues in Process Models and Integrated Environments for Trusted Systems Development,” 109-113. *Proceedings of the 11th National Computer Security Conference*. Fort George G. Meade, MD, Oct. 17-20, 1988. Washington, D.C.: United States Government Printing Office, 1988.

[Mead 01]
Mead, N. R., R.C. Linger, J. McHugh, and H.F. Lipson. “Managing Software Development for Survivable Systems.” *Annals of Software Engineering 2* (2001): 45-78.

Find Us Here

Share This Page

## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# Strategic Architecting

NEWS AT SEI

Author

Rick Kazman

This library item is related to the following area(s) of work:

Software Architecture

This article was originally published in News at SEI on: March 1, 2006

For more than a century, since the "discovery" of the 80-20 principle by the Italian economist Vilfredo Pareto, scholars and practitioners alike have preached the merits of applying this principle in business. The popular thesis, as applied to software development, is that 80% of property X involves only 20% of property Y. For example, "80% of the defects are found in 20% of the code," or "80% of the value is found in 20% of the features." The implication of such results seems clear: Focus your efforts on the high-payoff. But the reality is less simple.

Unfortunately 80-20 rules are almost always retrospective: You can only confidently apply them *after* the system has been developed and you have collected data from your experiences. As Butler Lampson has written, "it is normal for 80% of the time to be spent in 20% of the code, but *a priori* analysis or intuition usually can't find the 20% with any certainty" [Lampson 93]. Thus the *real* use of the 80-20 principle has been as a way to find out about all the opportunities you missed.

There have been several practical impediments to making strategic, up-front use of an 80-20 rule in software development:

1. How can you determine in advance where the 20% is?
2. What confidence is there that your currentproblem follows the same 80-20 pattern?
3. Does the expected benefit for the 80% outweigh the cost for focusing on the related 20% that (in theory) will deliver it?
4. What is the specific strategy (i.e. what problems to solve, using which techniques,

Related Links

**News**

SATURN Conference Announces Additional Keynote, Conference Scholarships

Distinguished Speakers, Strong Technical Program Set for SATURN 2014

See more related news »

**Training**

Big Data - Architectures and Technologies

Documenting Software Architectures - eLearning

See more related courses »

and in what order) that will cause you to focus on the 20% in question?

While 80-20 rules have indeed found wonderful tactical application—for example, IBM's classic OS 360 performance optimization—and have made reasonable strategic rules for business management, the above problems have made a *disciplined* approach to strategically managing software development elusive.

Clearly we would prefer to practice *strategic software engineering*, where we use methods that explicitly optimize expected value with respect to cost. Software architecture, in fact, provides a sound foundation from which to practice a strategic approach to software engineering. Consider the question of architecture flexibility; for example: How much flexibility should you build into a software architecture?

If you build in too much, your architecture, although highly flexible and easy to modify, will likely take a long time to create and suffer runtime performance problems. This is because flexibility is typically achieved by separating functions from each other with layers of abstraction, and these layers incur a runtime performance penalty. If you don't build in enough flexibility, your architecture will be rigid, with lots of hard-wired dependencies or, worse still, will be a "spaghetti" architecture in which everything is tangled together with everything else—then you will likely have a quick time to market, but you might suffer in the long run when you need to modify the architecture.

Port and Huang have suggested that while there is no "right" amount of architecture flexibility, you can choose the amount you want and engineer your system to achieve this amount [Port 03]. In effect, you are choosing to optimize the expected value from your architecture (choosing the 20%) rather than blindly building the architecture and hoping that the 20% that you focused on is the correct 20%. In this case you would choose the amount of architecture flexibility based on the percentage of features that you anticipated would need to be modified.

At the Software Engineering Institute, we have created a number of methods that promote strategic thinking for software architecture. This is appropriate, since an architecture is, by design, strategic: it is meant to be the carrier of the earliest, most fundamental, and, hence, hardest-to-change design decisions. The Quality Attribute Workshop (QAW) helps a group of stakeholders elicit and refine the most important quality attribute scenarios that will shape an architecture, thus helping with problem 1 above, "How can you determine in advance where the 20% is?" The 20% will be centered around your highest priority functions and quality attribute scenarios.

The Attribute-Driven Design method aids in attacking problem 4: "What is the specific strategy ... that will cause you to focus on the 20% in question?" The ADD method provides an explicit strategy for turning the high-priority quality attributes and functions into an explicit architecture that meets those requirements. And the Architecture Tradeoff Analysis Method (ATAM) and the Cost-Benefit Analysis Method (CBAM) help to address problems 2 and 3: "What confidence is there that your currentproblem follows the same 80-20 pattern?" and "Does the expected benefit for the 80% outweigh the cost for focusing on the related 20% that ... will deliver it?" ATAM checks the elicited scenarios with a broad group of system stakeholders and checks the mapping of scenarios onto an architecture to understand how well the proposed solution fits the problem. And CBAM helps model and predict the expected benefit that particular architectural strategies will achieve.

More recently we have been investigating the use of real options theory from economics to aid in understanding the costs and benefits of architectural flexibility. We will report on this work in an upcoming installment of *The Architect*.

## References

[Lampson 83]
Butler Lampson, "Hints for Computer System Design," *Operating Systems Review*, 15(5), Oct. 1983, 33-48.

[Port 03]
Dan Port, LiGuo Huang, "Strategic Architecture Flexibility," *Proceedings of 19th IEEE International Conference on Software Maintenance (ICSM'03)*, 2003.

## About the Author

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

The views expressed in this article 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.

# A Unified Process Improvement Approach for Multi-Model Improvement Environments

NEWS AT SEI

Authors

Urs Andelfinger

Andre Heijstek

Patrick Kirwan

This library item is related to the following area(s) of work:

System of Systems

This article was originally published in News at SEI on: April 1, 2006

Organizations throughout the world are turning to an ever-increasing set of international standards and models, such as Capability Maturity Model Integration (CMMI), Six Sigma, IDEAL, and others, in their effort to achieve competence in the processes used to manage their businesses, increase customer satisfaction, and achieve and maintain competitive advantage. The success of the Capability Maturity Model for Software (SW-CMM) and CMMI models, among others, has contributed to this trend.

The SEI's work with industry organizations reveals, however, that many organizations encounter difficulties in implementing process improvement successfully in multi-model improvement environments. These interactions have led the SEI's research team to develop an approach for successfully implementing process improvement in this challenging environment.

As organizations in multi-model improvement environments attempt to implement the standards and models they have chosen to meet business objectives and those that are mandated by regulators, they typically encounter a common set of barriers. Such barriers are frequently found in the automotive, financial, telecommunications, and service sectors. These barriers include the following:

- the proliferation of models that the organization must consider

Related Links

**Training**

Migrating Legacy Systems to SOA Environments - eLearning

Service-Oriented Architecture: Best Practices for Successful Adoption

See more related courses »

- the differences in structure and terminology across the standards and models
- difficulty in recognizing similarities among standards and models
- conflict between different improvement programs within the organization, as each attempts to champion and foster improvement based on its standard or model
- a lack of sustainability and institutionalization of achievements made by individual improvement programs
- a proliferation of the number and types of audits, assessments, and benchmarking that the organization must undergo while performing the functions needed to manage its business
- the perception of high risk associated with process improvement, especially in a multi-model environment

Because organizations are typically structured to implement individual model-based improvement initiatives independent of one another, the risk that a new improvement initiative will fail is high. In addition, improvement programs compete for the resources available in the organization. Because improvement programs and the improvements resulting from such programs are not coordinated, those charged with the day-to-day execution, maintenance, and management of the organization's processes begin to experience fatigue.

The challenge is to succeed in implementing process improvement despite the additional difficulties found in a multi-model improvement environment.

### The Unified Process Improvement Approach

To succeed, organizations must change the way they approach process improvement. Improvement initiatives in multi-model environments must be integrated across relevant standards and models, and this integration must continue. A first step of integration is to recognize that despite the different structures and terminologies and despite different levels of abstractions, the standards and models used in the organization share common element types. The challenge is to examine the models to identify the common elements and then to organize implementation of these standards and models in the *organization's* processes in relation to these common elements.

The SEI examined a series of standards and models and identified three common element types:

- good-practice elements
- improvement methods
- institutionalization elements

Every standard and model relevant to an organization will contain some or all of these element types. Once an organization starts to view the standards and models using element types, it has made a significant step toward a more effective approach to process improvement in a multi-model environment.

### Good-Practice Elements

Good-practice elements define what an organization must improve in any particular area. The specific practices of CMMI are an example of good-practice elements. Other models with such elements include ISO 9001, COBIT, and ITIL.

Having identified good-practice elements across the relevant standards and models, the organization must view these elements as requirements on the organization's process. This set of requirements then becomes the focus of change instead of the many individual models. Thus, the many good-practice elements across the relevant models become the relevant requirements to be satisfied by the organization's processes. Therefore, these requirements can be approached as such when planning improvement cycles and structuring the organization to support process improvement.

### Improvement Methods

Improvement methods are the model elements that help an organization effect its needed changes and facilitate its technology-transition process. These methods help

the organization master the change-management process, communicate the vision of the future organization, plan and execute the resulting improvement initiatives, and manage them over time. The improvement methods also provide guidance for forming organizational structures, roles, processes, and methods needed to effect change. Examples of improvement methods include Total Quality Management, IDEAL, Six Sigma, assessments methodologies, and audits.

Organizations should select the components of any of the improvement methods available that are useful to them or familiar to them and that are best suited to their needs and cultures. The most important change is that organizations apply a single, uniform approach across all improvement initiatives. This does not mean using, for example, only Six Sigma; rather, it means that the organization applies its own uniform improvement methodology across all improvement initiatives in the organization no matter what standard or model or combination it implements. In particular, the organization should avoid using different transition methods for different models. Using one transition method increases the organization's ability to communicate across models and individual improvement initiatives.

### Institutionalization Elements

While helping industry partners implement CMMI-based improvement in multi-model improvement environments, the SEI saw that the strong emphasis in CMMI on institutionalizing the improvements was missing or only weakly represented in many other models. To sustain and institutionalize improvement, it is critical to apply a unified, common set of goals and practices related to institutionalization across the good-practice elements and improvement methods of all the standards and models used by the organization. The Generic Goals and Generic Practices of CMMI offer an excellent basis for institutionalization and may easily be extended by describing elaborations for the good-practice elements and improvement methods of other standards and models. This unified approach to institutionalization also contributes to a common understanding of how to sustain hard-won improvements across all standards and models, helping management, process users, and process professionals within organizations communicate with each other more effectively.

### Managing Process Relationships in a Multi-Model Improvement Environment

If an organization revises its process-improvement efforts using this approach, it will establish a sound basis for implementing improvement in multi-model improvement environments. A major obstacle to success, however, remains: the complexity of the relationship among standards, models, and the organization's own processes. For this reason, the unified process improvement approach stresses the importance of placing the organization's own processes at the center of the improvement effort—because the standards and models can ultimately be implemented only in the organization's process.

The standards and models must be mapped to the subprocesses, procedures, and instructions of the organization's process. This mapping of individual standards and models to the organization's own process must take place at the level of abstraction embodied in the standards and models. This work should be supported by the use of metadata, an annotation mechanism, to reference individual parts of an organization's own process model to the corresponding parts of each relevant standard or reference model. This metadata-based mapping allows individuals in different roles in an organization to generate context-specific views of the organization's process in relation to the standard or model currently in focus. It also helps organizations prepare more effectively for the many audits, assessments, and benchmarking activities to which they are committed because they can generate the relevant view easily.

### Building the Experience Base

Organizations working in multi-model improvement environments may benefit from examining relevant standards and models in terms of good-practice elements, improvement methods, and institutionalization elements. By implementing process improvement in terms of these elements and linking the relevant standards and models to their own processes using metadata to allow the generation of context-specific views, organizations will have an effective approach to driving improvement in multi-model improvement environments.

The SEI continues to develop and refine the unified process improvement approach and to expand the experience base with its partners in industry.

## About the Authors

Patrick Kirwan is as a senior member of the technical staff at the Software Engineering Institute in Europe. He is currently working on the challenges associated with the effective transition of new technology to industry in multi-model improvement environments. His research interests also include the challenges associated with building large interoperable systems of systems as part of the Integration of Software-Intensive Systems (ISIS) initiative.

Urs Andelfinger is a visiting scientist at the Software Engineering Institute in Europe and a professor of software and business engineering at Darmstadt University of Applied Sciences, Germany. He is currently working on effective process improvement adoption approaches with a focus on the financial industry. He has 12 years' industry and consulting experience in the banking and automotive domains.

Hans Sassenburg is a part of the SEI's Software Engineering Process Management group and works as a visiting scientist for the Software Engineering Institute in Europe. In 2002 he started doctoral-level research studying the area of software-release decisions. This work was finished in January 2006.

André Heijstek has almost 20 years' experience with process improvement all over Europe. He led the corporate CMM program at Ericsson and consulted in finance, automotive, utilities, telecommunications, IT, and defense companies. Heijstek is an SEI authorized lead appraiser and trainer for CMMI. His current research interest is process adoption for sustained business benefit.



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# A Methodology to Support Software-Release Decisions

NEWS AT SEI

Author

Hans Sassenburg

This article was originally published in News at SEI on: April 1, 2006

Software is everywhere, and developing it has become a major worldwide industry. We find software embedded, for example, in watches, coffee makers, cars, televisions, airplanes, telephones, reservation systems, and medical equipment. Software not only pervades a multitude of products, but also is an important corporate asset, and demand is increasing. Yet software projects are characterized by schedule and budget overruns and the delivery of unreliable and difficult-to-maintain software products.

Despite an exponential increase in the demand for and dependence on software, many software manufacturers exhibit unpredictable behavior. It is sometimes difficult to determine, for example, a software product's release date, its features, the associated development costs, or the resulting product quality.

Uncertainty in the release date causes difficulties in planning product promotions, customer training, and maintenance support. Resource utilization across projects can become inefficient and difficult to manage when projects do not meet schedules. Customers have difficulties planning for the introduction of new software into their organizations when a scheduled release date is missed.

The exponential growth of the number of software products and releases suggests that end users will be exposed to more defects if the software industry is not able to reduce defect potentials and increase removal efficiencies at a similar rate. Competitive pressures related to feature content, time to market, and product quality will make product-release timing decisions both more important and more complex.

## A Strategic Software-Release Methodology

Software-release decisions often have strategic implications because of the high costs of reversing the decision. Prospective losses also may arise long after the release decision has been made; for example, in cases where liability leads to lawsuits. Existing decision

models for product release often do not account for market strategy. Ideally, release-decision criteria should align with corporate strategy.

A software-release decision can be seen from different perspectives:

- **Maximizing behavior**. A software-release decision is a tradeoff between early release to reap the benefits of an earlier market introduction and deferred release to enhance functionality or improve quality. If a software product is released too early, the software manufacturer incurs the post-release costs of later fixing failures. If a software product is released too late, the additional development cost and the opportunity cost of missing a market opportunity could be substantial. These two alternatives must be compared to determine which alternative maximizes economic value.
- **Optimizing behavior**. A release decision is affected by the difficulty of verifying the correct implementation of functional and non-functional requirements. How much testing is needed? Software manufacturers must find the optimal level of information because information has its price in cost and time. In practice, cost and time will constrain the ability to retrieve complete and reliable information, so this search for information should be considered as an economic activity. This leaves the software manufacturer with the problem of finding the optimal level of information where marginal value equals marginal costs and marginal yield is zero. This optimal level is difficult, if not impossible, to find.
- **Satisficing behavior**. Decision-making in the real world is often unstructured and normally involves various stakeholders who may have reasons to release a system or software product because of political or business pressures even though they know that it still contains defects. A study of spacecraft accidents, for example, reveals that, although system and software engineering were inadequate during development, management and organizational factors—including the diffusion of responsibility and authority, limited communication channels, and poor information flows—played a significant role [Leveson 04].
- **Decision Implementation**. A decision is considered successful if there is congruence between the expected outcome and the actual outcome. This definition sets requirements for decision implementation. In practice, there are many obstacles to the successful implementation of almost any decision, including
  - the reduced importance of a decision once it is made and implemented
  - the control of the outcome of a decision by stakeholders not involved in making it
  - the development of new situations and problems that command the attention of the decision-makers once the choice has been implemented

### Improving Strategic Software-Release Decisions

To investigate how to improve strategic software-release decision making, the SEI supported research into designing a release-decision methodology. This research reviewed the four perspectives detailed above from both a theoretical and an empirical point of view by studying practical examples. The results helped to frame a proposed release-decision methodology to address software-release decisions from different perspectives. The methodology, which consists of a defined set of practices, combines insights from economics, software management, and social psychology.

Studies in three different organizations validated the methodology. One participating organization is a leading global financial services company that provides financial services and products to retail and business markets. Services include insurance, pensions, occupational health and safety, asset management, investments, leasing, real estate, venture capital, and mortgage finance.

The research examined a project in this organization's IT department, which develops custom systems for internal and external use. The initial estimate for the schedule was 10 months and for the pre-release cash outflows, €15M. Budgets were reserved for the

technical infrastructure and post-release cash outflows for maintenance and exploitation. During the first months, the project encountered several setbacks: technical problems surfaced and the development budget turned out to be optimistic. Progress control was lacking, mainly through the absence of clearly defined milestones or quality gates. These problems increased, and in November 2001, the project was redefined. Both senior management and the marketing department exerted pressure on the product-development team to release the product as soon as possible. The team was, however, faced with an unstable product under test and had to use a veto several times to postpone a scheduled release date. When the product was released, uncertainty was high because many known problems were not resolved (although not considered critical), and the organization judged that continued testing would reveal more defects, including potentially critical ones, that could severely hamper the correct functioning and stability of the product.

After the product release, a special task force assumed responsibility for corrective-maintenance activities. This team needed more than a year to resolve the known and newly detected defects. Despite the original requirement to develop a maintainable product, the organization decided in 2004 to start a pre-study toward a new product to replace this product because corrective maintenance and functional enhancements proved difficult and costly. In other words, the early release of the product saved the organization additional testing costs, but the post-release maintenance cost turned out to be significantly higher than expected. A retrospective review of this project using the release-decision methodology enabled this organization to assess the project from a release decision point of view.

Figure 1 illustrates how the organization scored on the identified practices in the methodology. Lack of a product-development strategy (release definition) and lack of information as input to the decision-making process (release information) led to a poorly structured release-decision process without consensus among the stakeholders involved (release decision). Sufficient financial resources saved the organization in the short term by making it possible to patch the released software.

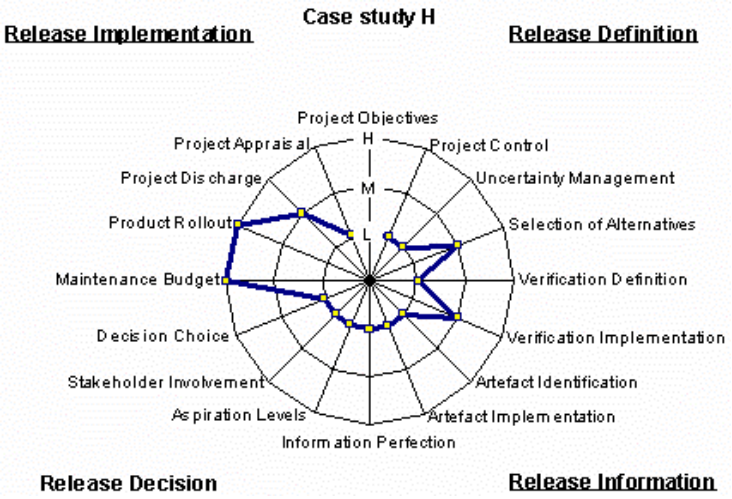



*Figure 1: Radar Presentation of Case Study Results*

These results from a practical setting indicate that this software-release decision methodology supports understanding, analysis, assessment, and improvement of the capability of software manufacturers in this problematic area. Research in software-manufacturing environments is under way to study the effects of applying the methodology at the start of projects to proactively aim for release-decision success.

## Reference

[Leveson 04]
Leveson, N.G. "The Role of Software in Spacecraft Accidents." *AIAA Journal of Spacecraft and Rockets, 41,* 4, 2004. Appendix: Release-Decision Methodology

## Appendix: Release-Decision Methodology

In the release-decision methodology framework, four areas in the software-release

decision-making process are distinguished, each addressing the process from a different perspective. A process area is defined as a cluster of related practices that, when performed collectively, achieve a set of goals considered important for establishing process capability in that area. Each process area consists of four relevant practices, describing *what* is to be accomplished but not *how*. Through this approach, the descriptions of practices provide the potential for interpretation and customization to the external market environment and to internal strategic and functional characteristics of a software-manufacturing organization.

Identified process areas are (see Figure 2):

1. *Release Definition*. Decision-making is mainly viewed from a quantitative perspective, assuming that information is nearly perfect—complete and reliable. It emphasizes the *maximizing behavior* approach with emphasis on mathematics, economics, and statistics. In software-release decisions, decision-making from a quantitative perspective is concerned with the definition and control of a product-development strategy—setting the managerial objectives with their priorities and ensuring that they are attainable. The availability of a product-development strategy enables the comparison and evaluation of different release alternatives, answering the question, "Which alternative maximizes economic value?"
2. *Release Information*. This process area is concerned with the search for alternatives during product development—for example, the identification and collection of information needed to compare and evaluate release alternatives. This search is derived from the formulated product-development strategy. Decision-making is also viewed from a quantitative perspective, but with the recognition that information is imperfect in the sense that not everything can be expressed in numbers and that information has its price in time and money. For this process, mathematics, economics, and statistics still play important roles, but the maximizing behavior approach is extended with an *optimizing behavior* approach: What is the optimal volume of information? Insufficient information increases uncertainty and hampers the decision-making process, whereas too much information is a waste of scarce resources. There is an optimum above which the cost for searching for more information exceeds the benefits.
3. *Release Decision*. Decision-making is viewed from a psychological, sociological, and socio-psychological perspective, addressing factors that influence individual and group behavior. It recognizes the imperfections of information and acknowledges that stakeholders involved in the choice may have different preferences with respect to the decision outcome. The challenge is to use a judgmental strategy to reach a decision that meets the formulated objectives and is agreeable to all stakeholders involved. The concept of optimizing behavior is extended with a *satisficing behavior* approach: Which outcome satisfies the needs of all stakeholders involved?
4. *Release Implementation*. Decision-making is viewed from an implementation perspective once a decision has been made and is implemented. This assumes that a successful decision requires follow-up and control. For software-release decisions, it is necessary to identify the factors that ensure congruence between the expected and the actual outcomes. The organization should evaluate the decision-making process and its outcome to promote organizational learning.

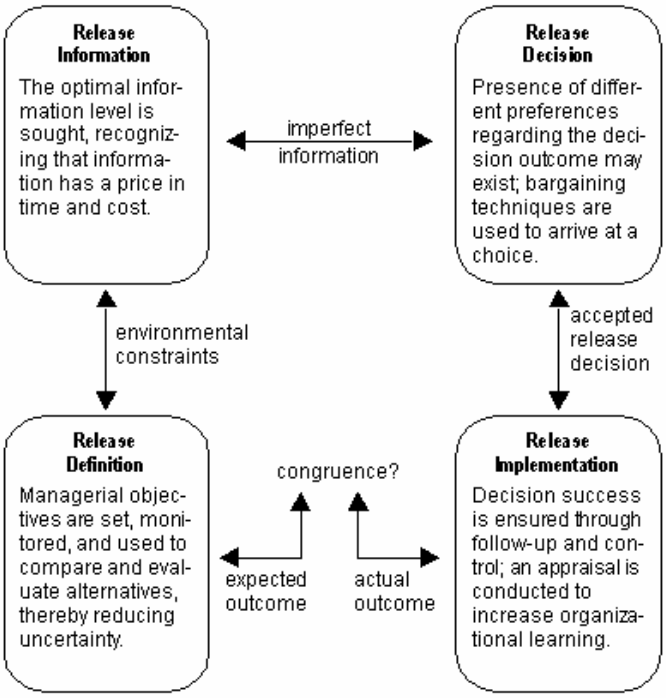



*Figure 2: Release-Decision Methodology*

## About the Author

Hans Sassenburg works as a visiting scientist for the Software Engineering Institute in Europe. He is a part of the Software Engineering Process Management group. In 2002 he started doctoral-level research studying the area of software release decisions. This work was finished in January 2006.

# FAQs Part 4: Product Lines in the Context of Acquisition

NEWS AT SEI

Author

Paul C. Clements

This library item is related to the following area(s) of work:

Software Product Lines

This article was originally published in News at SEI on: April 1, 2006

*"We're a U.S. government contractor, and our government customer wants to build a product line by buying core assets from us and then staging an open competition to build products using those core assets. How can we possibly compete on the product-building side when all these new companies enter the fray and claim to be able to build products less expensively because they didn't have to pay for the core assets?"*

First, other contractors can't necessarily build derivative products less expensively. At best, they can probably build them as cheaply, but even that is questionable. In practice, the core-asset contractor actually may have an advantage on follow-on product development by virtue of its domain knowledge, expertise, and intimate knowledge of the core assets. Other contractors may have a significant and potentially steep learning curve with regard to understanding the functionality and technical characteristics of the core assets, augmenting them with product-specific assets, and integrating and testing them to create a finished product. In fact, it is usually those other contractors that are concerned about the unfair advantage the asset contractor has because, under acquisition reform, contractor experience often plays a large part in the technical evaluation criteria.

All of this indicates the need to create a business strategy that effectively balances the interests of the core-asset contractor and the acquisition organization (and other prospective product-development contractors). The core-asset contractor obviously has an interest in protecting the competitive edge it has in the marketplace. Its core assets are an example of its domain knowledge and software expertise and may be a major

Related Links

**Training**

Software Product Lines - eLearning

Software Product Lines

See more related courses »

factor in maintaining its competitive advantage. Accordingly, there are several options that the core-asset contractor, in conjunction with the acquisition organization, can pursue. They include

- selling the core assets outright to the government (with negotiated government-usage rights) while maintaining all commercial rights to the core assets
- licensing government usage of the core assets on a per product (or per asset) licensing basis (e.g., the government pays a license fee for each product that is developed using the core assets)
- negotiating a follow-on contract by which the asset contractor manages, sustains, and evolves the core assets and provides technical support for product development through the auspices of the acquisition organization
- opening up product development to competition through a leader-follower type of contract with the core-asset contractor taking the leader role
- allowing the core-asset contractor to compete for product development (the same way as any other contractor) or, alternatively, prohibiting the core-asset contractor from competing on product development
- avoiding reliance on any one contractor by letting an umbrella contract (competitively) to multiple contractors that allows them to compete or collaborate on the follow-on development of individual products using the core-asset base; this would allow a greater number of contractors to participate in product development and could allow the core-asset contractor to compete

While there is no one best option, the non-exhaustive list above demonstrates that there are many viable contracting options that can be pursued depending on the goals and acquisition strategy of the government agency and the business strategy of the core-asset contractor.

Finally, if the core assets make it so straightforward to produce products based on them, then the core-asset contractor is in a perfect position to rapidly bring entirely new products to the commercial marketplace, thus taking advantage of the new production capability created courtesy of the government.

*"We're a government contractor who wants to bid on a competition to build products based on core assets developed by another contractor. How can we possibly compete given the intimate knowledge possessed by the asset-development contractor?"*

The fact that both this question and the preceding question are frequently asked shows that product lines do not particularly provide an advantage in either direction. In fact, this question is no different than asking "How can I hope to compete because I didn't build the commercial off-the-shelf (COTS) software or the government-furnished equipment (GFE) that I'm required to use under this contract?" Contractors compete successfully under those conditions all the time. So the premise behind the question—that a contractor can compete successfully only if it is tasked with building all of the software—is just not valid. A pragmatic answer is that your company is no worse off than if the product line strategy had not been pursued.

*"I'm a government contractor, and the government wants me to supply reusable core components for other organizations to use in building a product line. Why should I open myself up to the legal liability? I'll be liable if they use my components incorrectly!"*

Liability issues can be tricky. They are negotiated between the acquisition organization

and the development contractor(s) and often involve legal counsel. However, in the scenario described in the question, the government contractor would not be directly liable for how another (third-party) government contractor uses its product, and especially if it uses it incorrectly. A contractor who develops components for a government agency is liable to the government—not to another government contractor—to the extent stipulated in the expressed warranties that are part of the contract. There may also be implied warranties of fitness for use for the purpose for which the government will use the items. Contracting officers have to consult with counsel before asserting any claim for breach of an implied warranty.

The important thing to remember is that liability issues are not unique to product lines. The government often contracts for a piece of equipment from one manufacturer and then provides it as government-furnished equipment (GFE) to another contractor to integrate into its final product. In such cases, the liability for the GFE items rests with the government, although the government may have recourse to go back to the original development contractor to have a defect corrected, depending on the contractual warranties that were negotiated and agreed to by both parties.

For commercial items (i.e., nondevelopmental items), liability considerations (expressed and implied) are described in Part 12 (Acquisition of Commercial Items) of the Federal Acquisition Regulations [FAR 97]. General and specific liability considerations that apply to both commercial items and developmental items are described in Part 46 (Quality Assurance) of the FAR.

Specific solicitation provisions and contract clauses on warranties and liabilities that may apply are described in Part 52 of the FAR and Part 252 of the of the Defense Federal Acquisition Regulation Supplement [DFARS 98]. They include the following sections:

- FAR: 52.246-18; Warranty of Supplies of a Complex Nature
- FAR: 52.266-19; Warranty of Systems and Equipment under Performance Specifications or Design Criteria
- FAR: 52.246-23; Limitation of Liability
- FAR: 52.246-24; Limitation of Liability, High Value Items
- DFARS: 252.246.7001; Data Warranty

The bottom line is that during the contract-solicitation period, any contractor considering bidding on a government contract can formally request that the contracting officer define in writing the extent and limitation of contractor liabilities.

## References

[FAR 97]
Federal Acquisition Regulation, FAC 97, 1997. The FAC 97-11 version of the Federal Acquisition Circular (FAC) includes amendments effective as of May 3, 1999.

[DFARS 98]
Defense Federal Acquisition Regulations Supplement, 1998 Edition, 1998.

## About the Author

Paul Clements is a senior member of the technical staff at the SEI, where he has worked for 10 years leading or co-leading projects in software product line engineering and software architecture design, documentation, and analysis. Clements is the co-author of three practitioner-oriented books about software architecture: *Software Architecture in Practice* (1998; second edition, 2003), *Evaluating Software Architectures: Methods and Case Studies* (2001), and *Documenting Software Architectures: View and Beyond* (2002). He also co-wrote *Software Product Lines: Practices and Patterns* (2001), and was co-author and editor of *Constructing Superior Software* (1999). In addition, Clements has also written dozens of papers in software engineering reflecting his long-standing interest in the design and specification of

challenging software systems. He received a BS in mathematical sciences in 1977 and an MS in computer science in 1980, both from the University of North Carolina at Chapel Hill. He received a PhD in computer sciences from the University of Texas at Austin in 1994.

The views expressed in this article 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.

# CMMI: The New Architecture

NEWS AT SEI

Author

Mike Phillips

This library item is related to the following area(s) of work:

Process Improvement

CMMI

This article was originally published in News at SEI on: May 1, 2006

As planning began for CMMI Version 1.2, the CMMI Product Team recognized that future expansion of CMMI coverage needed in domains closely related to development, such as services and acquisition, might require adjustments to the architectural approach used for CMMI V1.0 and V1.1. Following is the information you'll need to understand these changes.

## CMMI Version 1.2 Now 'CMMI for Development'

Because of the creation of CMMI models for services and acquisition, what I have been calling CMMI Version 1.2 in my recent columns has been renamed CMMI for Development (CMMI-DEV). Besides this name change, the modifications made to this model are quite small. A single model for development, scheduled for publication this year, provides amplifications for the systems engineering, software engineering, and hardware engineering disciplines. IPPD coverage will be included in the model as an addition. (In V1.1, IPPD was a discipline.) The CMMI Development Team also simplified and consolidated the coverage of both IPPD and Supplier Sourcing in a way that eliminates the need for three process areas:

- Organizational Environment for Integration (OEI)
- Integrated Teaming (IT)
- Integrated Supplier Management (ISM)

With the first two releases of CMMI, it was important to recognize the distinctions of the specific disciplines and to recognize the heritage of the improvement models for

Please note that current and future CMMI research, training, and information has been transitioned to the CMMI Institute, a wholly-owned subsidiary of Carnegie Mellon University.

Related Links

**News**

SEI to Co-Sponsor 26th Annual IEEE Software Technology Conference

SEI Cosponsors Agile for Government Summit

**Training**

See more related courses »

**Events**

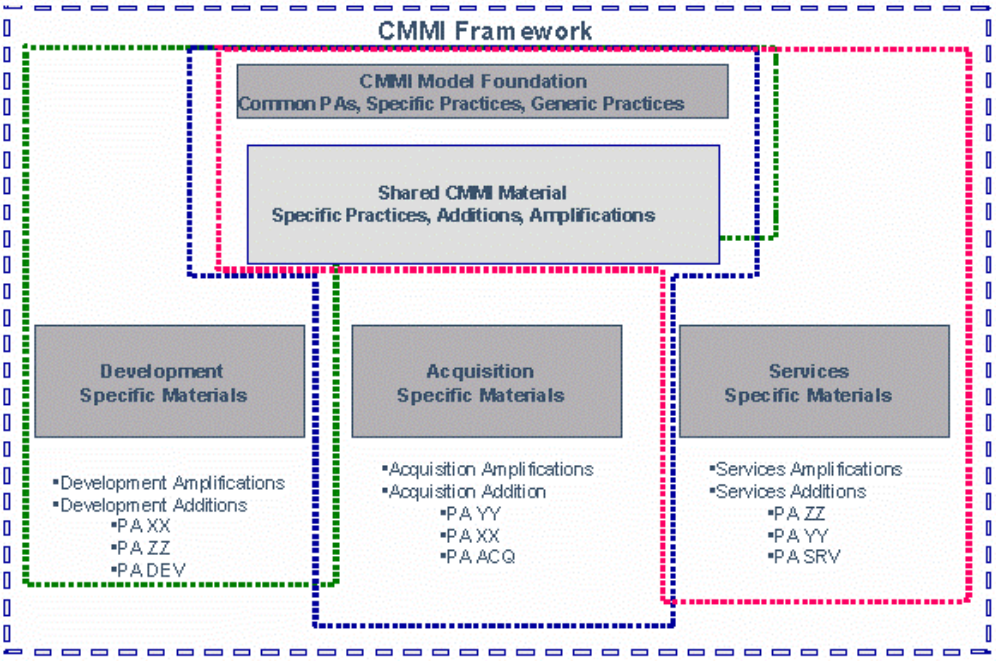
Team Software Process (TSP) Symposium 2014

Nov 3 - 6

each of the disciplines. However, these distinctions have now become less important, and the unifying engineering-development processes have demonstrated synergies that go beyond the original source models. The team also received requests from users and the CMMI Steering Group to simplify the material. The increasing number of possible variations (and therefore printed models) to address the various combinations of existing engineering disciplines made movement in that direction undesirable. Consequently, we added amplifications for hardware-engineering examples but chose not to call out another model variation in the model name. Nor are multiple models available for users to choose from. Instead there is one integrated model containing the best development practices.

### Changes to CMMI Beyond CMMI for Development

As we began to consider future coverage of organizational process improvement, we sought to maintain the greatest possible commonality among all the models created from the CMMI common framework of best practices. The figure below depicts this commonality along with needed specificity. This provides a way to avoid having any CMMI model grow too large for effective use.




Based on the initial efforts to maximize commonality among CMMI models, 16 of the 22 process areas of CMMI-DEV comprise the process improvement core for the three areas of interest currently being pursued: development, acquisition, and services. These 16 process areas (in alphabetical order) are

- Causal Analysis and Resolution (CAR)
- Configuration Management (CM)
- Decision Analysis and Resolution (DAR)
- Integrated Project Management (IPM)
- Measurement and Analysis (M&A)
- Organizational Innovation and Deployment (OID)
- Organizational Process Definition (OPD)
- Organizational Process Focus (OPF)
- Organizational Process Performance (OPP)
- Organizational Training (OT)
- Project Monitoring and Control (PMC)
- Project Planning (PP)
- Process and Product Quality Assurance (PPQA)

- Quantitative Project Management (QPM)
- Requirements Management (REQM)
- Risk Management (RSKM)

We use the word *constellation* to refer to a set of model components, training materials, and appraisal documents in the CMMI Framework that covers an area of interest, such as development, services, or acquisition. Each constellation includes the 16 common process areas above, with additions unique to the area of interest, or shared across some, but not all, of the constellations.

We recognized that even with common process areas, we needed to allow some flexibility. None is allowed, however, for the "required" (goals) or "expected" (practices) levels of the 16 core process areas. Additions to these process areas will be allowed, just as the IPPD addition is allowed and encouraged in the Development constellation. In the informative material, we allow a little more flexibility so that typical work products can be added or substituted to fit a process area in each constellation. The only other substitutions or deletions allowed within the 16 core process areas will be at the level of the informative material judged specific to development. This occurs in the current model in subpractices, where development-specific explanations are often found. These statements may be tailored to the needs of the new constellation and include informative paragraphs below sub-practices and generic practice elaborations.

More tailoring is permitted to describe activities captured primarily in the Engineering process areas of CMMI-DEV. While some of the constellations may share components with the Engineering process areas in CMMI-DEV, the shared material may be arranged and grouped differently to meet the needs of the constellation's user base. If these adjustments change the process area in any significant way, the process area will be given a different name to avoid confusion in use, training, or appraisals. If a particular PA can be shared by two constellations, these PAs will be designed to capture that commonality as well. (For example, the existing VER or VAL might be usable in one of the future models but not in others, so it would be shared across two constellations.)

Teams that are setting out to develop their own internal expansions of the CMMI Framework can request a more detailed version of the CMMI architecture to assist in their development efforts—but all users should understand the broad approach we have put in place to create an architecture that allows flexibility of coverage while preserving the key ingredients of commonality to assure adequate stability across the set of emerging constellations.

## About the Author

As the director of special projects at the Software Engineering Institute, Mike Phillips leads 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, Ohio. In addition to his bachelor's degree in astronautical engineering from the U.S. Air Force Academy, Phillips has master's 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.

Contact Us
info@sei.cmu.edu
412-268-5800

# Meet Ricky & Stick

NEWS AT SEI

This article was originally published in News at SEI on: May 1, 2006

If your work involves software, and if you've ever laughed out loud over a Dilbert cartoon, then just wait until you meet Ricky and Stick, the cartoon duo who recently made their way onto the SEI Web site.

Ricky and Stick are best friends who live on the same street and play together a lot. Their escapades around school and the neighborhood, which frequently get them in trouble, are captured in *The Adventures of Ricky & Stick: Fables in Software Acquisition*, a book of cartoons with themes that relate to software, acquisition, or government programs in general.

*Ricky & Stick* is the brainchild of David Carney, senior member of the technical staff at the SEI. To turn the idea into a book, he collaborated with David Biber, graphic designer and illustrator with the SEI's CERT Program. Together, Carney and Biber—through Ricky, Stick, and their friends—give a send-up to the profession that will leave software developers, engineers, and acquisition professionals chuckling and groaning with familiarity.

*"No need to try it out—it'll work just fine."*
*T*esting and Modeling

Carney has taken an informal and irreverent look at software before; his *Little Red Book* a short, tongue-in-cheek essay about the dangers and challenges of using commercial off-the-shelf (COTS) software in government systems is a takeoff on Mao Zedong's famous Little Red Book. Hoping only that the essay would amuse a few people, Carney was thoroughly unprepared for how deeply it resonated in the Department of Defense community. Carney's *Little Red Book* has undergone numerous reprints, and he still receives email from readers who appreciate its spurious Chinese aphorisms.

Asked recently to offer something short and to the point that would prepare beginning program managers for being stuck between demanding users, angry program executive officers, and frustrated software engineers, Carney chose to use a humorous approach once again. And *Ricky & Stick* was born.

“I decided that these little stories should be fables,” says Carney, “each of which includes a moral relevant to software, to acquisition, or to government programs. Possibly the most important point—and yet another similarity to the *Little Red Book*—is that these fables are based on real-world experiences: all of the situations in this book are inspired by programs that I know.

“A common thread among these fables,” he says, “is the need is to keep sight of a few simple, fundamental realities, which are all too easy to dismiss as mere common sense. But when the world seems to be coming apart at the seams, it is amazing how easy it is to let common sense fly out the window. A besieged program manager—even an experienced one—can sometimes make decisions that appear reasonable under pressure, but in retrospect seem harebrained.”

Cartoons in the book are organized into six chapters:

- Testing and Modeling
- Estimation and Metrics
- Requirements
- Integration and Interoperability
- Deployment
- Business Processes

Each cartoon in the book has accompanying text that leads to a bottom-line statement or moral, which Carney encapsulates in the epilogue. Examples include “Using the ‘seems OK to me’ rule is usually a disaster,” and “Counting the right things is better than counting the wrong things.”

The book's sponsor was the U.S. Air Force Acquisition Center of Excellence, which, during the book's development, was led by acting director Col. Ralph DiCicco, now retired. The center's current director, Brig. Gen. Janet C. Wolfenbarger, wrote in the introduction, "This book isn't an official guide to best practice, and it certainly isn't a textbook. But in a kind of off-beat way, it's an entertaining yet insightful look at some of the things that can really happen in software acquisition; each fable is based on true examples where our acquisition system has broken down."

"In a way," says Carney, "using Ricky and Stick, I can say more and say it more memorably in six panels than I can in a lengthy technical report."

Installments from the book recently began to appear on the Acquisition Support Program's pages on the SEI Web site; one chapter will appear each month until the complete text is online.

Carney says also that he and Biber are looking for readers to send more situations, stories, or scenarios. "If you can help us concoct a few more of these little fables, we'll definitely find a way to use them somehow."

# Quality Attributes and Service-Oriented Architectures

NEWS AT SEI

Authors

Len Bass

Paulo Merson

William O'Brien

This library item is related to the following area(s) of work:

Software Architecture

This article was originally published in News at SEI on: May 1, 2006

Building systems to satisfy current and future mission/business goals is critical to the success of a business or organization. Software architecture is the bridge between mission/business goals and a software-intensive system. Quality-attribute requirements drive software architecture design [SEI 06]. Choosing and designing an architecture for such systems—one that satisfies the functional as well as the nonfunctional or quality-attribute requirements (reliability, security, maintainability, etc.)—are vital to the success of those systems. Recently, the use of service-oriented architecture (SOA) has gained widespread popularity as the approach for various types of systems. Although some work has been done on analyzing how particular quality attributes such as security and interoperability are handled within an SOA, we undertook a more thorough examination of the relationship between SOA and quality attributes.

The choice to use an SOA approach in the development of an architecture depends on several factors including the architecture's ultimate ability to meet functional and quality-attribute requirements. Usually, an architecture must satisfy many quality-attribute requirements in order to achieve the organization's business goals. In almost all cases, tradeoffs have to be made among these requirements. In some cases, satisfying these requirements may be easier using an SOA; in others, it may be more difficult. The following summarizes the results of our recent investigation into how an

Related Links

**News**

SATURN Conference Announces Additional Keynote, Conference Scholarships

Distinguished Speakers, Strong Technical Program Set for SATURN 2014

See more related news »

**Training**

Big Data - Architectures and Technologies

Documenting Software Architectures - eLearning

See more related courses »

SOA supports quality attributes [O'Brien 05].

- **Interoperability**: Through the use of the underlying standards, an SOA provides good interoperability support, allowing services and applications built in different languages and deployed on different platforms to interact. However, semantic interoperability is not fully addressed. The standards to support semantic interoperability are immature and are still being developed.
- **Reliability**: Undependable message transmission is possible in many areas, but using Web Services (WS) Reliability and WS Reliable Messaging standards can help. As with any element in an architecture, service reliability is still an issue.
- **Availability**: It is up to the service users to negotiate a service-level agreement (SLA) that can be used to set an agreed-upon level of availability and to include penalties for noncompliance with the agreement. Also, if a service user builds into its applications contingencies, such as for handling exceptions when an invoked service is not available (e.g., dynamically locating another source for the needed service), availability should not decrease and could actually be improved in comparison to other architectural approaches.
- **Usability**: If the services within the application support human interactions with the system and there are performance problems with the services, usability may decrease. It is up to the service users and providers to build support for usability into their systems.
- **Security**: The need for encryption, authentication, and trust within an SOA approach requires detailed attention within the architecture. Many standards are being developed to support security, but most are still immature. If these issues are not dealt with appropriately within the SOA, security could be compromised.
- **Performance**: An SOA approach can have a negative effect on the performance of an application due to network delays, the overhead of looking up services in a directory, and the overhead caused by intermediaries that handle communication. Both the service user and service provider must design and evaluate their portions of an SOA-based design to ensure that performance requirements are met.
- **Scalability**: There are ways to deal with an increase in the number of service users and the increased need to support more requests for services. However, these solutions require detailed analysis by the service providers to make sure that other quality attributes are not negatively affected.
- **Extensibility**: Extending an SOA by adding new services or incorporating additional capabilities into existing services is supported within an SOA. However, the interface or formal contract for services must be extendable when necessary, without causing a major impact on the service users' applications.
- **Adaptability**: The use of an SOA approach should have a positive effect on adaptability as long as the adaptations are managed properly. However, the management of this quality attribute is left up to the service users and providers, and no standards exists to support it.
- **Testability**: Testability can be negatively affected by the use of an SOA because of the complexity of the testing services that are distributed across a network. Those services might be provided by external organizations that do not have access to the source code and therefore have no way of knowing the code coverage of a series of test cases. Also, if the SOA uses runtime discovery of services, it may be impossible to identify which services are used until a system executes. It is up to the service users and providers to test the services, and very little support is currently provided for the end-to-end testing of an SOA.
- **Auditability**: Auditability can be negatively affected if the right capabilities for end-to-end auditing are not built into the system by the service users and are not incorporated into the services by the service providers.
- **Operability and deployability**: Operation and deployment of services and systems that use other services must be managed carefully to help meet the quality of service (QoS) specified in SLAs. The interactions and tradeoffs among this and other quality attributes must be monitored and managed.

- **Modifiability**: Because the underlying logic is hidden behind the service interface, modifiability of services or an application that uses them is directly supported using an SOA approach. However, that interface must be designed to mitigate changes, since the effect of changes on external users of the service might be difficult to identify if the service is externally available.

Choosing and designing the right architecture for a system—one that satisfies its functional and nonfunctional (quality-attribute) requirements—are vital to the success of that system. It is particularly important to examine how using an SOA supports the quality attributes most critical to that system. When an organization designs and implements a system using an SOA approach, it must make various tradeoffs among the quality-attribute requirements that will affect whether the organization will fully meet its business goals. SOA is still an emerging technology, many of the issues between SOA and quality attributes have not been thoroughly researched, and many of the standards posing as remedies are immature.

If external services, or even those outside the control of the development department, are used, an SLA must be established between the various parties to guarantee QoS for a set of essential services. Building a system that relies on third parties without the necessary agreements in place can make it difficult for the system to meet its quality-attribute requirements. Failing to meet them would adversely affect an organization's ability to meet its business goals and, in turn, its success.

We are continuing to investigate the relationship between quality attributes and SOAs, so be sure to look for more information on this topic in our future columns.

## References

[SEI 06]
Software Engineering Institute. Software Architecture Technology Initiative (2006).

[O'Brien 05]
O'Brien, L.; Merson, P.; & Bass, L. *Quality Attributes and Service-Oriented Architectures*(CMU/SEI-2005-TN-014). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

## About the Authors

Liam O'Brien works for the Irish Software Engineering Research Centre, Limerick, Ireland. Until recently, he was a senior member of technical staff at the Software Engineering Institute, where he worked in the areas of service-oriented architectures, modernization of legacy systems, architecture reconstruction, and architecture-centric software life cycles. His latest publications include several reports published by Carnegie Mellon and papers at various international conferences on these subjects. He has more than 16 years of experience in software engineering practice and research.

Len Bass is a senior member of the technical staff at the Software Engineering Institute who participates in the High Dependability Computing Program. He has written two award-winning books on software architecture as well as several other books and numerous papers in a wide variety of areas of computer science and software engineering. He is currently working on techniques for the methodical design of software architectures and to understand how to support usability through software architecture. He has been involved in the development of numerous production or research software systems ranging from operating systems to database management systems to automotive systems.

Paulo Merson is a member of technical staff at the Software Engineering Institute, where he works in the Software Architecture Technology and the Predictable Assembly from Certifiable Components Initiatives. He is currently investigating service-oriented architectures, aspect-oriented software development, model-driven development, and software architecture representation. Merson has more than 15 years of experience in software development. Prior to joining the SEI, he was a J2EE consultant and worked on the implementation of several enterprise applications.

Mellon University. This article is intended to stimulate further discussion about this topic.



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# Large-Scale Work–Part V: Building Team Ownership

NEWS AT SEI

Author

Watts S. Humphrey

This library item is related to the following area(s) of work:

Process Improvement

This article was originally published in News at SEI on: May 1, 2006

This is the fifth column in a series on large-scale development work. The first column briefly summarized the issues of large-scale development work. It then addressed the problems of organization size and how, as organizations grow and mature, they necessarily become less efficient. The second column dealt with project issues and particularly with the management decision process. Often, the most serious problems in large projects concern the way decisions are made. In the third column, I focused on the people and teams that make up the project staff and the issues involved in maximizing their performance. The fourth column addressed the problems of applying democratic management principles in large organizations. As demonstrated in politics, autocratic management is not an effective way to run a large and complex modern country. Autocratic management has also proved troublesome for large and complex organizations and projects. In this column, I discuss why so many organizations appear to be autocratic and why project ownership is important.

## Why Are Organizations Autocratic?

Most organizations aren't really autocratic, they just look that way. The problem is not that the managers are truly autocrats, but that their people act as if they were. For example, during my years at IBM, I ran a development laboratory of several thousand people. I made a practice of walking through the lab several times a week and chatting with folks. I soon got to know the crew in the model shop pretty well. Two of the development engineers had starting coming in late and hanging their coats on the model shop coat rack so nobody would know they were late. At the end of the day, they left right on time. The entire shop was in an uproar.

Related Links

**News**

SEI to Co-Sponsor 26th Annual IEEE Software Technology Conference

SEI Cosponsors Agile for Government Summit

**Training**

See more related courses »

**Events**

Team Software Process (TSP) Symposium 2014

Nov 3 - 6

(< 5 minute) survey.

One day, the shop elected a spokesman, and he came to see me. He was obviously very upset, and he told me the entire story. In my staff meeting later that day, I told this story to the senior managers. I concluded by saying that this was unacceptable behavior, and it had to stop. Thereafter, I was known as an autocrat who insisted on punctuality. That, of course, was not the case, but there was nothing I could do about it.

While a great deal could be said about the subject of image and how to develop or improve one, only three points are pertinent to this discussion. First, managers are people, and they occasionally get annoyed and take action without thinking about how it might affect their image. Second, technical managers have so much to do that they simply can't take to time to explain everything that must be done. They just have to do what they think is right and assume that their people will trust them and follow their guidance. The third part of the autocratic image problem concerns positive actions, and this is where many managers fail. Managers must provide the guidance and support their teams need to develop a feeling of ownership among all of their members.

### How Do You Develop Owners?

In my lab, the machinists clearly felt and acted like owners. They had all been with the company for many years and had large parts of their savings invested in IBM stock. They had an ethic of getting to work early and working until they finished that day's jobs, even if it meant staying late. But at least two of the development engineers did not have this feeling. So how do you overcome this kind of problem?

I recently heard a story that described what a now-retired vice president did to handle this problem. In this large military contractor's development laboratory, dates were often set by the customer, by some other team, or by executive fiat. The development teams couldn't do much but struggle with impossibly tight schedules. This VP understood that such schedules would often slip, but he expected his teams to do their best to meet these dates anyway.

When team leaders came to him with proposals to slip their dates, however, he would consider the proposals, but only with two caveats. First, these team leaders had to provide an understandable rationale for the slip. Second, they had to have a recovery plan that they were certain to meet. In effect, he said: "The first slip's on me, but the second one's on you." This made the team leaders owners of their recovery plans and schedules. As far as these team leaders were concerned, this executive was no autocrat.

So making team leaders into owners would seem to be relatively straightforward: management just has to let them set their own dates. While this may help, it doesn't really address the problem of making the developers into owners. To do that, management must have the entire team, not just the team leader, set the dates.

### Negotiating with Management

To make developers feel like owners, management must really make them owners. This can be done only by having all the members of each development team participate in making its own plan. Then they all must participate in negotiating that plan with management. To do this, however, the teams must know how to make reasonably accurate plans, and they must be guided through this planning process by someone who can coach them on how to make accurate plans and on how to negotiate plans with management.

The obvious next question is: "How much evidence does it take to convince management to agree to the team's plan instead of the one that management originally wanted?" Managers have commitments too, and sometimes these commitments are very hard to change. So teams can expect their managers to be hard to convince. On the other hand, managers are frequently seasoned professionals, and many of them have also been developers. They have generally been around long enough to understand the common problems that development teams have with schedules.

So the key is for teams to do more than just complain that this is a tough schedule and that they don't see how to meet it. They have to make very detailed and thorough

plans. And, in doing so, they must work with the entire team and do their very best to produce a plan that meets management's date. Then, if they can't, they will know why and be able to explain the reasons to management.

While this may sound too easy, it turns out to be surprisingly effective. I have worked with hundreds of teams while at IBM and the SEI, and I know of no case where a team has followed this advice and not succeeded in convincing management that it had a better and more realistic plan for doing the job. Even when the team's plan took twice as long as management wanted or took twice as many people as originally planned, management has still bought the team's recommendation or a negotiated variant. The results of this strategy have also turned out to be good business: when teams produce their own plans, their typical schedule errors drop from 50% or more to less than 10% [Davis 03].

### Self-Directed Teams

While it can take a lot of work to make a really detailed and accurate plan, it is not really that hard to do. In fact, the Team Software Process (TSP) provides an entire process framework and a set of management and team-member courses that have proved successful at doing just this [Humphrey 02, 05, 06a, 06b]. TSP also addresses more than just the schedule issues that most often lead to autocratic behavior. It shows teams how to own and manage all aspects of their work. With TSP, teams select their own strategies, define their own processes, and make their own plans. They then manage their work to these plans and keep management posted on their progress.

For large-scale work, however, the problem is not just getting a team or two to make plans and to believe that they own these plans, it is building a feeling of ownership across an entire large-scale program. While the first part of doing this must be making plans and establishing ownership at the team level, that is only one step. How do you then extend this individual team-based ownership strategy to cover an entire large-scale program? This is the scale-up problem that I will address in the next column.

### Acknowledgments

First, I want to thank Phil Gould for his e-mail describing the story of the VP of a large defense contractor. It is a marvelous example of how executives, by being sensitive to the ownership issue, can make an enormous difference in how their teams feel about their work and their executives. Also, thanks to Bob Schaefer for his e-mail about convincing management to agree to new plans.

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 David Carrington, Harry Levinson, and Bob Schaefer.

### In Closing, an Invitation to Readers

In this particular series of columns, I discuss some of the development issues related to large-scale projects. Since this is an enormous subject, I cannot hope to be comprehensive, but I do want to address the issues that you feel most strongly about. So, if there are aspects of this subject that you feel are particularly important and would like covered, please drop me a note with your comments, questions, or suggestions. Better yet, include a war story or brief anecdote that illustrates your ideas. I will read your notes and consider them when planning future columns.

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu

### References

[Davis 03]
Davis, Noopur, & Mullaney, Julia. *Team Software Process (TSP) in Practice* (CMU/SEI-2003-TR-014). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

[Humphrey 02]
Humphrey, Watts S. *Winning With Software: An Executive Strategy*. Boston, MA:

Addison-Wesley, 2002.

[Humphrey 05]
Humphrey, Watts S. *PSP: A Self-Improvement Process for Software Engineers*. Boston, MA: Addison-Wesley, 2005.

[Humphrey 06a]
Humphrey, Watts S. *TSP: Leading a Development Team*. Boston, MA: Addison-Wesley, 2006.

[Humphrey 06b]
Humphrey, Watts S. *TSP: Coaching Development Teams*. Boston, MA: Addison-Wesley, 2006A.

## About the Author

Watts S. Humphrey founded the Software Process Program at the SEI. He is a fellow of the institute and is a research scientist on its staff. From 1959 to 1986, he was associated with IBM Corporation, where he was director of programming quality and process. His publications include many technical papers and several books. His most recent books are *Introduction to the Team Software Process* (2000) and *Winning With Software: An Executive Strategy* (2002). He holds five U.S. patents. He is a member of the Association for Computing Machinery, a fellow of the Institute for Electrical and Electronics Engineers, and a past member of the Malcolm Baldrige National Quality Award Board of Examiners. He holds a BS in physics from the University of Chicago, an MS in physics from the Illinois Institute of Technology, and an MBA from the University of Chicago.

The views expressed in this article 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.

# Ultra-Large-Scale (ULS) Systems Roundtable

NEWS AT SEI

Author

Bill Pollak

This library item is related to the following area(s) of work:

Ultra-Large-Scale Systems

This article was originally published in News at SEI on: June 1, 2006

In Issue 4 of 2005, *news@sei* we presented a feature article about a study that the SEI was conducting on behalf of the Office of the Assistant Secretary of the Army (Acquisition, Logistics, & Technology). The study brought together experts from within and outside the field of software engineering and from a variety of institutions and organizations to consider the challenges of ultra-large-scale (ULS) systems: given the issues with today's software engineering, how can we build the systems of the future that are likely to have billions of lines of code? Although a billion lines of code was the initial challenge, increased code size brings with it increased scale in many dimensions, posing challenges that strain current software foundations.

*Ultra-Large-Scale Systems: The Software Challenge of the Future* (ISBN 0-9786956-0-7) is the product of this 12-month study. The report, available on the Web, details a broad, multi-disciplinary research agenda for developing the ultra-large-scale systems of the future. The principal team of authors who wrote the report consists of Linda Northrop (study lead), Peter Feiler, John Goodenough, Rick Linger, Tom Longstaff, Rick Kazman, Mark Klein, and Kurt Wallnau from the SEI, along with Richard P. Gabriel, Sun Microsystems, Inc.; Douglas Schmidt, Vanderbilt University; and Kevin Sullivan, University of Virginia.

"The DoD has a goal of information dominance," says Northrop, director of the Product Line Systems Program at the SEI. "Achieving this goal depends on the availability of increasingly complex systems characterized by thousands of platforms, sensors, decision nodes, weapons, and users, connected through heterogeneous wired and wireless networks. These systems will be ULS systems. Although they will comprise far more than just software," says Northrop, "it is software that

fundamentally will make possible the achievement of the DoD's goal.

"Yet software is the least well understood and the most problematic element of our largest systems today. Our current understanding of software and our software-development practices will not meet the demands of the future. To make significant progress in the size and complexity of systems that can be built and deployed successfully, we require a culture shift. In this report, we identify the kinds of research that will effect such a culture shift. The United States needs a program that will fund this software research required to sustain ongoing transformations in national defense and global interdependence. The report provides the starting point for the path ahead."

In this article, five members of the author team discuss the report and its significance. Participants are

- Richard P. Gabriel, distinguished engineer, Sun Microsystems, Inc.
- John B. Goodenough, SEI fellow and senior member of the technical staff, SEI
- Thomas A. Longstaff, deputy director for technology, CERT Program, SEI
- Douglas Schmidt, associate chair of computer science and engineering, Vanderbilt University
- Kevin Sullivan, associate professor and Virginia Engineering Foundation Faculty Fellow in Computer Science, University of Virginia

***Pollak: Please describe the problems that the team considered and why these problems are important or significant.***

**Longstaff**: The problem we addressed is the need to improve the science of scaling up our ability to develop ULS systems. Scale in itself is a difficult concept to define. Our initial investigation indicated that more work was needed by a broader group of people to think this problem through and define what science is needed. What eventually came out are the seven research areas that are described in the report. (*Note: These are briefly described in the executive summary.*)

**Schmidt**: The basic problem we have today in the DoD is that systems are increasingly getting integrated out of many parts provided by many different suppliers—the common term for this is "systems of systems." This problem has been challenging for the past five years or so and is likely to get harder and harder in the future. Examples of systems that already have this problem are Future Combat Systems (FCS, Army), Global Information Grid (GIG, Air Force), and FORCENet (which is largely Navy driven). There is also an effort to wire all the services together, which is even harder.

The challenge that we were given is, what do we do when we have even more ambitious requirements in the future. Even FCS consists of no more than 30-35 million lines of code. We were given the charter to understand what would be the software-technology issues for systems that were a billion lines of code or larger. Lines of code per se are not so important; the idea is that these systems are much larger than systems we have to deal with today, when even today's systems are stretching what we can do from a technology and management standpoint. Our report is an effort to look ahead to what might happen in the next five years when the systems we are dealing with will get larger by an order of magnitude or two.

**Goodenough**: We're talking about a change in the way of thinking about how systems are developed. We try to build current large-scale systems today in the same way that we did smaller systems. The ULS study focused on the fact that techniques that were appropriate for smaller scale systems are not appropriate for larger systems—we need a new way of thinking about things. It's like being out hiking in the desert and bringing along some water, but not knowing how much water you need, because you are just thinking of it as being a hot day. But the low humidity in the desert means that you have new conditions to deal with, so you have to take a different approach or you won't be successful.

The point we're making is not simply that ULS systems are different, it's that the assumptions that we usually make when we create systems don't apply. The mental

model of how we build systems today is that, if we work hard enough, we can create a system that is close enough to perfection that it's good. But in ULS systems, we have to account for the idea that the systems are so large that failures are normal, so we have to plan ahead for dealing with the failures that will occur. With a ULS system, we'll never get it close to perfection, because when we get it to where it's working, we'll find out that it needs to do something different, and it will have to evolve in another direction. Or it will have certain failure characteristics that we'll have to cope with in different ways.

We don't think of systems today in which the evolution of the system is guided by the people who are actually using it. That's what happens in the Internet, but DoD systems are not procured in the way that the Internet was developed. For ULS systems, though, we need to have more of an Internet-development philosophy. Different units will find different ways of using the system and will somehow define their own capabilities, which will fall within the general flow of the ULS system. We talk today about "building" systems. In the world of ULS systems, we'll talk about "evolving" systems. We'll never start with a new system, we'll start with an existing system that moves on in another direction. This is not our current mental model. When we start talking about evolving, the nature of the problems we need to solve and the appropriate solution techniques change. That's what this study is all about: how we need to change what we think the problems are and how we need to change our solution approaches, because the old solution approaches are applicable to the old problems but not to the new problems. And we don't know what the new solution approaches are. When we decide that we can't muddle through any longer, that requires a new set of software engineering techniques that only a few groups of people here and there in the industry are thinking about. We need to stimulate and nourish these groups of people so that these ways of dealing with systems become more vibrant and produce results that can be applied on a broader scale.

**Gabriel**: The problem we addressed was how to build things at much larger scale. We build medium-to-large systems today through planning and trying to define and understand requirements. For ULS systems, it's obvious that we can't do that. So the challenge is, how do you get people who don't know each other to collaborate to put something together that's coherent, and does that imply that some parts of the system are so large that they can't be designed by people but must be designed by other software? This is important because, as we learn more about how to build software and have more capabilities we need to build in, we'll be making these systems whether we want to or not. What we do now is the Dorothy technique: clicking our heels together and hoping for the best. If you look at systems that are built today, they don't exactly do what people wanted; once people see and experience the system's capabilities, they say, "Oh, that's not what I meant." Or "Gee, I wish it would do this other thing." So we end up iterating. Software companies today are always doing some sort of iterative, incremental evolution of requirements, design, and implementation. The industry does that, and military contractors do that.

**Sullivan**: Computer science has enabled the development of truly remarkable kinds of systems, from music downloads to global information search to weather prediction to business management on a global scale. At same time, the revolution in computing has only begun. We've seen the revolution in desktop computing and networking of desktop machines, but we're now looking at a new world of computing systems that are much more deeply embedded into the fabric of our lives. Our experience to date with the kinds of systems we've built until now has shown us that the primary impediment to the realization of these new systems is our inadequate understanding of how to construct the software components of these systems so that they do what they're supposed to do in a way that is secure and dependable. But many decades of research in software engineering and computer science haven't yet resolved some of the fundamental difficulties that we face in building, validating, and verifying complex pieces of software.

We believe that the most important impediment to building next-generation systems is the software. We observed in the report that we have taken a traditional engineering viewpoint on software for a number of decades. In considering the question of what are the big breakthroughs that are needed, we decided that it might be necessary to go beyond the traditional engineering perspective—from centralized systems to highly

decentralized systems. We see some of this in the Internet today, with emerging classes of systems. So we looked at alternative metaphors for the construction of these systems —not just an engineering metaphor anymore, but an industry-structure metaphor or an economic metaphor or a complex-adaptive-systems metaphor or a biologically-inspired-systems metaphor. We're also moving from a metaphor of control to a metaphor of enablement, from a Tayloresque notion of control of engineering processes to a decentralized notion of regulation and inducement. We're thinking about producing the kinds of systems that are needed through a much more exploratory and emergent kind of process than has been common in the past.

***Pollak: What approach did you use to collaborate in investigating the challenges of ULS systems?***

**Longstaff**: We took a unique approach, which was not to pull together a group of like-minded computer scientists, but to bring together people with a variety of backgrounds and disciplines. So for example, we brought in people who understand the behavioral side of the use of computer systems, the history of the security of systems, the formal development and structuring of software algorithms, systems of systems and DoD work in this area, fundamental language issues in developing software, and the culture behind the development of computer science projects and systems.

The risk in bringing this diverse group together was that we wouldn't have a common basis for communication. But the group did an excellent job of defining the challenges and looking at the implications of the challenges in each of their independent areas. By keeping a relatively neutral focus in the writing team, we were able to create a consistent report written by a few individuals while still retaining the broad perspective of all the people who were in the original meeting.

**Schmidt**: What was different in this group was its multidisciplinary nature. We had people who provided an economics and management perspective as well as people with expertise in other parts of computing such as human-computer interaction, quality issues and statistics, programming languages, system modularity, design, and software engineering. There was an attempt from the beginning to be interdisciplinary and to include both people from traditional software engineering as well as people with a different perspective.

Typically with these kinds of studies, people come up with solutions that are purely technology driven; recommending, for example, a better network, operating system, or distributed system, middleware, or language. We have these kinds of recommendations in our conclusions and certainly don't ignore them, but the results are much broader. They include management-policy issues, human-computer interactions, the human dimension of computing, how to get stakeholders and users involved, and the economic dimensions. And these issues are not only identified in their own right, but they are woven through parts of the more conventional technology dimensions.

**Goodenough**: The assembly included people who are not traditionally invited into software engineering events. It was a diversity reflecting the idea that we need new perspectives, and if you need new perspectives, you can't go to the same people you always go to. This was part of the original charge to SEI—to go outside the usual boundaries and to think innovatively. It was Linda's idea to identify people who were outside the box but who at least knew what the box was so they could contribute in a reasonable way. We got lots of good ideas from such people in the initial workshop, and then the writing team helped to pull things together. We needed people who were able to reduce innovative ideas into understandable and believable prose. We had some incredible writers, and we threw away some incredible amounts of good prose in order to get a report that was succinct.

**Gabriel**: I was very happy after the first meeting. I knew right then that this could be a significant event in software. I was surprised at how well run the first meeting was—it was very well planned and facilitated. The SEI team didn't have preconceptions or try to impose expectations about what the ideas would be. The set of people who participated in the first meeting was good, and the subset of people on the writing team was outstanding.

**Sullivan**: The first important thing that happened was that Linda found a group of people who tend to think beyond the current state of the art—people who were not primarily incremental thinkers. The multidisciplinary nature of the team was important too. We had people with expertise in ethnographics, reliability, software architecture, financial economics—a variety of different perspectives—as well as people within the computer science field who are sympathetic for the need to reach out and exploit, import, adapt, and leverage theories and thinking from other disciplines.

Once we got those people together, the next step was to the extent possible to explore the range of issues, possibilities, and dimensions in which we might formulate a new kind of research agenda. The challenge for the writing team was then to reconcile, integrate, and refine, reduce, and focus down the key intellectual contributions made by the broader group and bring it together into a coherent presentation. In doing that, some additional insights were reached on how to put an overarching structure on the story.

***Pollak: What key insights about ULS systems emerged from the study?***

**Longstaff**: The most groundbreaking insight is the prominence of human issues with regard to ULS systems. This captures up-front attention in the report and glues together what is said later about the underlying computer science. Normally, you tend to shunt those issues off until after you develop the underlying technology. A key insight to the whole report is that ULS systems can't be successful without thinking of the human aspects not only of the computer system but also of the surrounding infrastructure of that system—how people will interact with the development of the system, with specification and design, with evolution. That's different from other reports that you might read.

**Schmidt**: A key insight is the need to be able to engage various stakeholders up and down different levels of the echelon—to get, for example, the people in the government policy-making realm to work with the people involved with the actual production of the software, those doing the R&D, and those doing the actual deployment and operation of the system. Also key is the ability to come up with a research agenda that spans different levels of the chain of command and different sectors of the industry. We characterize this in the report as a "socio-technical ecosystem," and this represents a novel insight about how to organize the research agenda.

Also an important result was the mapping of the DoD mission and capabilities called out in the *Quadrennial Defense Review Report* (February 2006) with research breakthroughs, needs, and approaches in the research tracks. In the report, we provide mappings and a way forward that would be relevant both to policy makers in the DoD as well as to technologists—that combination is rare. It's a multidimensional roadmap that enables different stakeholders and different people who need work done to take different paths and to use this report for their own purposes.

**Goodenough**: The key insight for me is the idea of thinking of ULS systems as ecosystems in which the behavior of participants is governed by rules, similar to the way zoning laws in a city help to shape the way the city grows. We were able to consider what are the equivalent in the ULS system domain of zoning laws and codes of behavior, which make life in a city acceptable. With ULS systems, we're going to talk about metrics of effectiveness that are more like the gross national product—not something that is an aggregate of a lot of individual test-point probes, but a sampling of what's going on in the system that gives insight into whether the overall system is tending toward viability or degradation. People who are running Google or the AOL mail system today use these kinds of probes to evaluate the health of their systems. The kinds of problems they are solving are perhaps more focused than the kinds of problems that we want the ULS systems to solve, but these techniques are hints of how we will need to deal with the ULS systems of the future. So we're not talking about radically inventing something that hasn't existed somewhere in some shape before, we're talking about focusing attention on those solutions that already exist that can be evolved into more powerful solutions that can be generalized to work at the ULS system scale.

**Gabriel**: The key insights for me are: identification of different viewpoints for looking

at these systems and how to operate them; the various metaphors that we used to understand ULS systems, such as the city metaphor and the ecosystem metaphor; the realization that the development process has to be more like a complex adaptive system than like a great brain trust that designs the system; and the realization that other disciplines can teach us things that will help us build and understand these systems and understand the organizations and the people that put them together, in order to be more effective.

**Sullivan**: The key insights correspond to the research areas that we identified in the report: human interaction; computational emergence; design; computational engineering; adaptive system infrastructure; adaptable and predictable system quality; and policy, acquisition, and management. These are the crucial areas in which we need much better understanding. I also think it is absolutely true that we need to consider people and organizations and economics, and that whole social aspect of computing systems more thoroughly. The challenges of ULS systems are difficult to handle entirely within the computer science field, and this insight dictates new kinds of collaborations between computer scientists and social scientists.

***Pollak: After the release of the report, what do you hope will happen in the future?***

**Longstaff**: I hope that senior leadership in the DoD will see what is possible with these ULS systems and will understand that in order to really achieve what we want, significant effort will have to be put into developing a different approach to assembling large-scale systems. I hope leadership will see the vision of the report and be captured by its possibilities, and that the implementation of any science agenda will be aligned with that vision.

**Schmidt**: Strategically, I hope that we can use this report as a way to help revitalize and create incentives for more technical work in these areas, as the starting point for getting things moving. From a more tactical point of view, the report makes it possible to identify the most promising or interesting research to investigate if it's possible to make progress in these areas and to help continue to flesh out the research agenda defined in the report. The research areas that we identified need further prototyping and experimentation so that we can have confidence that proposed solutions will be technically sound.

**Goodenough**: My hope is that this study will stimulate a whole different focus of thinking about software engineering problems. Our challenge of assumptions in the report will begin to enter other peoples' thinking. What happens, for example, if we don't know exactly what a system is supposed to do, but we still need to validate the system and ensure that it's safe? How do we reason that it's sufficiently safe or functional when we don't know as much about the system as we usually expect to know? We may seem to be asking a lot of questions that seem almost impossible or silly, but by asking them we begin to find new approaches that begin to address problems. My hope is that the solution that emerges from this work has some of the characteristics of a new paradigm of software engineering.

**Gabriel**: I hope that researchers in universities and industrial labs will take this seriously and that funding will come through to focus on it. A realistic hope is that it will cause enough of a stir that research in the software world moves enough in this direction that we'll be able to make some progress. People are already researching these ideas, but these people are today considered on the fringe. I hope that such people will now be considered slightly more toward the mainstream, more legitimate.

**Sullivan**: One of the most important statements in the report is that software is both the key enabler and the key impediment to progress. There simply is no way that industry is going to solve these fundamental problems in basic science and engineering on its own given its short-term and relatively low-technical-risk approach. It's also unlikely, in my judgment, that the software engineering research community by itself will be able to make the progress do what's needed, especially under the present conditions of woefully inadequate research funding and with its relatively stable perspective on how to proceed. We need to something really big, really innovative, and well informed. There really is a crying need for a visionary and substantial new

emphasis on research in fundamental issues in software. I would like the DoD and other agencies to get that message loud and clear. My hope would be that the right people at the executive level in government and industry get this message and understand that if they do something about it, it will make a world of difference in their own futures.



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# The Hottest Issues in Software Product Lines:

NEWS AT SEI

Author

John McGregor

This library item is related to the following area(s) of work:

Software Product Lines

This article was originally published in News at SEI on: June 1, 2006

The SEI is hosting the 10th International Software Product Lines Conference (SPLC 2006) in Baltimore, Md., from August 21 to 24 at the Baltimore Marriott Waterfront. SPLC is more than a forum for research results: it provides a venue for the product line community to share discoveries, experiences, and questions. The research and applied software product line communities come together at SPLC to discuss problems, explore solutions, and establish collaborations. SPLC brings together the three product line constituencies—organizational managers, technical managers, and software engineers—to exchange ideas and interact.

Highlights of the conference are

- keynote talks by Carliss Baldwin of the Harvard Business School and Gregor Kiczales of the University of British Columbia
- speakers ranging from business-unit vice presidents to software engineers
- interactive panels, including one on product derivation and one on testing, where detailed model problems will be used to ensure in-depth discussion
- numerous opportunities for networking, including social events and several lively birds-of-a-feather sessions

The software product line strategy continues to bring about order-of-magnitude improvements to organizations in cost, quality, and time to market. With the maturing of the field, the following areas of concerns come into focus:

- production planning

Related Links

**Training**

Software Product Lines - eLearning

Software Product Lines

See more related courses »

- economic issues
- variability management

The SPLC is facilitating the interaction of product line professionals in these areas.

### Production Planning

As a consultant to many large software development organizations, I have witnessed many spectacular failures. None was more dramatic than the efforts of one company that attempted to design a platform on which it planned to build several of its products. Multiple business units cooperated to fund and manage the platform development. By some measures, this effort was a technical success: The company produced an innovative architecture that promised to see its products into a fruitful new generation. Ultimately, however, the effort was a business failure. No single group or team within the company had control over both the platform and the products built from it. Individual product managers had incompatible objectives, while the schedule for incremental delivery of the platform satisfied none of the product teams. The platform team elicited requirements from the product teams, but no group explicitly considered how the product teams would produce products from the platform. The result? The platform was built, but because the organization had not planned for product production, no products were ever constructed using it.

As this failed effort indicates, production planning should occur before the core assets, much less the products, are produced. The planning should begin with development of a production strategy, which provides overarching guidance to the production-planning activity. This guidance should then be provided to the core-asset as well as product developers. The production strategy provides the basis for engineering the production method and the detailed processes, models, and technologies used to produce products. These two assets—the production plan and the production strategy—are essential inputs to the plan, which details how individual products are built [Chastek 02a, Chastek 02b].

SPLC 2006 will offer insights into product production and production planning.

- In his tutorial titled “Generative Software Development,” Krzysztof Czarnecki will describe one path to the automatic production of products. His approach uses feature-based modeling to specify a product.
- John Hunt will present his work on “Organizing the Asset Base for Product Derivation.” He will show the results of investigations into three techniques for organizing asset bases for optimal use in producing products.
- Kwanwoo Lee and his colleagues will present their work on “A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering.”

### Product Line Economics

An essential step before adopting the product line strategy is to determine whether the product line approach is appropriate given the business and technical context of the organization. The rate at which products in a domain are changing, the size of products in the domain, and the degree of similarity among the products are some of the factors that affect the viability of the product line approach in an organization. The business environment, such as whether the company produces software for internal use or sale or under contract to government agencies such as the Department of Defense (DoD), also affects the factors that should be considered.

The product line community has developed several models that incorporate these factors into computations to determine the costs and benefits of the product line strategy in a specific context. For example, the Software Engineering Institute’s (SEI's) Structured Intuitive Model of Product Line Economics (SIMPLE) is the result of a collaboration of several researchers and organizations [Böckle 03, Clements 05]. SIMPLE uses a core set of cost functions that guide the modeler in identifying and collecting the appropriate information for estimating the costs associated with the product line effort.

At SPLC 2005, a panel on economic models presented several approaches, including

SIMPLE. At SPLC 2006, this topic will continue to be explored:

- Carliss Baldwin's keynote talk at SPLC will explore how design architectures "affect economic incentives and patterns of competition over new products."
- Dharmalingam Ganesan, Dirk Muthig, and Kentaro Yoshimura will present their work on "Predicting Return-on-Investment for Product Line Generations."
- At the SEI reception on Wednesday evening, the latest version of the SIMPLE Web site will be introduced.

### Variability Management

An organization that wants to effectively exploit the commonality among products (even as few as two of them) must be aware of, and be able to manage, the variation among them. Variation among products is integral to conducting activities that range from the initial scoping of the set of products that constitutes the product line to the design, implementation, and testing of those products.

Many problems related to variability management remain unresolved:

- No one notation has gained widespread acceptance for representing variation in product specification or design. Several feature-modeling notations are currently in use. The most widely used software-design notation—the Unified Modeling Language (UML)—does not have explicit facilities for representing variability.
- The task of physically managing all the specifications and implementations of product variants challenges existing configuration management (CM) practices. Many CM tools are unable to map a single asset onto multiple products.
- Variability arises at virtually every level of product development. For this reason, techniques are needed to establish traceability from the earliest identification of product variability through product implementation and testing.
- There are no widely accepted tactics for mapping the identified need for a variant to the appropriate mechanism for implementing that variation.

At SPLC 2006, several presenters will continue to pursue variability management:

- Klaus Pohl, Frank van der Linden, and Andreas Metzger will present a tutorial titled "Software Product Line Variability Management."
- Paul Clements and Dirk Muthig will host a research workshop on "Managing Variability for Software Product Lines: Working with Variability Mechanisms."
- Gregor Kiczales' keynote on "Radical Research in Modularity: Aspect-Oriented Programming and Other Ideas" will look forward to new tools for implementing variation management.

What other enriching opportunities will happen at SPLC? I have no idea—but some are sure to. Often the most useful exchanges are those that happen in response to a question during a session or during an exchange between panelists, in a conversation at the conference reception, or during the breaks between sessions. The SPLC organizers are trying to provide an environment in which these encounters are encouraged and facilitated. Visit www.splc.net for the latest information on what we've planned and how you can register.

### References

[Böckle 03]
Böckle, G.; Clements, P.; McGregor, J.; & Schmid, K. "A Cost Model for Software Product Lines," 310-316. Proceedings of the Program Family Engineering (5) Conference. Siena, Italy, November 4-6, 2003. Berlin: Springer, 2003.

[Chastek 02a]
Chastek, G. & McGregor, J. D. *Guidelines for Developing a Product Line Production Plan* (CMU/SEI-2002-TR-006, ADA406687). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Chastek 02b]
Chastek, G.; Donohoe, P.; & McGregor, J.D. Product Line Production Planning for the

Home Integration System Example (CMU/SEI-2002-TN-029, ADA408820). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Chastek 03]
Chastek, Gary & Donohoe, Patrick. *Product Line Analysis for Practitioners* (CMU/SEI-2003-TR-008, ADA421616). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

[Clements 02]
Clements, P. & Northrop, L. *Software Product Lines: Practices and Patterns*. Reading, MA: Addison-Wesley, 2002.

[Clements 05]
Clements, Paul; McGregor, John D.; & Cohen, Sholom. *The Structured Intuitive Model for Product Line Economics (SIMPLE)* (CMU/SEI-2005-TR-003, ADA441881). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

[SEI 05]
Software Engineering Institute. *A Framework for Software Product Line Practice* (2005).

## About the Author

John D. McGregor is a visiting scientist at the SEI and an associate professor of computer science at Clemson University. He is the general conference chair for SPLC 2006 and a partner in Luminary Software, LLC—a software product line consulting firm. McGregor is coauthor of *A Practical Guide to Testing Object-Oriented Software*, (Addison-Wesley, 2001) and *Object-Oriented Software Development: Engineering Software for Reuse* (International Thomson Computer Press, 1992). He writes a bimonthly column on strategic software engineering for the *Journal of Object Technology*. He earned BS, MS, and PhD degrees from Vanderbilt University.

The views expressed in this article 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.

# Five Maxims about Emergent Behavior in Systems of Systems

NEWS AT SEI

Author

John Morley

This library item is related to the following area(s) of work:

System of Systems

This article was originally published in News at SEI on: June 1, 2006

A highway traffic tunnel is open on both ends. There are no traffic signals to control speed or obstacles to prevent vehicles from traveling at the posted speed limit. In fact, typically there are signs that encourage drivers to maintain their speed as they approach and enter a tunnel.

Yet, traffic often slows just before the entry to a tunnel. Why? What force causes this behavior? View the highway-tunnel-automobile combination as a system of systems, and you can call the slowing of traffic an emergent property.

David Fisher, a senior member of the technical staff in the Integration of Software-Intensive Systems (ISIS) initiative at the Carnegie Mellon Software Engineering Institute, calls emergent properties "cumulative effects" of the actions and interactions among "constituents" of complex systems, a constituent being any "automated, mechanized, or human" participant.

## Engineers of Software-Intensive Systems Must Manage Emergent Behavior

What causes an effect like that of slowing traffic? It is, Fisher says, the result of emergent behavior, and may involve influence, cooperation, cascade effects, distributed control, and orchestration.

While the cause may be only of vexing curiosity to an automobile driver stuck in line before a tunnel, emergent behavior is inherent in complex, software-intensive systems. Understanding and harnessing these effects is crucial to success in systems of systems.

Related Links

**Training**

Migrating Legacy Systems to SOA Environments - eLearning

Service-Oriented Architecture: Best Practices for Successful Adoption

See more related courses »

"A system of systems *depends on emergent behaviors to achieve its purpose," Fisher says. Success in systems* of systems requires cooperation among many stakeholders and operational components toward a shared purpose.

Too often, however, failure of constituents to cooperate effectively and competing purposes result in software-intensive systems of systems that are plagued by interoperability problems, which lead to cost overruns, inadequate performance, schedule delays, and other difficulties.

"Exploiting emergent behavior offers great potential for systems of systems, not only to overcome the problems of interoperation but also to achieve levels of adaptability, scalability, and cost-effectiveness not possible in traditional systems," Fisher asserts.

Unfortunately, today's software and systems engineering practitioners do not have the methods and tools they need.

"We need new software and systems engineering methods that manage emergent behavior and exploit emergent effects to offer the possibility of cost-effective and predictable solutions in systems of systems," Fisher says, explaining one reason that he recently published a new report called *An Emergent Perspective on Interoperation in Systems of Systems* [Fisher 06].

## Examining Emergent Behavior in a System-of-Systems Context

To "facilitate discussion and reasoning about interoperation within systems of systems," Fisher explores "interdependencies among systems, emergence, and interoperation" and develops maxim-like findings such as these:

1. **Because they cannot control one another, autonomous entities can achieve goals that are not local to themselves only by increasing their influence through cooperative interactions with others**. Such cooperation can arise through the independent choice of a constituent or direct influence by neighbors. It is important to note, however, that every choice is influenced by the constituent's current situation, which is itself the result of past influences.

2. **Emergent composition is often poorly understood and sometimes misunderstood because it has few analogies in traditional systems engineering**. The erroneous view that emergence is unpredictable (and thus undesirable) arises, at least in part, from the difficulty in understanding how a quality attribute can arise through composition from parts that do not possess that attribute. An example of this kind of emergence is a reliable system that is built from unreliable components.

3. **Even in the absence of accidents, tight coupling can ensure that a system of systems is unable to satisfy its objectives**. Tight coupling refers to the degree that constituents depend on each other. The more tightly constituents are coupled and the more their actions and interactions are unnecessarily constrained, the more likely it is that failures will occur and the less likely that intended global properties will emerge. And yet, traditional systems engineering methods, requirements processes, and software tools are designed to simplify management, in effect, by maximizing constraints and coupling.

4. **If it is to remain scalable and affordable no matter how large it may become, a system's cost per constituent must grow less linearly with its size**. Whether the cost is measured in computational cycles, storage capacity, communications bandwidth, power consumption, dollars, number of defects fixed, or otherwise, costs not meeting this standard will eventually be unaffordable.

5. **Delay is a critical aspect of systems of systems**. Every action and interaction in a system consumes time. Delay is inherent in any communication. No technology can eliminate delay. Emergent effects always occur later than their causes.

These may be obvious, but are too often ignored.

## Reference

[Fisher 06]

Fisher, David. *An Emergent Perspective on Interoperation in Systems of Systems*. (CMU/SEI-2006-TR-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2006.

### Additional Information

*An Emergent Perspective on Interoperation in Systems of Systems.*



## For more information

Contact Us

info@sei.cmu.edu

412-268-5800

# Governance Issues for a Service-Oriented Architecture

NEWS AT SEI

Authors

Patrick R. Place

Dennis B. Smith

This library item is related to the following area(s) of work:

Performance and Dependability

This article was originally published in News at SEI on: June 1, 2006

A service-oriented architecture (SOA) is an architecture built around a collection of services with well-defined interfaces. A service is a coarse-grained, discoverable, and self-contained software entity that interacts with applications and other services through a loosely coupled, synchronous or asynchronous, message-based communication model. A system or application is designed and implemented using functionality from these services as part of its mission. SOA connotes a style or approach for developing a system. Thus *SOA* is not an alias for *system*, but rather describes a particular way in which systems might be constructed.

SOA systems have become more common because of the capability of accessing capabilities that are distributed across the Internet, and growing acceptance of approaches and technologies that support this system environment.

An example of a system built around an SOA is the Global Combat Support System–Air Force (GCSS-AF). The purpose of GCSS-AF is to provide timely, accurate, and trusted agile combat support (ACS) information to ACS personnel at all ranks to execute the Air Force mission throughout the spectrum of military operations. GCSS-AF includes an enterprise infrastructure for AF operations support with a common set of services, common software and hardware, a corporate data warehouse, and enterprise security.

The use of an SOA has implications for a number of aspects of traditional acquisition,

Related Links

**Training**

Software Architecture Design and Analysis

Modeling System Architectures Using the Architecture Analysis and Design Language (AADL)

See more related courses »

development, and sustainment. This article focuses on how governance, one of those aspects, is affected. Insights for these observations have come from our experience with several SOAs, including GCSS-AF.



### Range of SOA Systems

There is a wide spectrum of possible ways for systems to incorporate the principles of SOAs. One extreme of that spectrum would be a collection of services and applications —all running over a single, large infrastructure; existing unambiguously under a single authority; following consistent business rules; and executing defined processes known to all. In such a case, the boundary of ownership would be clear and encompassing. At the other extreme, we could imagine a collection of diverse, heterogeneous services, used in ad hoc fashion by a set of diverse applications, facilitated by various infrastructure capabilities, and governed by business rules known only to the applications themselves. The boundaries of ownership there would likely be at best complex and at worst highly ambiguous. The one extreme is one of absolute authority; the other is one of near-anarchy. The reality of large systems based on an SOA approach lies somewhere in the middle.

Given this reality, it is important to consider how services and applications relate to the boundaries of ownership domains. GCSS-AF is certainly an SOA that is entirely within the domain of the U.S. Air Force as a whole. But multiple participating organizations, separate divisions of the AF, and separate domains of interest (e.g., logistics, human resources, and finance) mean that for GCSS-AF, there are indeed different owners and diverse system boundaries.

### Role of Governance

The word *governance* describes many facets of management and policy for SOA-based systems. Depending on context, *governance* can refer to

- overseeing and enforcing policy (business design, technical design, application security) that directs the organization
- creating policy that directs the organization
- coordinating the people, policies, and processes that provide the framework for management decision-making
- taking action to optimize outcomes related to an individual's responsibility
- promoting efficiency in the organization
- determining the integrity of services

Good governance exemplifies such characteristics as accountability, freedom of association and participation, a sound judicial system, freedom of information and expression, capacity building, and similar things.

The concept of governance is not entirely new. What is new is that development and acquisition, are, in an SOA context, very different, and management and policy apply to more than simply construction. However, like most aspects of management, traditional or otherwise, governance will be reflected in one or more processes.

### Characteristics of Good SOA Governance

In the context of SOA, governance should encourage active and efficient use of available services by application builders. While a number of characteristics of governance apply within the SOA context, we will focus in this column on just the following:

- a flexible authority structure
- management incentives
- full operational life cycle

We describe these characteristics in terms of traditional system life-cycle development and discuss some of the differences that arise with SOAs. We then consider how the changed context of SOA affects the concept of governance, with examples drawn from the GCSS-AF experience as well as other examples.

### A Flexible Authority Structure

In traditional systems development, the authority for a software system strongly resides in the development process. Ownership is primarily centered on the agency that acquires and develops a system, and most concepts of software management are related to the traditional aspects of the software life cycle (e.g., requirements, design, testing, and maintenance).

By contrast, SOAs demand governance across all of the communities involved—operation, development, and acquisition. Ownership cannot rest solely in the development phase. Lines of authority are at once far broader and less well-defined, especially for large systems for which ownership is shared across multiple domains. Thus the context of SOAs involves multiple, overlapping networks of nodes–service providers, service consumers, and nodes that both provide and consume services. Because many service providers will participate in multiple systems and multiple processes simultaneously, the boundary lines of a system become blurred. Any successful SOA governance process will, by definition, be governance of only some part of the whole. Such a governance process will embrace rather than contest the notion of flexible authority and control.

Correspondingly, an SOA governance policy must be defined based on the realization that influence will be as valuable as control. The policy will necessarily take a distributed and cooperative view. Governance will depend on varying spheres of influence. Governance of a given sub-network of nodes (e.g., a particular system or SOA) may be well-defined, but in the aggregate, governance of the whole cannot be well-defined.

### Management Incentives

A number of incentives are related to traditional software development; the primary focus of the interested parties is on the system being developed. Thus, the contractor bids on and receives a contract to build the system. The acquirer lets that contract and is rewarded when the system is successfully built and deployed. This paradigm is equally true for the vendor who builds and sells commercial off-the-shelf (COTS) products. The product for sale is the system, regardless of any internal complexity, number of modules, and so forth; the incentive is to sell the system for a profit. In all of these cases, the primary focus is on *the system*, and the incentives flow from that focus.

The operational concepts required by an SOA demand that we rethink incentives for individuals and organizations. First, the system is far more diffuse; it may be difficult to establish where the boundaries of the system actually are. The contractor may be creating a system made up of a collection of applications that may depend on external services over which the contractor has ceded authority, design, performance characteristics, and evolution. In place of the familiar incentives of contractual obligations, the contractor now deals with service providers who offer service-level agreements (SLAs), which seldom are specified as requirements once were.

In addition, service providers, especially of Web services, have a changed concept of profit, because their services exist in a far more volatile marketplace than a COTS vendor's products. While service providers want to keep providing the service, they also have an incentive to attract more customers by constantly adding improvements. Yet, they gain little knowledge of how customers use the services; they may know only that customers pay a fee. As a result, they do not know whether their improvements will benefit existing clients.

There is also a different concept of success for acquirers. With SOAs, system deployment is likely to occur iteratively, and fielding a system is now fielding a collection of capabilities. The time frame for such ongoing deployment will likely span several years, making it difficult to clearly define when a program comes to an end and what its level of success has been.

In such an environment, governance has a vital relationship to managing incentives. An obvious incentive for an enterprise to use the services in an SOA environment is the promise of reduced development costs or delays. In other words, why should the enterprise underwrite the expense of creating some functionality when it is available at

a lower cost through such mechanisms as SLAs?

Incentives are also needed across industry boundaries. For instance, the government may desire that company A make use of various services in creating an application for a government contract. But if using many of those services provides a favorable revenue stream for company B (a competitor with company A), there is a strong disincentive for A to use those services.

In addition, there is a tension between local and global optimization; this is an instance where SOA differs from other system-of-system contexts. For example, it is often the case, for some system-of-system contexts, that local optimization defeats the global goals and that to optimize for the global means accepting suboptimal behavior at the local level. In such cases, it is conceivable that an owner might act for the greater good and accept suboptimal behavior in its system.

However, in an SOA context, service providers usually cannot optimize for global goals: they cannot know who is using the service, and any optimization that favors one user might be detrimental for others. Thus, governance for an SOA can consider incentives only at the global level, where the application developer operates.

### Full Operational Life Cycle

As with many other aspects, governance of software systems has generally focused on system development, and sometimes on deployment. Once a system has been deployed, most of the familiar management controls (especially those related to government acquisition) are at best replaced by another set of management policies or at worst simply removed. In traditional systems development, management control tends to be vested in a single person or agency. Thus, for a given system, there is a single acquisition authority during development and deployment, and after deployment, there is typically a single person or agency that owns the operation of the system.

In the context of SOA, the notion of a system becomes distributed widely among infrastructure providers, service providers, and application developers. Governance, too, becomes distributed, with shared responsibilities among several different agencies. As a result,

- The governance process will not be focused purely on the integration of services; it will cover things such as changes to a service beneath the interface, changes to the interface itself, and retirement of a service.
- The governance process will probably not describe how to migrate legacy applications to a service basis.
- The governance process will focus more on service providers regardless of whether they are providing infrastructure services or application-level services.

There is an important distinction for an SOA between governance of design-time issues and governance of runtime issues. In addition, there are also distinctions throughout the various parts of the overall life cycle in how governance is divided among the infrastructure provider, the service provider, and the application developer.

Thus, at design time, a governance process must constrain the construction of services, at least to enforce infrastructure requirements (whether they are stringent or flexible). In addition, a governance process will consider how services are registered and discovered. The parties affected by governance at design time are the infrastructure provider and the various service providers. At least some of the constraints will originate with the infrastructure provider, though the more likely condition is that constraints will be the result of negotiated agreements between infrastructure and service providers. Another set of negotiations, probably with still weaker binding force, will occur between different infrastructure providers, toward the goal of making services available on different infrastructures and through different registries.

However, at runtime, governance would deal with the definition of how a service behaves when called, how various policies (e.g., security) are enforced, how behavior is validated, and how services are replaced and retired.

## Conclusion

SOAs have implications for a number of aspects of the traditional software acquisition, development, and sustainment cycle. As we gain additional experience with the use of SOAs, we will continue to refine these preliminary ideas on governance.

## About the Authors

Dennis Smith is the lead for the SEI Integration of Software Intensive Systems (ISIS) Initiative. This initiative focuses on addressing issues of interoperability and integration in large-scale systems and systems of systems. Earlier, he was the technical lead in the effort for migrating legacy systems to product lines. In this role he developed the method Options Analysis for Reengineering (OARS) to support reuse decision-making. Smith has also been the project leader for the CASE environments project. This project examined the underlying issues of CASE integration, process support for environments and the adoption of technology. Smith has published a wide variety of articles and technical reports, and has given talks and keynotes at a number of conferences and workshops. He has an MA and PhD from Princeton University and a BA from Columbia University.

Pat Place is a senior member of the technical staff in the Integration of Software Intensive Systems (ISIS) initiative at the Software Engineering Institute (SEI). Recently he has been working on the theory and practice of interoperability in systems of systems. Prior work at the SEI has included the development of CURE and investigations into COTS-based development processes, communications in distributed real-time systems, architectures for flight simulators, and system specification using formal techniques. He also holds a position as adjunct professor at Carnegie Mellon University and teaches in the Carnegie Mellon Master of Software Engineering program. Before joining the SEI, he worked at various companies either creating software development tools or formal specifications, sometimes both.

The views expressed in this article 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.

# CMMI-DEV Version 1.2: What Else Has Changed?

NEWS AT SEI

Author

Mike Phillips

This library item is related to the following area(s) of work:

Process Improvement

CMMI

This article was originally published in News at SEI on: July 1, 2006

CMMI for Development (CMMI-DEV), Version1.2 includes improvements to all parts of the CMMI Product Suite in response to issues that have emerged in practice. The changes focus on improving the quality of CMMI products and the consistency of how they are applied.

The CMMI V1.2 Product Suite contains the following items:

- Appraisal Requirements for CMMI (ARC), V1.2
- SCAMPI A V1.2: Method Definition Document
- Introduction to CMMI V1.2 course

This latest version of the model integrates bodies of knowledge that are essential for development and maintenance but that have been addressed separately in the past, such as software engineering, systems engineering, and others. CMMI V1.2 embodies the concept of CMMI constellations, in which a set of core components can be augmented by additional materials. CMMI-DEV is the first such constellation. Currently there are plans for three constellations supported by the V1.2 model framework: development, acquisition, and services.

## What are the major changes to the model document?

In earlier columns, I described most of the intended improvements that would be considered for the CMMI V1.2 model. These changes included eliminating three

Please note that current and future CMMI research, training, and information has been transitioned to the CMMI Institute, a wholly-owned subsidiary of Carnegie Mellon University.

Related Links

**News**

SEI to Co-Sponsor 26th Annual IEEE Software Technology Conference

SEI Cosponsors Agile for Government Summit

**Training**

See more related courses »

**Events**

Team Software Process (TSP) Symposium 2014

Nov 3 - 6

process areas-ISM, OEI, and IT-while preserving coverage of these important practices. This change reduced the size of the model document by 15%, despite adding coverage of the work environment and hardware engineering. The bulk of the model-development effort involved revising the wording of model content that users found to be ambiguous, redundant, or confusing.

The CMMI sponsors and Steering Group also directed a final change to the model document based on concerns related to project start-up. The organizational process focus (OPF) process area already provided some guidance for the organization to deploy improved process artifacts to its projects, but the importance of effective new-project start-up was not emphasized. The CMMI Product Team formulated a new approach in which the second goal in OPF was split into two separate goals. The first of these (SG2) emphasizes the organization's need to develop organizational standard processes and their associated artifacts. The second (now SG3) emphasizes the need to provide these standard processes and artifacts to new projects and to monitor the projects' progress in effective start-up using the organizational process artifacts deployed. These changes to OPF do not represent a new CMMI requirement, but do clarify that improvement should be pursued throughout the organization, including new projects just starting. Emphasizing project start-up can only enhance the overall success of process improvement efforts across the enterprise.

### What are the major changes to SCAMPI?

While there are few changes to the actual SCAMPI method itself, there are numerous policy changes designed to improve confidence in appraisal results. Several of the changes demonstrate how the SCAMPI Upgrade Team has focused on improving the quality of CMMI appraisals.

One change, which I mentioned in an earlier column, is that there is now a limited period of validity for CMMI appraisals. Now that V1.2 has been released, all CMMI appraisals are considered valid for a maximum of three years. This limitation applies to both V1.1 and V1.2 appraisals.

Because of the move to a new product suite, there is a grace period of one year-until August 31, 2007. In other words, if your last appraisal was, say, conducted six years ago, you will have one year (until August 31, 2007) to conduct another one before the current one will be considered invalid. However, if your appraisal was completed two years or less before the release of CMMI V1.2, it has the three year period of validty.V1.1 appraisals will be accepted by the SEI through the end of August 2007.

Another way to think about this transition to a limited period of validity is that beginning on September 1, 2007, no valid CMMI appraisal can be more than three years old.

As of September 1, 2007, all new CMMI appraisals must be V1.2 appraisals. A V1.2 appraisal is one in which

- both the model and appraisal method used are from the V1.2 CMMI Product Suite and
- all appraisal team members have either attended a V1.2 Introduction to CMMI training course or have upgraded from V1.1 using the CMMI Version 1.2 Upgrade Training online course

The Version 1.2 appraisal disclosure statement (ADS) was the chief focus of SCAMPI improvements. This document is the summary of results for an appraisal. (By the way, if your supplier is claiming CMMI results, ask to see this document to gain confidence in these claims.)

An improvement to the ADS is having the sponsor sign the ADS. The V1.2 ADS is to be used for both V1.1 and V1.2 appraisal reporting for any appraisal starting on or after November 1, 2006. Because of the importance of appraisal quality, lead appraisers can perform V1.2 appraisals only after they

(1) successfully complete CMMI Version 1.2 Upgrade Training for Instructors, Lead Appraisers, and Team Leaders

(2) pass an exam, and
(3) attend a Face-to-Face Workshop. The first of these workshops is planned for October 19, 2006, in Charlotte, N.C., at the annual CMMI Lead Appraisers and Instructors Workshop. Therefore, no V1.2 appraisals can begin until after that first workshop.

The workshop will also be offered at the CMMI Technology Conference and User Group in Denver on November 17, 2006; at SEPG 2007 in Austin, Texas; and at E-SEPG 2007 in Amsterdam, Netherlands. Other offerings are being considered as well.

High-maturity appraisals are another focus for improvement in V1.2. A small but significant change is that the ADS will need to document the specific linkage of high-maturity efforts with business objectives in the appraisal organization. This requirement is designed to address concerns about being able to claim high maturity when performing any statistical process improvement somewhere in the organization. Such an approach to statistical process improvement is not consistent with the intent of the model. The improvements to CMMI V1.2 are designed to assure that the quality improvements associated with a maturity model are, in fact, aligned with business objectives. This alignment is clearly consistent with the CMMI model practices and intent.

A more significant change is that lead appraisers must specifically qualify as high-maturity appraisers. A team was created, comprising both industry and SEI expertise, to establish an initial approach for lead appraiser high-maturity certification and a process for building high-maturity appraisal skills among lead appraisers. Note that because no lead appraiser will be automatically authorized for high-maturity certification, no appraisers will be available to determine levels 4 or 5 for V1.2 appraisals until after the CMMI Lead Appraisers and Instructors Workshop in Charlotte. Therefore, phasing in this approach may mean that some high-maturity appraisals currently planned for the near future may need to remain V1.1 appraisals or be rescheduled further in the future.

### What are the major changes to training?

The CMMI training team has tested the revised Introduction to CMMI, Version 1.2 training. This new Introduction to CMMI training is now available from the SEI and will soon be available from SEI Partners. Version 1.2 of the Intermediate, Instructor Training, SCAMPI Lead Appraiser Training, and SCAMPI B and C Team Leader Training courses also is available.

An online CMMI V1.2 Upgrade Training course is available to all of the more than 54,000 people who have taken the Introduction to CMMI course. This course describes, in detail, the changes to the CMMI Product Suite and reviews key CMMI concepts. The description of this course is now available on the SEI Web site.

Lead appraisers must assure that appraisal team members are qualified to participate on V1.2 appraisal teams. Potential appraisal team members who have completed Introduction to CMMI training prior to V1.2 can take the upgrade course online and register with the training registry to fulfill the requirement for appraisal team participation.

Lead appraisers and instructors must take an upgrade training course online as well. Their course contains more information and includes an exam.

### So what are the key transition dates again?

All of the key documents associated with V1.2 are available today. The number of available V1.2 courses will increase as instructors become qualified. The last V1.1 classes will be held in December 2006.

Beginning November 1, 2006, all appraisals must use the improved Version 1.2 Appraisal Disclosure Statement.

During the transition period, both V1.2 and V1.1 appraisals will be accepted by the SEI. That period is one year, ending on August 31, 2007, after which only V1.2 appraisals will be accepted. The end of the transition period also marks the time when the 3-year period of maximum validity takes full effect and appraisals completed any earlier than

September 2004 become invalid.

## Summary

The update to V1.2 includes a number of improvements across the product suite to make this process improvement tool more usable and to increase confidence in its results. In the next column, I will write more about making the transition to the new version.

## About the Author

As the director of special projects at the Software Engineering Institute, Mike Phillips leads 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, Ohio. In addition to his bachelor's degree in astronautical engineering from the U.S. Air Force Academy, Phillips has master's 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.

# CERT Launches Secure Coding Standards Web Site

NEWS AT SEI

This article was originally published in News at SEI on: July 1, 2006

The Carnegie Mellon Software Engineering Institute (SEI) CERT Program has deployed a secure-coding Web site at www.securecoding.cert.org as a forum in which software developers can codify a practical and effective set of secure coding practices for popular programming languages. These coding practices can then be used by software developers to eliminate vulnerabilities before software is operationally deployed.

The purpose of this project is that the practices can be used by developers for professional development and as the basis for organizational coding standards supporting the quality of their products.

Jeffrey Carpenter, manager of the CERT Coordination Center, says that the project is part of a larger secure-coding initiative within the CERT/CC to eliminate dangerous coding practices that can result in exploitable software vulnerabilities. According to Carpenter, “CERT is in a unique position to coordinate development of a set of secure coding practices because of its long history in analyzing and responding to software vulnerabilities.”

CERT's initial efforts are focused on the development of secure coding practices for the C and C++ programming languages. CERT senior vulnerability analyst Robert Seacord is leading the secure-coding initiative. Seacord is a leading authority on secure coding, author of the book Secure Coding in C and C++ [Seacord 05], and technical expert for the ISO/IEC JTC1/SC22/WG14 international standardization working group for the C programming language.

“C and C++ were selected because a large percentage of critical infrastructures are developed and maintained using these programming languages,” Seacord says. “C and C++ are popular and viable languages although they have characteristics that make them prone to security flaws.”

“Today’s dependency on networked software systems has been matched by an increase in the number of attacks against governments, corporations, educational institutions, and individuals. These attacks result in the loss and compromise of sensitive data,

system damage, lost productivity, and financial loss," says Seacord. To address this growing threat, the introduction of software vulnerabilities during development and ongoing maintenance must be significantly reduced, if not eliminated.

There are a number of available resources, both online and in print, containing coding guidelines, best practices, suggestions, and tips. The Motor Industry Software Reliability Association (MISRA) developed guidelines for the use of the C language in critical systems [MISRA 04], and more recently the U.S. Department of Homeland Security launched its *Build Security In* Web site to promote the codification of practices and rules. These sources, however, do not provide a prescriptive set of secure coding practices that can be uniformly applied in the development of a software system.

"Without secure coding practices, software vulnerability reports are likely to continue on an upward trend," Seacord says. "At CERT/CC, we have had nearly 4,000 vulnerabilities reported in the first half of 2006. To stop the threats, we need to develop secure software from the outset."

The secure coding practices proposed by CERT are based on standard language versions as defined by official or *de facto* standards organizations such as ISO/IEC. CERT is not an internationally recognized standards body, but plans to work with organizations such as ISO/IEC to advance the state of the practice in secure coding. The ISO/IEC JTC1/SC22 WG14 international standardization working group for the programming language C, for example, has offered to provide direction in the development of the C-language secure coding practices and to review and comment on drafts of the informal CERT standard.

According to WG14 convener John Benito, "The secure coding standard is going in the correct direction, and I have confidence the final product will be useful to the community."

CERT is also working with standards groups, such as the ISO/IEC working group on Guidance for Avoiding Vulnerabilities through Language Use (OWGV). While the ISO/IEC group is working on providing language-independent guidance, CERT is working on developing and consolidating the language-specific guidance that provides the foundations for the ambitious goals of OWGV.

Jim Moore, convener of OWGV, states that "CERT's efforts in identifying and documenting secure coding practices for C and C++ will contribute to the standardization of these practices and advance the goals of the OWGV."

### Community Involvement

The success of the secure coding standards depends largely on the active involvement of members of the secure software and C and C++ development communities. Rules and recommendations for each coding practice are solicited from the communities involved in the development and application of each programming language, including the formal or de facto standard bodies responsible for the documented standard.

These rules and recommendations are edited by CERT senior members of the technical staff for content and style and placed on the secure coding standards Web site for comment and review. Users are invited to discuss and comment on the publicly posted content. Once a consensus develops that the rule or recommendation is appropriate and correct, the final rule is incorporated into the coding standard.

Once practices are documented, tools can be developed or modified to verify compliance. Compliant software systems can then be certified as compliant by a properly authorized certification body. Seacord also envisions a training-and-development program to educate software professionals regarding application of secure coding practices.

The development of secure coding practices is a necessary step to stem the ever-increasing threat from software vulnerabilities. CERT's goal is that the enumeration of secure coding practices will allow for a common set of criteria that can be used to measure and evaluate software development efforts.

The public can review or comment on the existing content at the secure coding Web site or submit new ideas for secure coding practices by e-mailing secure-coding@cert.org. Robert Seacord can be reached at rcs@cert.org.

## References


[1] Seacord, R. *Secure Coding in C and C++*. Addison-Wesley, 2005. See http://www.cert.org/books/secure-coding for news and errata.

[2] MISRA C: 2004 *Guidelines for the use of the C language in Critical systems*. MIRA Limited. Warwickshire, UK. October 2004. ISBN 0 9524156.

# Claims You'll Need to Justify to Assure That Your System Will Succeed

NEWS AT SEI

Author

John Morley

This library item is related to the following area(s) of work:

Software Assurance

This article was originally published in News at SEI on: July 1, 2006

Suppose that your team is developing a state-of-the-art, software-intensive system expected to contribute mightily to your organization and its mission. You know that your team will have to gain confidence that the system will meet expectations and perform as expected.

*Unfortunately, you cannot test everything.*

Your system might really be a system of systems. Once deployed, it would rely on interoperation among its member systems and on the emergent behavior (i.e., behavior that is not resident in any individual system) that interoperation produces. Testing every detail about those systems and their interactions, if even possible, would take too long and cost too much. How can you keep testing costs under control without releasing unacceptable systems?

Or, your system could be a high-assurance system, which demonstrates a condition that occurs rarely, for example. "Because a condition is rare, perhaps showing up only once in a million transactions, it might not be tested for before the system is deployed," explains Charles Weinstock, senior member of the technical staff at the Software Engineering Institute (SEI). "If in operation, however, the system's workload increases to the level of a million transactions per day, this 'rare' failure will occur daily, on average."[1]

So, if testing isn't enough, what else can you do to support claims about your system's

Related Links

**Training**

Security Requirements Engineering Using the SQUARE Method

Software Assurance Methods in Support of Cyber Security

dependability such as:

- My system will perform as expected.
- My system achieves the level of interoperability required.
- My system is safe.
- My system is secure.

### Promising Approach Relies on Evidence and Arguments

The SEI is investigating analysis-based assurance, an approach that shows promise for helping to gain confidence in claims about the quality of systems of systems and high-assurance systems.

"Analysis-based assurance is a set of techniques that improves confidence that a system or component of a system will perform as expected," Weinstock explains. "It can replace testing where testing at all would be impossible and augment it where thorough testing would be infeasible or too costly. Or analysis-based assurance can reduce the number of tests that would need to be designed to assure the desired performance."

An organization can use analysis-based assurance methods to leverage scarce assurance resources and produce a reviewable artifact that provides assurance of mission-critical properties and go/no-go criteria at different stages of development.

This technology can aid organizations in other ways, such as

- supporting an engineering culture characterized by explicit articulation and verification of claims
- providing guidance for system-of-systems test and evaluation techniques
- promoting evidence-based high-assurance practices

### Structured Evidence and Argument Used To Support or Refute a Claim

Central to analysis-based assurance is the assurance-case method. An assurance case is a structured set of arguments and a body of evidence showing that a system satisfies specific claims with respect to a given quality attribute.[2] Because it involves a rigorous reasoning process, the assurance-case method can be widely applied—used to satisfy, for instance, claims you need to justify regarding your system's performance, interoperability, safety, and security.


*Figure 1: An assurance case is concerned with building evidence-based arguments to demonstrate claims about the behavior of a system*

For example, you may need to prove a claim stated as "My system is dependable" for your system. "We would seek to know more about which dependability properties are important in this particular system," says John Goodenough, SEI fellow and leader of the SEI investigation of analysis-based assurance. "A more appropriate claim would be, 'The system meets its dependability requirements as detailed in document XXX.'"

Using the assurance-case method, you can then devise a way to show convincingly that the claim is true. "In this example, a strategy might be 'Show that each of the dependability requirements is met individually. Then show that they are met

collectively,'" says Goodenough.

"This leads to sub-claims, at least one for each of the dependability requirements, with a strategy and sub-sub-claims for each. Eventually the argument gets down to *ground truth*[3]—which might be a formal proof, a law of physics, or perhaps even an exhaustive enumeration of possibilities. Once every sub-claim is successfully driven down to its solution, we have an argument that the original claim has been satisfied.

Importantly, this argument can be referred to whenever a question about the claim is raised. In particular, it can be used to identify potential problems when a change in the system is contemplated," Goodenough notes.

In *Dependability Cases*, Weinstock, Goodenough, and John Hudak (also a senior member of the SEI technical staff) provide an example of the approach in use through the building of a dependability case for the problem of synchronizing the clocks of a low-orbiting satellite and its ground-control station. In the example, they discuss and illustrate the claim, context, and assumptions about the problem; a strategy to "prove" the claim; sub-claims for the sources of the problem; and sub-dependability cases.

## Assurance-Case Patterns Can Reduce Effort

It is common for assurance-case arguments with the same structure to appear in many different contexts. Essentially, the recurring arguments are assurance-case patterns of reasoning that encapsulate what evidence needs to be collected and what pitfalls are likely to occur in applying a particular type of argument. They also capture the expertise of people in an organization who know how to attain assurance.

Assurance-case patterns provide a body of best practices in the form of ready-to-instantiate arguments where only the specific details need to be incorporated, reducing the effort needed to develop an assurance case. As a result, the use of these patterns could help an organization leverage scarce assurance resources.

A model for an assurance-case pattern drawn from the safety-critical milieu is the "as low as reasonably practicable" (ALARP) principle that argues in this way: when a particular hazard cannot be eliminated or reduced to a negligible level, an argument needs to be developed showing that the risk has been reduced to a tolerable level, and any further reduction is impractical.

## SEI To Work with Organizations on Assurance

After initial work establishing assurance cases as part of software development practice, the SEI is seeking organizational partners to develop case studies and a catalog of assurance-case patterns. The SEI can help partner organizations to improve how they establish and sustain a sense of assurance by reviewing the "as-is" state of their assurance practices and recommending a desired state and path forward.

For more information, contact us using the link the *For More Information* box at the bottom of this page.

1 This example is adapted from *Ultra-Large Scale Systems: The Software Challenge of the Future*.

2 The concept of an assurance case has been derived from that of a safety case. Safety cases have been used successfully for more than a decade to argue that safety-critical systems in areas such as flight control, railroad signaling, and nuclear reactor shutdown systems meet their safety properties. It also has origins in the concept of a dependability case, described by the Performance-Critical Systems (PCS) Initiative at the SEI in the technical note *Dependability Cases*. Both safety cases and dependability cases are types of assurance cases: a safety case supports a safety claim; a dependability case supports a dependability claim.

3 In the context of assurance cases, *ground truth* can be taken to mean elemental, basic, or fundamental.

Find Us Here

Share This Page

## For more information

Contact Us

info@sei.cmu.edu

412-268-5800