# Results of SEI Independent Research and Development Projects and Report on Emerging Technologies and Technology Trends

John Bergey
Sven Dietrich
Donald Firesmith
Eileen Forrester
Angel Jordan
Rick Kazman
Grace Lewis
Howard Lipson
Nancy Mead
Ed Morris
Liam O'Brien
Jeannine Siviy
Dennis Smith
Carol Woody

*October 2004*

TECHNICAL REPORT
CMU/SEI-2004-TR-018
ESC-TR-2004-018

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

# Results of SEI Independent Research and Development Projects and Report on Emerging Technologies and Technology Trends

CMU/SEI-2004-TR-018
ESC-TR-2004-018

John Bergey
Sven Dietrich
Donald Firesmith
Eileen Forrester
Angel Jordan
Rick Kazman
Grace Lewis
Howard Lipson
Nancy Mead
Ed Morris
Liam O'Brien
Jeannine Siviy
Dennis Smith
Carol Woody

*October 2004*

**SEI Director's Office**

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Christos Scondras
Chief of Programs, XPK

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.



This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

# Table of Contents

# List of Figures

# List of Tables

# Abstract

Each year, the Software Engineering Institute (SEI) undertakes several Independent Research and Development (IR&D) projects. These projects serve to (1) support feasibility studies investigating whether further work by the SEI would be of potential benefit, and (2) support further exploratory work to determine whether there is sufficient value in eventually funding the feasibility study work as an SEI initiative. Projects are chosen based on their potential to mature and/or transition software engineering practices, develop information that will help in deciding whether further work is worth funding, and set new directions for SEI work. This report describes the IR&D projects that were conducted during fiscal year 2004 (October 2003 through September 2004). In addition, this report provides information on what the SEI has learned in its role as a technology scout for developments over the past year in the field of software engineering.

# 1 Introduction

This document briefly describes the results of the independent research and development projects conducted at the Carnegie Mellon Software Engineering Institute (SEI) during the 2003–04 fiscal year. It also provides information about what the SEI has learned in its role as a technology scout for developments over the past year in the field of software engineering.

## 1.1 Purpose of the SEI Independent Research and Development Program

SEI independent research and development (IR&D) funds are used in two ways: (1) to support feasibility studies investigating whether further work by the SEI would be of potential benefit and (2) to support further exploratory work to determine if there is sufficient value in eventually funding the feasibility study work as an SEI initiative. It is anticipated that each year there will be three or four feasibility studies and that one or two of these studies will be further funded to lay the foundation for the work possibly becoming an initiative.

Feasibility studies are evaluated against the following criteria:

- Mission criticality: To what extent is there a potentially dramatic increase in maturing and/or transitioning software engineering practices if work on the proposed topic yields positive results? What will the impact be on the Department of Defense (DoD)?
- Sufficiency of study results: To what extent will information developed by the study help in deciding whether further work is worth funding?
- New directions: To what extent does the work set new directions as contrasted with building on current work? Ideally, the SEI seeks a mix of studies that build on current work and studies that set new directions.

At a DoD meeting in November 2001, the SEI's DoD sponsor approved a set of thrust areas and challenge problems to provide long-range guidance for the SEI research and development program, including its IR&D program. The thrust areas are survivability/security, interoperability, sustainability, software R&D, metrics for acquisition, acquisition management, and commercial off-the-shelf products. The IR&D projects conducted in FY2004 were based on these thrust areas and challenge problems.

### 1.1.1 Overview of IR&D Projects

The following research projects were undertaken in FY2004:

- Levels of Anonymity and Traceability (LEVANT)
- Architecture-Based Self-Adapting Systems
- Eliciting and Analyzing Quality Requirements: A Feasibility Study
- Enabling Technology Transition Using Six Sigma
- A Method to Analyze the Reuse Potential of Non-Code Software Assets

These projects are described in detail in this technical report.

## 1.2 Purpose of Technology Scouting

Technology scouting has always been an implicit activity of the Software Engineering Institute and is embedded in the SEI's mission of technology transition. Because of the institute's small size relative to other research institutions, the SEI applies the most leverage to its active initiatives, but it also watches for other emerging technologies, in the U.S. and internationally.

The SEI has recently been asked to report on the state of the art of software technologies—those that are pushing the frontiers of the SEI's current programs and initiatives and also those that transcend them.

### 1.2.1 Overview of the 2004 Report on Emerging Technologies and Technology Trends

In this report, we have provided descriptions of new or emerging technologies. These descriptions include the technologies' purpose and origin. Where possible, we have indicated the technologies' level of maturity and have provided information about related trends. The following technologies are described:

- Open Grid Services Architecture
- Integrated Security Services for Dynamic Coalition Management
- Model-Driven Architecture
- Service-Oriented Architecture
- Automated Lexical and Syntactical Analysis in Requirements Engineering
- Q Methodology

- Emergent Algorithms for Interoperability
- Aspect-Oriented Software Development
- Generative Programming
- Software Assurance
- Recent Advances in Intrusion Detection Systems
- Applying Statistics in Software Engineering
- Advances in Software Engineering Processes

# 2 Levels of Anonymity and Traceability (LEVANT)

**Howard Lipson, Sven Dietrich**

## 2.1 Purpose

In the cyber world, the current state of the practice regarding the technical ability to track and trace Internet-based attacks is primitive at best. Sophisticated attacks can be almost impossible to trace to their true source. The anonymity enjoyed by today's cyber-attackers poses a grave threat to the global information society, the progress of an information-based international economy, and the advancement of global collaboration and cooperation in all areas of human endeavor.

Society continues to migrate increasingly critical applications and infrastructures onto the Internet, despite severe shortcomings in computer and network security and serious deficiencies in the design of the Internet itself. Internet protocols were designed for an environment of trustworthy academic and government users, with applications that were oriented primarily toward research and information exchange. In this era of open, highly distributed, complex systems, vulnerabilities abound and adequate security, using defensive measures alone, can never be guaranteed. As with all other aspects of crime and conflict, deterrence plays an essential role in protecting society. Hence, the ability to track and trace attackers is crucial, because in an environment of total anonymity, deterrence is impossible, and an attacker can endlessly experiment with countless attack strategies and techniques until success is achieved. The ability to accurately and precisely assign responsibility for cyber-attacks to entities or individuals (or to interrupt attacks in progress) would allow society's legal, political, and economic mechanisms to work both domestically and internationally, to deter future attacks and motivate evolutionary improvements in relevant laws, treaties, policies, and engineering technology. On the other hand, there are many legal, political, economic, and social contexts in which some protection of anonymity or privacy is essential. Without some degree of anonymity or privacy, individuals or entities whose cooperation is vitally needed may not fully participate (or participate at all) in the use or operation of systems that support the critical functions of the global information society.

Hence, traceability and anonymity are attributes that are central to the security and survivability of mission-critical systems. We believe that principled, fine-grained tradeoffs between

traceability and anonymity are pivotal to the future viability of the Internet. However, such tradeoffs are rarely explicitly made, the current capability to make such tradeoffs is extremely limited, and the tradeoffs between these attributes have occurred on an ad hoc basis at best. This study, which will carry over into FY2005, is investigating the feasibility of a disciplined engineering design of Internet protocols (in the context of key policy issues) to allow optimal, fine-grained tradeoffs between traceability and anonymity to be made on the basis of specific mission requirements. We see this study as a first step toward the development of a discipline of Internet engineering, which would translate traditional design and engineering processes, such as thorough requirements gathering and attribute tradeoff analyses, into the unique context of the Internet environment and its associated security and survivability risks [Lipson 99].

## 2.2 Background

Malicious users exploit the severe weakness in existing Internet protocols to achieve anonymity, and use that anonymity as a safe haven from which to launch repeated attacks on their victims. However, Internet users often want or need anonymity for a variety of legitimate reasons. On the other hand, service providers and other victims of cyber-attack want and need traceability for accountability, redress, and deterrence. The engineering challenge is to balance the apparently conflicting needs of privacy and security, and to allow considerable flexibility in doing so by providing fine-grained “levels” of anonymity and traceability.

Existing Internet protocols were never engineered for today’s Internet, where the trustworthiness of users cannot be assumed, and where high-stakes mission-critical applications increasingly reside. Today’s Internet protocols were initially developed for a small prototype ARPANET, and later for the research-oriented NSFnet, both of which supported communities of highly trustworthy academic and government users. Our current track-and-trace capability is limited in the extreme by the existing protocol and infrastructure design, and requires a major reengineering effort from both technical and policy perspectives, as described in an SEI special report funded by the U.S. State Department [Lipson 02].

## 2.3 Approach

In any Internet transaction, trust ultimately depends not on IP addresses but on particular relationships among individuals and their roles within organizations and groups (which may be economic, political, educational, or social). Trust cannot be established while maintaining total anonymity of the actors involved. It goes without saying that there is a great need for privacy on the Internet, and it must be carefully guarded. However, trust and privacy tradeoffs are a normal part of human social, political, and economic interactions, and such tradeoffs can be resolved in a number of venues, for example in the marketplace. Consider the telephone system, in particular the caller identification (caller ID) feature, which displays the

phone number, and often the name, associated with incoming calls. Caller ID is a feature for which many customers are willing to pay extra in return for the privacy benefits associated with having some idea of who's calling before they answer a call. However, callers are sometimes given the option of being anonymous (i.e., not identifiable by the caller ID feature) by default or on a call-by-call basis. To more fully protect their privacy, the caller ID customer can choose to block all incoming calls from anonymous callers. The anonymous caller is notified of this fact by an automated message. For callers that pre-arrange with their phone company to be anonymous by default, the only way to complete the call is to enter a key sequence to remove the anonymity for this particular call and to redial. Customers that achieve anonymity on a call-by-call basis (by entering a specific key sequence) can choose to redial without entering the key sequence that denotes anonymity. This choice is a form of negotiation between the caller and the intended recipient of the call, and it is a tradeoff between anonymity and trust that is supported by the technology of caller ID and the marketplace. There is no government mandate that all calls must be anonymous or that no calls are allowed to be anonymous. The individual caller chooses whether or not to relinquish anonymity (or some degree of privacy) in exchange for the perceived value of completing the call by increasing the degree of trust as seen by the recipient.

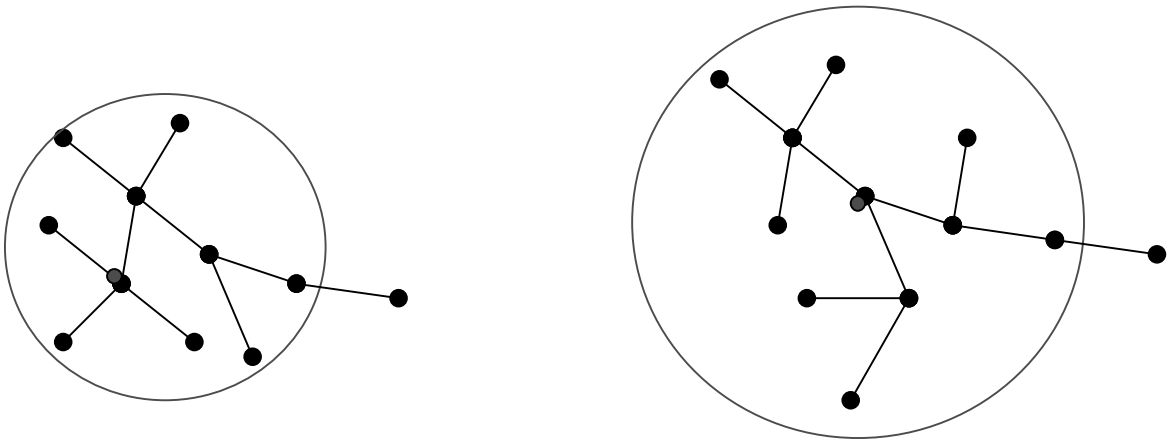
One can envision next-generation Internet protocols supporting this kind of marketplace negotiation of trust versus privacy tradeoffs. For example, we are exploring the possibility of third-party certifying authorities, which would serve as brokers of trust. These certifying authorities would provide mechanisms whereby packets would be cryptographically signed with very fine-grained authentication credentials of the sender. This is not the same as having an individual digitally sign a message, as a digitally signed message may be too coarse-grained for a particular scenario and may reveal too much. Another capability might be the escrowing, by these certifying authorities, of complete identifying information for a specified period of time, to be revealed in the event that one or more of a user's packets have been identified as participating in a confirmed attack.

We are investigating the feasibility of a disciplined engineering design of Internet protocols (in the context of key policy issues) to allow optimal, fine-grained tradeoffs between traceability and anonymity to be made on the basis of specific mission requirements. Our goal is to provide an exemplar for the application of principled software and systems engineering practices in the unique context of the Internet. A key part of this process is our exploration of alternative designs for new Internet protocols that allow both the originator and the recipient of an Internet transaction to decide what levels of anonymity to accept.

### 2.3.1 Meaning of *k*-anonymity

We say that a user is *k-anonymous* in a network context if the user is only traceable to a set of measure *k*, where this could mean either a set of size *k* or a set of radius *k* in the topological

sense of the network (as shown in Figure 1). Latanya Sweeney originally defined the notion of *k*-anonymity in the privacy context for medical patient data [Sweeney 02].



Set of size *k*=10

Set of radius *k*=2
Meaning of "radius" depends on metric

*Figure 1: Examples of k-anonymity*

As a caveat, we note that a contiguous set reveals information about the set itself. For example, for a set that is contiguous with respect to university affiliation, knowing that one member belongs to the Carnegie Mellon group allows you to infer that all members of the set belong to Carnegie Mellon. On the other hand, a disjoint set may be workable, but it is nontrivial to express its properties (e.g., as a lattice, random sparse points in space, or a number of Chaumian mix-nets). An attached label may be sufficient, such as joining a "group," and cryptographic group signing could be used to identify the associated group.

### 2.3.2 An Initial Protocol for Anonymity and Traceability Tradeoffs

A naïve protocol would implement an interaction between a user and a service, as shown in Figure 2. A simple negotiation for anonymity will not work in this context, as the one-on-one negotiation reveals the identity of the user.

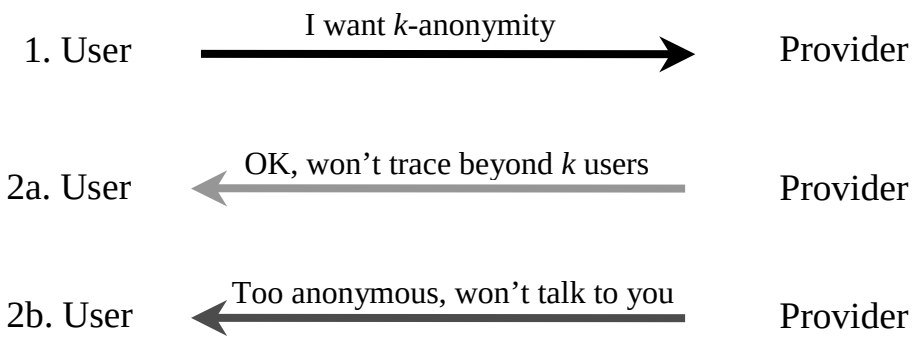



*Figure 2: Anonymity–Traceability Negotiation Using a Naïve Protocol*

### 2.3.3 User and Service Provider Goals

Making effective anonymity and traceability tradeoffs requires an understanding of the specific goals of users and service providers.

The user goals may differ on a case-by-case basis. Some examples include:

- User may want to hide its location and identity entirely (large *k*)
- User may want to hide its location somewhat (e.g., reveal the city, but not street address)
- User may want to hide its location, but not its identity

Similarly, the service providers may have different goals and/or requirements. Some examples:

- Provider may want to know both user's location and identity
- Provider may want to know user's location somewhat
- Provider may want to know user's identity, but does not care about user's location

### 2.3.4 Refining the Initial Protocol

By rethinking the process, we add an *introducer*, or *trusted third party* (TTP), to act as an intermediary in the protocol negotiation. In the examples, the TTP relays the message in the direction shown, hiding the identity of the user from the provider. The refined protocols can be seen in Figure 3 and Figure 4, and show the progressive introduction of interaction from both the user and provider perspectives.

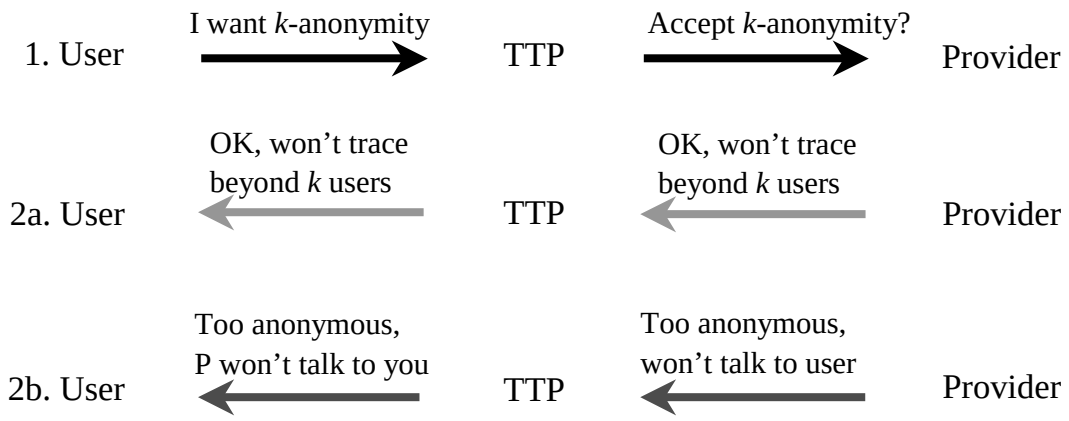



*Figure 3: Adding a Trusted Third Party to the Negotiation Protocol*

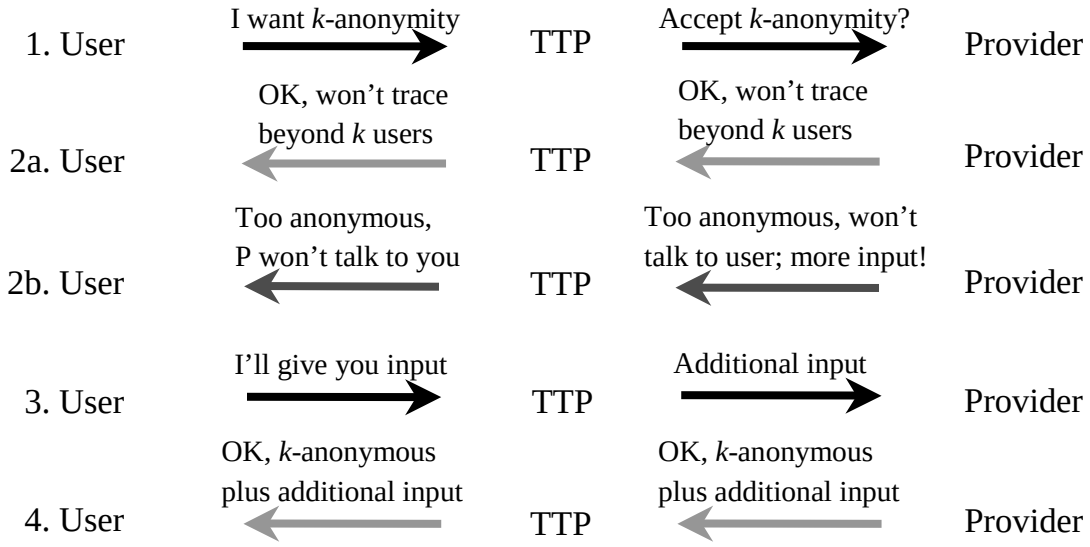



*Figure 4: Refined Protocol with Additional Negotiation Options*

The refined protocol shown in Figure 4 allows for more sophisticated negotiations of anonymity and traceability tradeoffs, on a case-by-case basis. Further development and refinement of Internet protocols that support fine-grained negotiations of tradeoffs between anonymity and traceability (on a mission-by-mission basis) requires a disciplined engineering

approach involving requirements elicitation and refinement, development of protocol specifications, and validation of the protocol specifications (through modeling, prototyping, or other means).

## 2.4 Collaborators

The principal investigators for the LEVANT project are Howard Lipson and Sven Dietrich, who are both members of the CERT® Coordination Center (CERT/CC) at the SEI. The project team also includes Ashish Shah, a doctoral student at the Department of Engineering and Public Policy at Carnegie Mellon University.

In addition to the SEI IR&D funding, Howard Lipson and Sven Dietrich have been awarded two consecutive Carnegie Mellon CyLab "seed grants" (sponsored by the Army Research Office) for the LEVANT project. The first award provided support for our doctoral student during the past academic year, and the second award will provide continued doctoral student support for the 2004–05 academic year.

## 2.5 Evaluation Criteria

We have proposed the following long-term success criteria for this project:

- Our initial engineering design of new Internet protocols (that permit fine-grained anonymity and traceability tradeoffs) sheds new light on how to improve the survivability of Internet-based mission-critical systems.
- The proposed study leads to the discovery of significant insights into how traditional engineering tools, techniques, and processes can be effectively applied in the context of the Internet environment and the stringent security and survivability requirements of critical infrastructures and other mission-critical systems.
- Our research papers are accepted in peer-reviewed venues (e.g., conferences and journals in software engineering, privacy, security, and survivability).
- An ultimate long-term goal would be the adoption of our new protocols by Internet standards bodies, along with the broad recognition of the need to promulgate and adopt the engineering techniques used to produce them.

## 2.6 Results

The LEVANT project feasibility study is a work in progress that will be completed during FY2005. Other urgent CERT/CC business allowed us significantly less time during FY2004 than we had hoped to devote to the LEVANT project. Nonetheless, we are encouraged by our progress in this challenging research area, and we have been awarded a second CyLab grant

for the LEVANT project that will fund our doctoral student to work on the project for an additional year (September 2004–August 2005).

We have been working to establish a solid theoretical foundation on which to base principled engineering tradeoffs between traceability and anonymity. Progress includes an extensive examination of the research literature and work on a new conceptual model that helps clarify the relationships between anonymity and traceability. We expect the model to continue to evolve into a foundation for understanding and expressing the full range of engineering requirements for the design of Internet protocols that support attribute tradeoffs and negotiations, as well as help us to generate examples of specific user requirements for anonymity and traceability that must be satisfied for particular applications, systems, or missions. A primary goal of our conceptual model is to help us better delineate the space of possible tradeoffs between traceability and anonymity, and to evaluate the feasibility of designing general-purpose Internet protocols that implement the largest possible range of anonymity–traceability tradeoffs. One of the key engineering requirements for the design of such protocols is that they effectively support anonymity–traceability tradeoff negotiations between service providers and their clients.

In order to better understand the anonymity–traceability tradeoffs that Internet protocols should support, we have done an extensive review of the research literature that exists in the anonymity and traceability space. We have also examined exceptionally relevant research on trust negotiation, contract signing, oblivious transfer and privacy-preserving data mining. Our literature search has included interoperable strategies in automated trust negotiation, protocols that facilitate multiparty computations, protocols for signing contracts, and social matching algorithms.

Finally, we have surveyed this research area from an economic and public policy perspective. We have tried to better understand the economic aspects of personal privacy and the economics of anonymity and traceability. We have looked at the economic incentives and disincentives for service providers to support various anonymity and traceability services.

### 2.6.1 Case Study in Anonymity–Traceability Tradeoffs: A Comparative Analysis of Privacy Implications of Transit Card Systems

A case study carried out by our doctoral student, Ashish Shah (with initial results submitted as a class project report), surveyed the transit card architectures deployed in Washington, D.C., New York City, and Hong Kong. Dependence on public transportation in the United States and other parts of the world is expanding, with significant investment underway to modernize and automate current fare-collection systems.

Transit cards often allow travelers to access multiple modes of transportation with a single card, regardless of whether the transportation is administered by one agency or by multiple agencies within a region. The card issuers also provide mechanisms for travelers to replace lost cards. In addition, transit card advocates say, these systems are easy to use and the cards could be used for other applications, such as payments at retail stores and parking lots. Because they are potentially linkable to individual travelers and to past trips, transit cards raise a number of privacy concerns not raised by cash or token systems. These include the ability of transportation authorities and other parties to track commuters and maintain a profile of their travel habits. This information has value to law-enforcement agencies as well as to marketers. It is not clear whether travelers are aware of the privacy risks associated with the transit cards, or how they feel about the tradeoff of privacy for convenience. Typically, individual travelers have limited ability to take steps to protect their own privacy if they want to use these systems. Therefore, the decisions that are made about how these systems are implemented and administered are central to the degree of privacy the system affords. The purpose of the case study is to identify and discuss the privacy issues that arise due to the deployment of transit card systems and to identify design alternatives and tradeoffs that might be considered to reduce privacy risks.

## 2.7 Publications and Presentations

We are in the process of writing a technical report that describes the initial research results of the LEVANT project. This report will describe our conceptual model for anonymity and traceability tradeoffs, and will also specify the engineering requirements for a general-purpose Internet negotiation protocol that supports anonymity–traceability tradeoffs, based on user and service-provider preferences that are specified at the time of a transaction.

Howard Lipson and Sven Dietrich have made presentations on the LEVANT project to CyLab industrial affiliates, the Army Research Office, and other potential sponsors. Ashish Shah represented the LEVANT project at a poster session at a CyLab Corporate Partners Conference.

## 2.8 FY2005 Tasks

The LEVANT IR&D project tasks for FY2005 include publication of one or more technical reports or papers, along with additional research to further develop and refine the conceptual model upon which the tradeoff negotiation protocol will be based. We will also explore in greater depth several of the economic and public policy issues relevant to this research area.

# 3 Architecture-Based Self-Adapting Systems

**Rick Kazman**

## 3.1 Purpose

An increasingly important requirement for software-based systems is the ability to adapt themselves at runtime to handle such things as resource variability, changing user needs, changing demands, and system faults. In the past, systems that supported such self-adaptation and self-repair were rare, confined mostly to domains like telecommunications switches or deep space control software, where taking a system down for upgrades was not an option, and where human intervention was not always feasible. However, today more and more systems have this requirement, including e-commerce systems and mobile embedded systems.

For systems to adapt themselves, one of the essential ingredients is self-reflection: a system must know what its architecture is, and what its status is, and it must be able to identify opportunities for improving its own behavior. For most complex systems it is crucial to have a well-defined architecture. Such a definition provides a high-level view of a system in terms of its principal runtime components (e.g., clients, servers, databases), their interactions (e.g., RPC, event multicast), and their properties (e.g., throughputs, reliabilities). As an abstract representation of a system, an architecture permits many forms of high-level inspection and analysis. Consequently, over the past decade considerable research and development has gone into the development of notations, tools, and methods to support architectural design.

Despite advances in developing an engineering basis for software architectures, a persisting difficult problem is determining whether a system as implemented has the same architecture as was originally designed. Without some form of consistency checking that guarantees the relationship between an architecture and the system as implemented, the benefits of the architecture will be hypothetical at best, invalidating the primary purpose of the architectural design.

However, there are a number of hard technical challenges in bridging the gap between the as-designed and the as-implemented architecture. The most serious challenge is finding mechanisms to bridge the abstraction gap: in general, low-level system observations do not map directly to architectural actions. For example, the creation of an architectural connector might

involve many low-level steps, and those actions might be interleaved with many other architecturally relevant actions. Moreover, there is likely no single architectural interpretation that will apply to all systems: different systems will use different runtime patterns to achieve the same architectural effect, and conversely, there are many possible architectural elements to which one might map the same low-level events. In this work we have created a technique to solve the problem of dynamic architectural discovery for a large class of systems. The key idea is to provide a framework that allows one to map implementation styles to architecture styles. In this way we can provide a sound basis for discovering and reasoning about an architecture, which is the necessary prerequisite to self-adaptation.

## 3.2 Background

Currently two principal techniques have been used to determine or enforce relationships between a system's architecture and its implementation. The first is to ensure consistency by construction. This can be done by embedding architectural constructs in an implementation language (e.g., see Aldrich) where program analysis tools can check for conformance [Aldrich 02]. Or, it can be done through code generation, using tools to create an implementation from a more abstract architectural definition [Shaw 95, Taylor 96, Vestal 96]. While effective when it can be applied, this technique has limited applicability. In particular, it can usually only be applied in situations where engineers are required to use specific architecture-based development tools, languages, and implementation strategies. For systems that are composed out of existing parts, or that require a style of architecture or implementation outside those supported by generation tools, this approach does not apply.

The second technique is to ensure conformance by extracting an architecture from a system's code, using static code analysis [Jackson 99, Kazman 99, Murphy 95]. When an implementation is sufficiently constrained so that modularization and coding patterns can be identified with architectural elements, this can work well. Unfortunately, however, the technique is limited by an inherent mismatch between static, code-based structures (such as classes and packages), and the run-time structures that are the essence of most architectural descriptions [Garlan 01]. In particular, the actual runtime structures may not even be known until the program runs: clients and servers may come and go dynamically; components (e.g., DLLs) not under direct control of the implementers may be dynamically loaded, etc.

A third, relatively unexplored, technique is to determine the architecture of a system by examining its behavior at runtime. The key idea is that a system can be monitored while it is running. Observations about its behavior can then be used to infer its dynamic architecture. This approach has the advantage that, in principle, it applies to any system that can be monitored, it gives an accurate image of what is actually going on in the real system, it can accommodate systems whose architecture changes dynamically, and it imposes no *a priori* restrictions on system implementation or architectural style.

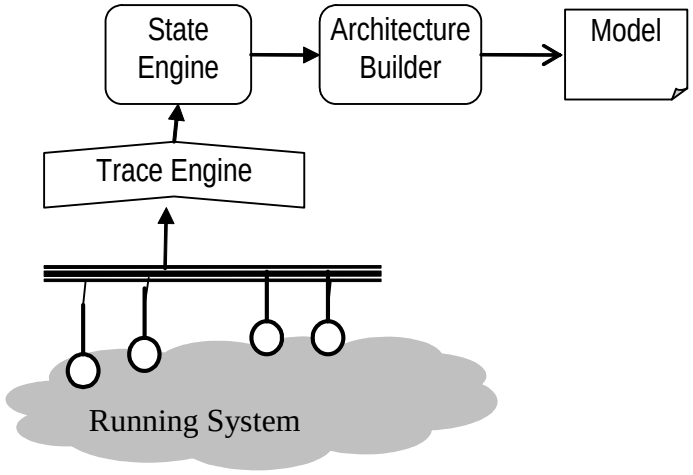



*Figure 5: The DiscoTect Architecture*

## 3.3 Approach

To address these concerns, we have developed a novel approach for extracting a system's architecture at runtime, without perturbing the system. To test this approach we have built the DiscoTect system, illustrated in Figure 5.

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

To handle the variability of implementation strategies and possible architectural styles of interest, we provide a language to define new mappings. Given a set of implementation conventions (which we will refer to as an implementation style) and a vocabulary of architectural element types and operations (which we will refer to as an architectural style), we provide a description that captures the way in which runtime events should be interpreted as operations on elements of the architectural style. Thus each pair of implementation style and architectural style has its own mapping. A significant consequence is that these mappings can be reused across programs that are implemented in the same style.

## 3.4 Collaborators

The collaborators on this research in 2003–04 were:

- Rick Kazman (SEI)
- David Garlan (Carnegie Mellon University School of Computer Science (SCS) faculty; self-supported)
- Bradley Schmerl (Carnegie Mellon SCS systems scientist; partially supported by the IR&D)
- Hong Yan (Carnegie Mellon SCS; supported by the IR&D)
- Jonathan Aldrich (Carnegie Mellon SCS; self-supported)

## 3.5 Evaluation Criteria

We have set ourselves four evaluation criteria for this work. These are

- at least one commercial or government system analyzed
- at least one journal or conference paper published on this research
- at least one technical report or technical note published on this approach
- clear guidance provided on the feasibility of the approach for future SEI investment and involvement

We believe that we have met all of these criteria, as we will describe next.

## 3.6 Results

During the past year, we have achieved a number of significant results. We have built an initial version of DiscoTect and have used it to analyze systems in three different architectural styles [Yan 04a, Yan 04b]. The most significant of these systems was implemented in Sun's Java 2 Enterprise Edition (J2EE), a commercial architectural framework [J2EE]. In each of the three systems that we analyzed we were able to discover important facts about the architectures that were hitherto unknown. These facts were typically mismatches between the as-documented and the as-implemented architecture. The discovery of these mismatches illustrates the power of the DiscoTect approach.

There are a number of advantages to the DiscoTect approach. First, it can be applied to any system that can be monitored at runtime. In our case, we have done three case studies on systems written in Java and we have done our runtime monitoring using AspectJ. We have recently experimented successfully with the use of AspectC to extract runtime information

from C and C++ programs. Second, we do not require any change to the system to allow it to be monitoring. All of the monitoring code is contained in the aspects. Third, by simply substituting one mapping description for another, it is possible to accommodate different implementation conventions for the same architectural style, or conversely to accommodate different architectural styles for the same implementation conventions. This means that DiscoTect mappings will be highly reusable.

Building on these results, we have embarked upon an effort to try to make DiscoTect more theoretically sound, more usable, and more easily transitionable to government and industry. We have formulated a set of recommendations for future SEI investment and involvement based on this effort.

## 3.7 Publications and Presentations

We published one paper and made a presentation to the International Conference on Software Engineering [Yan 04a] and we have published one SEI technical report [Yan 04b] on DiscoTect in 2003–04.

# 4 Eliciting and Analyzing Quality Requirements: A Feasibility Study

**Nancy Mead, Carol Woody, Donald Firesmith**

## 4.1 Purpose

### 4.1.1 The Need for this Feasibility Study

The vision of the SEI is "the **right** software, delivered **defect free**, on time and on cost, every time." The mission of the SEI is: "The SEI provides the technical leadership to advance the practice of software engineering so the DoD can acquire and sustain its software-intensive systems with predictable and improved **cost**, **schedule**, and **quality**."

Proper requirements engineering is critical to the fulfillment of both the SEI's vision and mission because the requirements determine

- what the **right** software is (i.e., requirements specify the right software, and requirements validation determines if the right software was built)
- what a **defect** is (i.e., a failure to meet the requirements)
- the minimum **cost** and **schedule** of the acquisition
- the standard against which **quality** is measured

### 4.1.2 The Criticality of Requirements Engineering to Our Clients

It is well recognized in the industry that requirements engineering is critical to the success of any major development project [Boehm 88, Willis 98]:

- As compared with defects found during requirements evaluations, defects cost
  - 10-200 times as much to correct once fielded
  - 10 times as much to correct during testing
- Reworking requirements defects on most software development projects costs 40-80% of the effort.

- The percentage of defects originating during requirements engineering is estimated at 42-64%.
- The total percentage of project budget due to requirements defects is 25-40%.
- A recent study by IBM's System Sciences Institute found that the relative cost of fixing software defects after deployment is almost 15 times greater than detecting and eliminating them in development, and Gartner estimates the cost of mitigating poor security is twice as great as the cost of prevention. The National Institute of Standards and Technology (NIST) reports that software that is faulty in the areas of security and reliability costs the economy $59.5 billion annually in breakdowns and repairs. The costs of poor security requirements show that there would be a high value to even a small improvement in this area.
- By the time an application is fielded in its operational environment, it is very difficult and expensive to significantly improve its security.
- Requirements problems are the number one cause of why projects
  - are significantly over budget
  - are significantly past schedule
  - have significantly reduced scope
  - deliver poor-quality applications
  - are not significantly used once delivered
  - are cancelled

## 4.2 Background

### 4.2.1 Previous SEI Work

The various initiatives and programs at the SEI deal with requirements engineering only to the extent required for their needs. Thus, there is no complete or integrated approach to requirements engineering at the SEI. This IR&D feasibility study draws on the following previous work regarding requirements engineering:

- The initial part of the SEI Architecture Tradeoff Analysis Method (ATAM) assumes that adequate quality requirements have not been specified, and provides guidance on informally identifying representatives of the most critical types of quality requirements so that major risks to and defects in the architecture can be identified. However, ATAM does not address all quality requirements, nor does it address how to formally analyze and specify them. Also, security and safety are not as central to ATAM as other quality requirements such as performance, availability, and modifiability.
- SEI Quality Attribute Workshops (QAWs) provide a way to identify the most critical quality requirements at the highest level of abstraction for the purpose of producing architecture test cases (scenarios) in support of architecture evaluations. As with ATAMs,

QAWs do not address all quality requirements nor do they address how to formally analyze and specify them.

- In the early 1990s the SEI did some work in requirements engineering and analysis that can provide some background material to this study. However, this work is somewhat dated and did not address the same problems that are covered in this feasibility study.
- The SEI developed a transition package for requirements management, but this was general in nature.
- The SEI developed and delivered a train-the-trainer course on requirements engineering in the early 1990s, and also developed a requirements engineering course for Carnegie Mellon University's Master of Software Engineering. Both courses are available on videotape from the SEI.
- Good operational practices used in the SEI Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) Method have been applied in an independent technical assessment to "health check" a software development effort. This catalog of practices was assembled from a broad range of sources, including the CERT Coordination Center (CERT/CC) at the SEI, British Standards Institution (BSI), and NIST. To address the quality requirement of security, the needs of the deployment environment must be considered, and the OCTAVE catalog of practices provides a good starting point.
- The e-Authentication Risk Assessment (e-RA) tool developed for the federal Electronic Government (e-gov) effort focused on authentication requirements, a subset of security requirements.
- Survivable systems analysis (SSA), developed in the SEI Networked Systems Survivabilty (NSS) program has addressed client problems in architecture and requirements.
- Commercial off-the-shelf software (COTS) and product line work at the SEI addresses tradeoffs between application requirements and the capabilities of COTS packages. The Vendor Rating and Threat Evaluation (V-RATE) work in the NSS program addresses the security risks of COTS software.

### 4.2.2 Previous Industry and Government Work

- ISAlliance issued the *Common Sense Guide for Senior Managers*, which established the top 10 recommended information security practices (see <www.isalliance.org>). Practice number 4 in this list calls for design of an enterprise-wide security architecture based on satisfying business objectives, which influences security requirements and is impacted by new development efforts.
- Industry contacts from ISAlliance (e.g., Nortel and Raytheon) have expressed interest in improved quality requirements elicitation and analysis.

## 4.3 Approach

While much has been written about quality requirements in the abstract, the mechanisms to translate the theory into practice have been unclear. If quality requirements are not effectively defined, the resulting system cannot be effectively evaluated for success or failure prior to implementation. Quality requirements are often missing in the requirements elicitation process. The lack of validated methods is considered one of the factors.

In addition to employing applicable software engineering techniques, the organization must understand how to incorporate the techniques into its existing software development processes. The identification of organizational mechanisms that promote or inhibit the adoption of quality requirements elicitation can be an indicator of the quality level of the resulting product.

Our literature search, focusing on safety and security software requirements engineering, provided an interesting range of material. Safety requirements engineering is a much more mature area with broad planning content, standards frequently focused on software for specific domains, and case studies that range over several years. Security requirements engineering has only been recently researched with much of the material assembled within the current year. Neither area had techniques or templates for requirements elicitation. By assembling an elicitation framework based on our initial research and applying it to a software development effort, we were able to identify additional research areas and refine the framework through further research.

If usable approaches to safety and security are developed and mechanisms to promote organizational use are identified, then software quality with respect to safety and security can be effectively characterized by an organization to assure the resulting product effectively meets these requirements.

An elicitation process for security requirements was developed and applied in a client case study. The case study results will be sanitized and published at a later time. The draft process is shown in Table 1.

*Table 1: Security Requirements Elicitation and Analysis Process*

| Number | Step | Input | Techniques | Participants | Output |
|---|---|---|---|---|---|
| 1 | Agree on definitions | Candidate definitions from IEEE and other standards | Structured interviews, focus group | Stakeholders, requirements team | Agreed-to definitions |
| 2 | Identify safety and security goals | Definitions, candidate goals, business drivers, policies and procedures, examples | Facilitated work session, surveys, interviews | Stakeholders, requirements engineer | Goals |
| 3 | Select elicitation techniques | Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of safety and security needed, cost benefit analysis, etc. | Work session | Requirements engineer | Selected elicitation techniques |
| 4 | Develop artifacts to support elicitation technique | Selected techniques, potential artifacts (e.g., scenarios, misuse cases, templates, forms) | Work session | Requirements engineer | Needed artifacts: scenarios, misuse cases, models, templates, forms |
| 5 | Elicit safety and security requirements | Artifacts, selected techniques | Joint Application Design (JAD), interviews, surveys, model-based analysis, safety analysis, checklists, lists of reusable requirements types, document reviews | Stakeholders facilitated by requirements engineer | Initial cut at safety and security requirements |
| 6 | Categorize requirements as to level (system, software, etc.) and whether they are requirements or other kinds of constraints | Initial requirements, architecture | Work session using a standard set of categories | Requirements engineer, other specialists as needed | Categorized requirements |
| 7 | Perform risk assessment | Categorized requirements, target operational environment | Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including hazard/threat analysis; Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) | Requirements engineer, risk expert, stakeholders | Risk assessment results, added mitigation requirements to bring exposure into acceptable level |

| | | | | | |
|---|---|---|---|---|---|
| 8 | Prioritize requirements | Categorized requirements and risk assessment results | Prioritization methods, such as Triage, Win-Win | Stakeholders facilitated by requirements engineer | Prioritized requirements |
| 9 | Requirements inspection | Prioritized requirements, candidate formal inspection technique | Inspection method, such as Fagan, peer reviews | Inspection team | Initial selected requirements, documentation of decision-making process and rationale |

### 4.3.1 Safety Requirements

The taxonomy of safety-related requirements illustrated in Figure 6 was developed. As illustrated in Table 2, this resulted in a set of generic reusable templates for engineering safety requirements. This taxonomy became a foundation for the development of a process for engineering safety requirements (see Figure 7), because the process depends on the type of safety requirements being engineered.

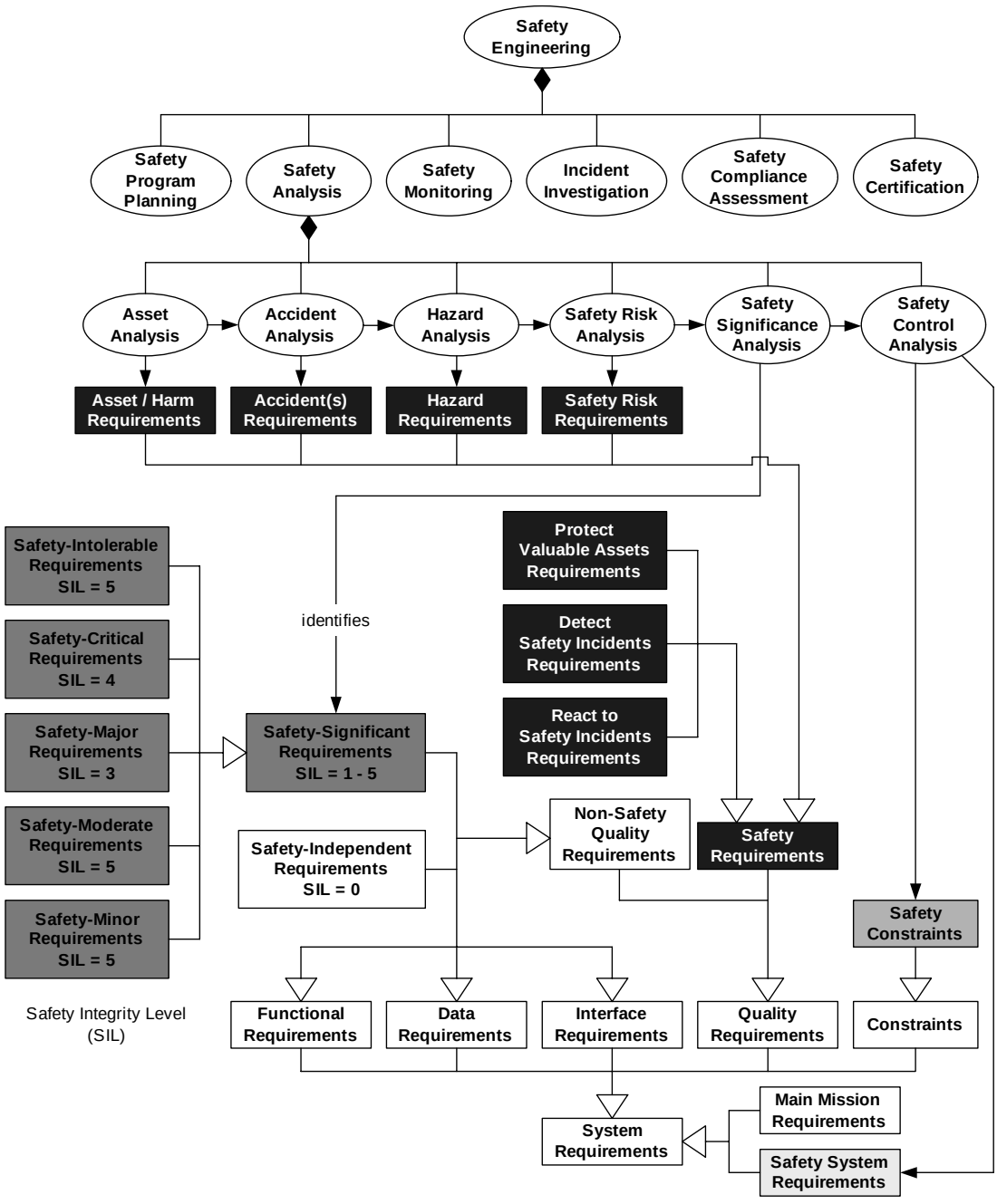



*Figure 6: Taxonomy of Safety-Related Requirements*

*Table 2: Reusable Templates for Safety Requirements*

| **Type of Safety Requirement** | **Form of Parameterized Requirement** |
|---|---|
| Prevention of Accidental Harm to Valuable Asset | The system shall not [cause \| permit to occur] [amount of a type of harm to a type of asset] more than [a threshold of measurement]. |
| Prevention of Safety Incidents (esp. Accidents) | The system shall not [cause \| permit to occur] [optional: accident severity] [type of safety incident] more than [a threshold of measurement]. |
| Prevention of Hazards | The system shall not [cause \| permit to occur] [type of hazard] more than [a threshold of measurement]. |
| Prevention of Safety Risk | The system shall not [cause \| permit to occur] a [harm severity category] [accident \| hazard] with likelihood greater than [probability \| accident likelihood category].<br>No credible system [accident \| hazard] shall represent a [threshold of measurement = safety risk category] safety risk. |
| Detection of Violation of Prevention | The system shall detect [accidental harm \| safety incident \| hazard \| safety risk]. |
| Reaction to Violation of Prevention | When the system detects [accidental harm \| safety incident \| hazard \| safety risk], then the system shall [list of system actions]. |

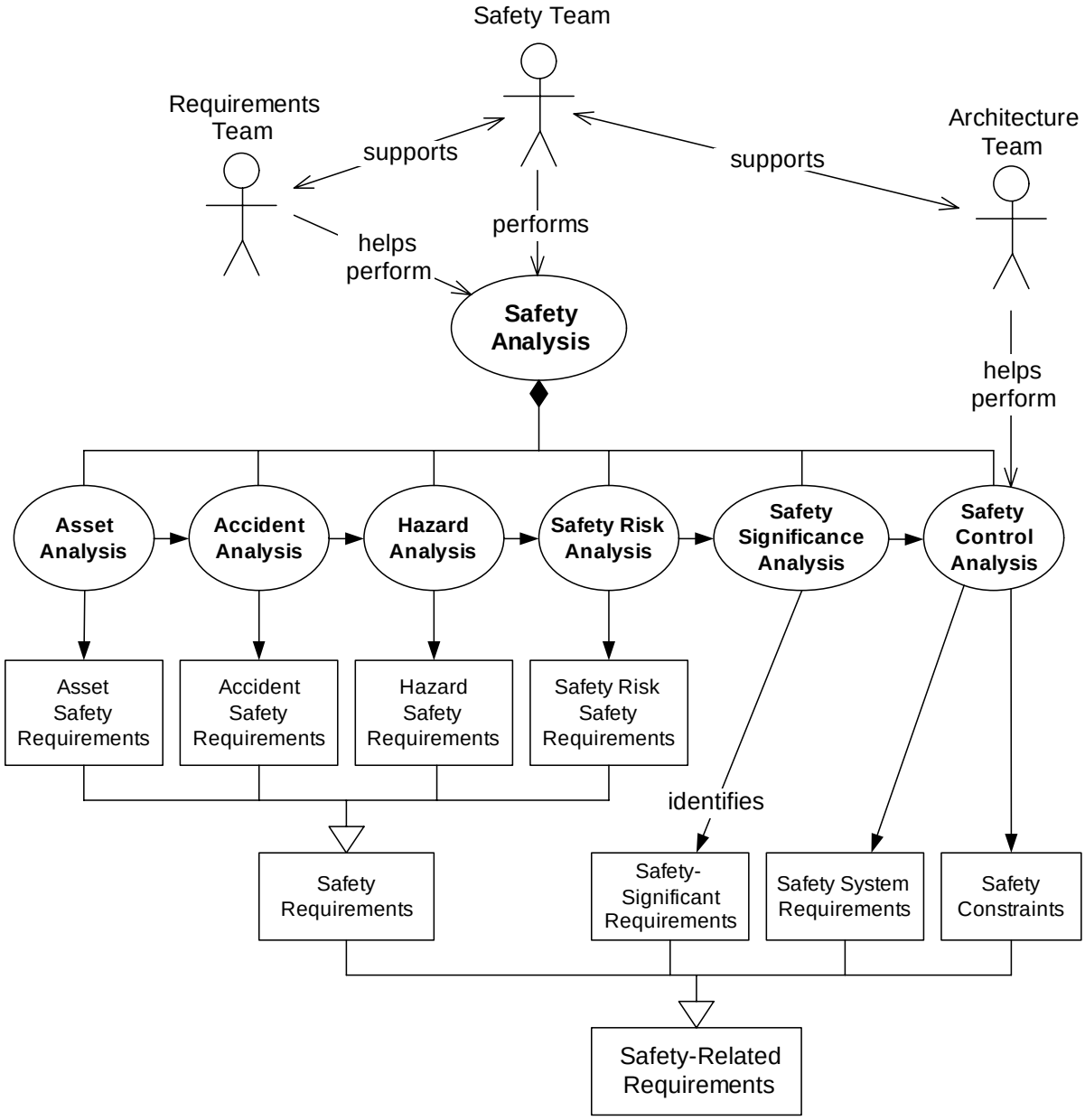



*Figure 7: Top-Level Process for Identification and Analysis of Safety-Related Requirements*

## 4.4 Collaborators

The primary SEI team members were Don Firesmith (Acquisition Support Program), Nancy Mead (Networked Systems Survivability [NSS] initiative), and Carol Woody (NSS). The following professionals agreed to be external collaborators:

- Dr. Natalia Juristo (Technical University of Madrid)

- Dr. Robyn R. Lutz (Jet Propulsion Laboratory)
- Mr. C.S. (Sekar) Chandersekaran (IDA)

The external collaborators provided advice and review of the work. They provided their own support.

## 4.5 Evaluation Criteria

A large case study project was undertaken with an industry client to test the validity of the proposed security requirements elicitation process. Seven Carnegie Mellon University graduate students worked on this project during the summer of 2004. Case study results were evaluated initially by the IR&D investigators for consistency and effectiveness. Controversial questions were shared with the external collaborators, other SEI staff members, and an external requirements engineering newsgroup for additional input.

Management issues were presented in various conferences and shared with collaborators for feedback and expansion.

## 4.6 Results

Draft processes for safety and security requirements were developed, as noted above. Management issues were surfaced and investigated. Although much work remains to be done, we believe that this IR&D project advanced the state of the art in the area of security and safety requirements. We would hope to see refinement of the draft processes and confirmation with additional practical case studies. Prototype tools were also developed to support some aspects of the processes.

There is much confusion surrounding terminology and no agreed-upon standard definitions. There is a long list of terms (e.g., asset, harm, vulnerability, threat, risk, and impact) that carry different meanings depending on the perspective of the user, which can vary by job role, organizational level, organizational domain, development life-cycle stage, and other factors. The lack of an agreed-upon set of definitions can derail any effort to identify and agree on requirements. Given the contextual nature of the definitions, the elicitation process must first establish appropriate definitions for use of the process within an organization.

There are standards for levels of safety, many of which are domain specific, but these can be measured at some level within a specific context. There is great disagreement on standards for levels of security. Security measurement research has been heavily focused in the operational arena with limited applicability to development. Risk is considered a factor in both

quality areas, but the application differs. It is unclear if the difference between security and safety levels is valid or the result of research limitations; further research is recommended.

The IEEE Standard for Software Safety Plans (IEEE Std 1228-1994) provides a framework for inserting safety considerations into the software development life cycle. A standard framework for addressing security across the SDLC does not exist. A possibility has been suggested by Gary McGraw of applying best practices in security requirements (including abuse cases), risk analysis, static analysis tools, and penetration testing [McGraw 04]. A panel discussion at the European Software Engineering Process Group conference (listed in the Publications and Presentations section, focused on the ability of current development methods to produce a secure product and identified critical shortcomings of current SDLC approaches. A case study incorporating the proposed requirements elicitation framework along with considerations for operational security earlier in the life cycle has been initiated to further analyze these ideas.

Early in the literature review, the potential conflict of quality efforts (perceived as time-consuming) with organizational management direction (driven by time-to-market and cost considerations) was identified. The importance of researching this conflict as a potential barrier to adoption of improved elicitation techniques was based on consistency with issues identified in the use of risk methodologies, such as OCTAVE for security considerations, within the operational system environment. Also, issues identified in independent technical assessments of major government software development projects supported concern about conflicting organizational priorities. A portion of the feasibility effort was focused on the identification of appropriate organizational behavior to foster the use of effective mechanisms for eliciting and analyzing quality requirements.

The management areas for consideration included

- recognition of organizational risk
- creating a risk-aware environment
- establishing organizational support for quality

## 4.7 Publications and Presentations

The following recent publications were used:

- Graff, M. & van Wyk, K. *Secure Coding Principles & Practices*. O'Reilly, 2003.
- Hoglund, G. & McGraw, G. *Exploiting Software: How to Break Code*. Addison-Wesley, 2004.

During the course of the IR&D project, several reports were published by the investigators:

- Firesmith, Donald G. "A Taxonomy of Safety-Related Requirements." *Proceedings of Requirements Engineering, 2004 Requirements for High Assurance Systems (RHAS) Workshop*. Kyoto, Japan. IEEE Computer Society, Washington, D.C., 6 September 2004.
- Mead, Nancy R. *Requirements Engineering for Survivable Systems* (CMU/SEI-2003-TN-013). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University (2003).

Carol Woody, one of the IR&D investigators, participated in the following panel discussion, presentation, and workshop. These activities were intended to present and initiate further discussion on the management issues:

- "Considering Operational Security Risks During Systems Development," Software Engineering Process Group Conference (SEPG 2004), Orlando, FL, March 9, 2004.
- "Can Secure Systems be Built Using Today's Development Processes," European SEPG, London, England, June 17, 2004. Summary <http://www.cert.org/archive/pdf/eursepg04.pdf>. Panel presentation <http://www.cert.org/archive/pdf/EurSEPG405d.pdf>.
- "Considering Security Risks During the System Development Life Cycle," Liberty University, Lynchburg, VA, August 6-8, 2004.

A workshop on Requirements for High Assurance Systems (RHAS04) was held in conjunction with the International Conference on Requirements Engineering on September 6, 2004. The workshop co-chairs were Donald Firesmith and Nancy Mead, two of the IR&D investigators. The focus of the workshop was safety requirements. Workshop proceedings were published by the SEI. The papers will also be available on the RHAS Web site after the conference: www.sei.cmu.edu/community/rhas-workshop/.

# 5 Enabling Technology Transition Using Six Sigma

**Jeannine Siviy, Eileen Forrester**

## 5.1 Purpose

The primary purpose of this project was to examine whether Six Sigma, when used in combination with another process improvement technology or technologies, makes the transition[1] of that technology more effective. While we were interested first in the transition of SEI technologies, what we learned also applies to non-SEI technologies, non-process technologies, and to Six Sigma users in general.

A secondary or implicit purpose of this project was to overcome a frequent misunderstanding about Six Sigma. We realized that many practitioners do not get the transition advantage they could from Six Sigma because they see it as a competitor with other improvement practices and models, rather than as an enabler, and therefore do not recognize its potential.

## 5.2 Background

Six Sigma is an approach to business improvement that includes a philosophy, a set of metrics, an improvement framework, and a toolkit of analytical methods. Its philosophy is to improve customer satisfaction by eliminating and preventing defects, resulting in increased profitability. Sigma (σ) is the Greek symbol used to represent standard deviation, or the amount of variation in a process. The measure *six sigma* (6σ), from which the overall approach derives its name, refers to a measure of process variation (six standard deviations) that translates into an error or defect rate of 3.4 parts per million, or 99.9997 percent defect free. In the Six Sigma approach, *defects* are defined as anything in a product or service, or any process variation, that prevents the needs of the customer from being met.

[1] By "transition" we mean all of the following: adaptation and introduction of technology by developers or champions, implementation of technology by organizations and change agents, and adoption of technology by its intended users. Most of our research results from this project center on implementation and adoption, but those results have profound implications for developers and champions of technology.

During the 1990s, large manufacturing companies such as Motorola, General Electric, and Allied Signal used Six Sigma processes to collect data, improve quality, lower costs, and virtually eliminate defects in fielded products. Using both statistical and non-statistical methods, the approach soon spread to several major service industries, and today software practitioners are exploring ways to apply Six Sigma techniques to improve software and systems development.

## 5.3 Approach

This project was conducted by two members of the SEI: Jeannine Siviy of the Software Engineering Measurement and Analysis initiative, and Eileen Forrester of the Technology Transition Practices and Capability Maturity Model Integration (CMMI) teams. Siviy is a "Black Belt Practitioner"[2] in Six Sigma, and Forrester is an applied researcher and consultant in technology transition. In learning about each other's domains we concluded that Six Sigma and technology transition are probably mutually reinforcing. In addition, we hypothesized that the successful transition of Six Sigma itself meant that a persistent community of practitioners, possibly more adept at adopting technology, exists and even pervades the software and systems community we serve. We wanted to discover if the community of Six Sigma practitioners might be more receptive to and capable of adopting SEI technologies and other technologies, thereby serving as a *de facto* community of innovators and early adopters.

Our approach was to use a combination of case interviews and site visits, surveys, field experience, discussions with technology and domain experts, and literature review to examine the use of Six Sigma as a transition enabler and to understand how organizations are using Six Sigma with other technologies. Both the surveys and the literature review were used to elicit hypotheses and candidate case studies. We also elicited hypotheses through interviews and discussions with other researchers and practitioners (see the Collaborators section for participants).

We used an inductive technique to generate our hypotheses, to be tested with qualitative field experience, akin to the "grounded theory" method. Put simply, *grounded theory* calls for the investigators to pose an explicit theory, observe field conditions to test the theory, revise the theory based on that experience, observe again with the revised theory, and so forth. It is a qualitative technique driven by data. We maintained a set of position statements throughout the project that reflected our current theories, and, as the project advanced, added inferences

[2] From isixsigma.com: Black Belts are the Six Sigma team leaders responsible for implementing process improvement projects within the business—to increase customer satisfaction levels and business productivity. Black Belts are knowledgeable and skilled in the use of the Six Sigma methodology and tools. They have typically completed four weeks of Six Sigma training, and have demonstrated mastery of the subject matter through the completion of project(s) and an exam. They coach Green Belts and receive coaching and support from Master Black Belts.

and findings. We periodically reviewed these with other researchers and practitioners for feedback, suggestions, and critique.

## 5.4 Collaborators

We included about a dozen members of SEI programs (including Software Engineering Process Management, Dynamic Systems, Product Line Systems, and Survivable Systems) as we performed the research, both to get the benefit of their expertise and to foster natural information flows for the project results in other programs. We held interviews and discussions with some to elicit and review our position statements as we progressed. With others, we collaborated to consider how Six Sigma is applied to their domains, or how it could be applied.

Four SEI affiliates participated in scoping this project, setting the research direction and refining survey questions and interview process: Lynn Penn, Lockheed Martin IS&S; Bob Stoddard, Motorola; and Dave Hallowell and Gary Gack, Six Sigma Advantage. We appreciate their contribution of time and multiple trips to Pittsburgh.

We engaged a small set of practitioners who are thinking about Six Sigma and architecture to improve our potential direction in that domain. These included John Vu of Boeing, Lynn Carter of Carnegie Mellon West, Dave Hallowell of Six Sigma Advantage, and Bob Stoddard of Motorola.

Two members of the Information Technology Process Institute, Gene Kim and Kevin Behr, reviewed our ideas for studying the IT operations and security domain.

The International Society of Six Sigma Professionals and isixsigma.com collaborated with us to formulate and distribute the surveys, and their memberships are interested to hear the results of the project.

Additionally, we would like to acknowledge the time investment of the interviewees and survey respondents whose thoughtful responses provided a rich data set. We estimate their time investment to exceed 150 hours.

## 5.5 Evaluation Criteria

This project was conceived as a feasibility study. SEI feasibility studies are undertaken to examine the technical and transition issues associated with a promising technology or significant engineering problem. These studies allow the SEI to consider a long-range direction for new work and an assessment of what issues should be addressed if the new work is to be pursued.

The criteria for evaluating feasibility studies are as follows:

- What is the problem in the practice of software or systems engineering or barrier that impedes adoption of new practices?
- Is this unique or critical to software or systems engineering?
- What is the technical or business case for recommending further work by the SEI?

We posited that Six Sigma is a unique *opportunity* or *enabler* (rather than a barrier) that could have the potential to improve transition generally for many software- and systems-related technologies. If we were to express this as a barrier, then the barrier we saw was that too few organizations were aware of—let alone taking advantage of—the possible enabler. Only a few innovative top performers are using Six Sigma as an enabler and it is occasionally in use serendipitously instead of strategically. The enabler is not unique to software and systems engineering but is critical and significant to that domain and to related domains on which software and systems engineering are dependent (administrative science and IT operations, for example).

### 5.5.1 Evaluation of Project Data

The project data consisted primarily of language data: text from case study interviews, survey responses, publications, and presentations. Surveys also yielded a limited amount of numerical and countable attribute data, most of which served to characterize the context of and demographics associated with the responses.

The language data was processed qualitatively; each portion of text was evaluated against a list of hypotheses created at the start of the project. This list was revised, as appropriate, using field and case study data.

Minimally, a hypothesis required one credible example of its application to be deemed "feasible." The example had to come from an organization that was at least progressing with one or more variants of Six Sigma and one or more of the improvement models and practices under study.

## 5.6 Results

Our findings in examining the efficacy of combining Six Sigma with other technologies to get more effective transition are so clear that the technical and business case for further work is simple. Given the field results when SEI technologies and other technologies are applied in concert with Six Sigma, and the wide applicability to a range of SEI technologies that we are uncovering, we see tremendous potential for Six Sigma to serve as a strategic amplifier for SEI technology transition success. The same success should be available for other technolo-

gies and other organizations. If the success enjoyed by project participants using CMMI and Six Sigma is emulated, the SEI and its community could have a repeatable transition accelerator for all of its technologies.

Although we faced an unfortunate challenge in gathering information and publishing our results, we see this as a testament to the value and power of Six Sigma: many companies regard Six Sigma use as a competitive advantage and tightly control any information associated with it. Several potential case or interview participants began discussions with us only to suspend our discussions at the direction of their senior management or legal departments. Most of our participants have placed stringent requirements on what we may publish or even share with other SEI staff and collaborators about their use of Six Sigma. In several cases, these same organizations share other data about themselves freely with us and with the community; their treatment of Six Sigma is a departure. We theorize that this is an indirect confirmation of the value of Six Sigma.

The following subsections describe the refined scope and scale of this project, a partial listing of findings that have been abstracted from collected data, an overview of demographic and contextual information, and recommendations for further work. Results are based on the project data set, including information from 11 case study interviews, 8 partial case study interviews, and survey responses from more than 80 respondents, representing at least 62 organizations and 42 companies (some respondents maintained an anonymous identity). Because of the proprietary nature of our data and the non-disclosure agreements in place, the results in this public report are intentionally at a high level. Numerous other findings will be documented separately. Additional publications, with additional detail, are planned, pending review by project collaborators.

### 5.6.1 Refinement of Scope and Scale

Our initial project supposition was that Six Sigma might enable, accelerate, or integrate SEI and other technologies. Through discussions and an initial project survey, we further theorized that Six Sigma, used in combination with other software, systems, and IT improvement practices, results in

- better selections of improvement practices and projects
- accelerated implementation of selected improvements
- more effective implementation
- more valid measurements of results and success from use of the technology

Based on a combination of SEI interest and community interest and technology readiness, as

evidenced through discussions and an initial project survey, we selected the following project priorities:

- primary focus: CMMI adoption and IT operations and security best practices
- secondary focus: architecture best practices and design for Six Sigma

### 5.6.2 Primary Findings

Six Sigma is feasible as an enabler of the adoption of software, systems, and IT improvement models and practices (also known as "improvement technologies"):

- Six Sigma is influential in the integration of multiple improvement approaches to create a seamless, single solution.
- Six Sigma can accelerate the transition of CMMI (e.g., moving from CMMI Level 3 to Level 5 in nine months, or from CMM Level 1 to Level 5 in three years, with the typical time being 12-18 months per level rating). Underlying reasons are both strategic (change in focus) and tactical (how the processes are implemented).
- Rollouts of process improvement by Six Sigma adopters are mission-focused as well as flexible and adaptive to changing organizational and technical situations.
- When Six Sigma is used in an enabling, accelerating, or integrating capacity for improvement technologies, adopters report quantitative performance benefits, using measures that they know are meaningful for their organizations and their clients (e.g., returns on investment of 3:1 and higher, reduced security risk, and better cost containment).
- Six Sigma is frequently used as a mechanism to help sustain (and sometimes improve) performance in the midst of reorganizations and organizational acquisitions.
- Six Sigma adopters have a high comfort level with a variety of measurement and analysis methods. They appear to be unfazed by "high maturity" or "high performance" behaviors, processes, and methods, even when they are at a "low maturity."
- Some business sectors are realizing greater success than others regarding the use of Six Sigma as a transition enabler.

Additional, CMMI-specific findings include the following:

- Six Sigma is effectively used at all maturity levels.
- Case study organizations do not explicitly use Six Sigma to drive decisions about CMMI representation, domain, variant, and process-area implementation order; however, project participants consistently agree that this is possible and practical.
- Project participants assert that the frameworks and toolkits of Six Sigma exemplify what is needed for CMMI high maturity. They assert that pursuit of high maturity without Six Sigma will result in much "reinvention of the wheel."

Architecture-specific findings include the following:

- Many survey respondents are in organizations currently implementing both CMMI and Six Sigma DMAIC[3] and many are in organizations progressing or using Design for Six Sigma (DFSS). Of the latter, the majority are at least progressing with CMMI (but some are not using CMMI at all) and none are using the SEI Architecture Tradeoff Analysis Method (ATAM). Note, however, that multiple organizations we studied are pursuing the joint use of Six Sigma, CMMI, and ATAM, focusing on the strong connections among DFSS, ATAM, and the engineering process areas of CMMI.

We found no supporting or refuting evidence for several hypotheses and we have several inferences (conclusions derived inductively from evidence, but not supported directly by evidence) that were not pursued because of time constraints. These will be described in future documents.

### 5.6.3 Context of Findings

The following questions and answers provide context for our findings and should be helpful for organizations considering the use of Six Sigma as a transition enabler.

*What did the case study and survey organizations look like? (Or, more specifically, "Did they look like my organization?")*

Generally speaking, the organizations that are achieving success in their use of Six Sigma as a transition enabler ranged from low to high maturity, spanned nearly all commercial sectors, ranged from medium to large in size, and included organic and contracted software engineering as well as IT development, deployment, and operations. (Note that "small" organizations' use of Six Sigma remains on the project hypothesis list, having been neither refuted nor supported by project evidence.)

We did not set out with a research question about which domains were enjoying success—we simply wanted to find evidence of successful use of Six Sigma to improve transition effectiveness. That said, we have evidence from a wide range of organization types and domains, with one exception: We do not have direct evidence from DoD organizations for use of Six Sigma as a transition enabler, except from the field experience of the two researchers. (We do have evidence from industry organizations that serve the DoD.) This does not mean DoD organizations are not employing Six Sigma and enjoying the same benefits, only that we do not yet have evidence. In fact, this may point to an area of needed follow-on work.

[3] DMAIC = Define-Measure-Analyze-Improve-Control, one of the improvement frameworks of Six Sigma

*What technologies did organizations deploy in conjunction with Six Sigma?*

This project focused on organizations that were at least "progressing" both with one or more variants of Six Sigma and with CMMI, Information Technology Infrastructure Library (ITIL), and/or Control Objectives for Information and related Technology (COBIT). However, we gathered data on other technologies in use and they ran the gamut of Capability Maturity Models other than CMMI, the People CMM, ISO standards, the SEI Team Software Process (TSP), ATAM, Goal-Question-Indicator-Measurement (GQIM), and Electronic Industries Association (EIA) standards. Demographic statistics will be presented separately.

*Why and how did organizations use Six Sigma?*

Frequently, Six Sigma was adopted at the enterprise level and the software, systems, or IT organization was called upon to follow suit. In some cases, the adoption decision was made based on past senior management experience (e.g., at the direction of a new senior manager who was just hired from General Electric). In other cases, a "burning business platform" (e.g., lost market share) drove the adoption decision. In all cases, senior management sponsorship was definitive.

Regardless of why Six Sigma was selected, successful organizations consistently deployed it fully (i.e., the following were all present: senior management sponsorship, a cadre of trained practitioners, project portfolio management, the philosophy, one or more frameworks, appropriate measures, and the analytical toolkit). Organizations tailored the focus of Six Sigma and its improvement projects to target key performance measures and the bottom line. "Line of sight" or alignment to business needs was consistently clear and quantitative. CMMI or ITIL process areas were implemented based on business priorities and were integrated with the organizational process standard (even at lower maturity). Organizations varied as to whether CMMI or ITIL started first, Six Sigma started first, or they all started together; the variance was sometimes strategic and sometimes an effect of enterprise and SEI timing. The other aspect of deployment that varied was whether Six Sigma practitioners and process improvement group members were the same or different people and within the same or different organizational divisions. Organizations were successful either way.

*Why does Six Sigma work as a transition enabler?*

Based on our research and knowledge of both Six Sigma and technology transition, we find that Six Sigma supports more effective transition because it requires alignment with business drivers, garners effective sponsorship, supports excellent and rational decision making, aids robust implementation or change management, and offers credible measures of results for investment. The latter is particularly crucial for convincing majority adopters to transition,

and is often the sticking point in failed transitions (popularly labeled after Moore as failing to "cross the chasm"[4]).

### 5.6.4 Path Forward

Below is a brief listing of several possible follow-on projects that have been identified as a result of project findings and analysis. Pursuit of these areas will depend on available funding and confirmation of value and interest. Additional details will be documented separately:

- the robustness of Six Sigma as a transition enabler, including examination of requisite characteristics for organizational and technology fit, as well as appropriate measures of transition progress (with specific attention to small, acquisition, and DoD organizations and specific attention to both technology developers and technology adopters)
- the codification of DFSS and component-based development techniques to enable organizations to more effectively integrate and deploy multiple models and standards in a way directly focused on mission success
- use of Technology Design for Six Sigma to contribute to the development of a holistic architecture technology
- use of Technology Design for Six Sigma to harmonize IT models (or to provide "harmonization guidance" to organizations)
- the application of Six Sigma's analytical toolkit to advance the state of measurement and analysis practice in software, systems, and IT—for instance, the integration of Critical Success Factors, GQIM, and elements of Six Sigma for enterprise measurement and the demonstration of methodologies for language and text data processing
- the ability of Six Sigma's focus on "critical to quality" factors and on bottom-line performance to provide resolution among peers with a similar rating and to provide visibility into (or characterization of) the specific performance strengths of each. As an example, with Six Sigma, an organization might be enabled to reliably make a statement such as, "We can deliver this project in +/- 2% cost and we have capacity for five more projects in this technology domain. If we switch technologies, our risk factor is "xyz" and we may not be able to meet cost or may not be able to accommodate the same number of additional projects."

## 5.7 Publications and Presentations

As of the publication date, the following publications and presentations have resulted from this project. Conference presentations (e.g., Six Sigma for Software Development; CMMI Users Group) and more detailed publications are planned for 2004 and 2005.

[4] Geoffrey A, Moore. *Crossing the Chasm*. Harper Collins, 1991.

- Heinz, L. “Using Six Sigma in Software Development.” *news@sei*, 2004 No. 1 <http://www.sei.cmu.edu/news-at-sei/features/2004/1/feature-3.htm>.

# 6 A Method to Analyze the Reuse Potential of Non-Code Software Assets

**Dennis Smith, Liam O'Brien, Ed Morris, John Bergey, Grace Lewis**

## 6.1 Purpose

The goal of reusing legacy assets has been an important one for both DoD and commercial systems for the past 25 years. The Defense Science Board Task Force Report on Defense Software cites an increasing focus on the integration of pre-existing parts or components as a major driver of the professional services and software industries [DSB 2000]. Most current work on reuse has focused on the reuse of code assets; current efforts in general have had mixed results. While it is commonly accepted within software engineering that non-code assets comprise the most valuable and resource-intensive software assets, most current work on reuse has focused on code assets. As a result there is an important unfilled need to support decision making on the critical non-code assets. This study has analyzed the issues of non-code assets and has made extensions to the SEI Options Analysis for Reengineering (OAR) method on a trial basis to determine its applicability to the problem. (See Appendix B for the list of activities that compose SEI OAR.)

## 6.2 Background

### 6.2.1 Software Reuse Background

Software practitioners have long recognized that significant benefits may come from reuse; however, the results in general have been mixed. On the positive side, there have been several widely cited reuse success stories. Nippon Electric Company (NEC) achieved vastly improved productivity (approximately 6.7 times improvement) and quality (2.8 times) through reuse [Isado 92; Matsumoto 95]. On two projects, Hewlett-Packard (HP) reduced defects 76% and 24%, with productivity improvements of 40 to 50 percent. On the other hand, in a study of 24 European companies attempting software reuse for the first time, Morisio, Ezran, and Tully found that fully one-third failed in the attempt, even though the companies were attempting to produce software with high commonality to previous work and employed relatively sophisticated software processes in their other work [Morisio 2002]. The DoD and other government agencies have supported a number of reuse efforts, including ASSET, Mountainnet, CARDS, and Cosmic. While these efforts provided some value for

individual projects, they did not result in significant savings across a number of programs, and they are no longer well supported.

The primary reasons for these mixed results include:

- Component reuse within reuse repositories tends to be at a low level of granularity; substantial savings from reuse requires substantial reuse of components at a high level of granularity.
- The focus tends to be on the technical issues of describing and cataloguing a component; there has been less effort aimed at providing motivation for reusing the components, and at understanding the set of management and organizational issues that must be addressed.
- There has not been clear guidance on how to evaluate existing components for their reuse potential.
- There has been insufficient attention to the requirements and architecture of the target system. For example, a 2003 experiment conducted by the SEI with a major DoD effort found that reuse estimates varied by as much as 30% depending on different assumptions about the target architecture and its middleware.
- The initial focus has often revolved around one-time reuse; true savings occur with systematic reuse across multiple products.

## 6.3 Approach

### 6.3.1 Dimensions of Reuse

For this study we used a framework initially developed by Ruben Prieto-Diaz who distinguished factors that differentiate between reuse approaches [Prieto-Diaz 93]. We applied the framework to more recent work and used it as a starting point for understanding the application of non-code assets.

Prieto-Diaz identified the following aspects of reuse:

- *substance*—the category of the "thing" to be reused (e.g., ideas, concepts, artifacts, components, procedures, skills, component models). The list of potentially reusable non-code assets can include such artifacts as requirements; architecture and design documents; plans for development, integration, test, management, and risk; manuals; templates/checklists for any asset; processes; use cases; schemas; data dictionaries; test cases; and enterprise models.

  Most assets that an organization creates more than once (particularly those that are customarily created for projects) are potential candidates for reuse. In our study of the reuse potential of non-code assets, we initially focused on assets that are closest to the code

level because they are more amenable to use by the OAR method. We also proposed ways to expand this approach in the future.

- *scope*—the intent of reuse. Vertical reuse refers to reuse within a common domain, while horizontal reuse refers to cross-domain reuse. Early reuse efforts tended toward horizontal reuse of small-grained components across a wide range of domains. They provided access to large numbers of general purpose source-code artifacts (e.g., searches, sorts, abstract data types) that could be incorporated into an application. More recently, general purpose (as well as domain- and application-type specific) component libraries have become available for a variety of languages (e.g., Java, Visual C++, Ada).

  An SEI experiment found widely varying reuse estimates depending on different assumptions made about the target architecture and its middleware, which suggests that in order to maximize reuse, components must either be designed with the required architecture in mind, or the architecture of the system must be influenced by the assumptions embedded in the component.

- *mode*—how the reuse activity will be conducted. Reuse can be opportunistic (i.e., ad hoc identification of components for a given project) or systematic (i.e., carefully planned, with defined processes, guidelines, and metrics). Opportunistic reuse does not lend itself to large returns on investment, because the costs are all borne by a single project. Recovery of these additional costs is an unlikely proposition with ad hoc reuse. Nor is the organization likely to accrue other potential benefits of reuse (e.g., reduced cycle time, improved quality) on the first attempt. As a result, most experts now believe that the real benefits of reuse become apparent with systematic reuse across multiple products, such as reuse within product lines.

  Not surprisingly, much of the effort expended in making reuse work has been directed toward solving the many technical issues involved in describing, cataloguing, and connecting components. Less effort has been directed at creating the organizational, managerial, and motivational processes that are necessary for systematic reuse. While we should be sensitive to attributing all reuse failure to reuse process failure, we do know that organizational structures and processes that encourage and support reuse are characteristic of many organizations that succeed in implementing reuse programs. These organizations embrace structures and processes that are sufficient to generate and maintain management commitment, provide consistent execution of technical activities that encourage and support reuse, provide training and incentives, and support the collection and analysis of metrics that document reuse value.

- *technique*—the approach used to achieve reuse. Compositional approaches connect reused components into systems, and generational approaches employ system and component models along with predefined points of variance to generate systems or parts of systems. However, more important than the choice of composition versus generation is the degree to which the approach embodies a specific software architecture for resulting systems. Among the most successful reuse approaches are those that relate software archi-

tectures, components, and assembly/generation strategies in constructing software product lines. In one study of an organization implementing a product line, verbatim code reuse on new systems averaged almost 80% [CelsiusTech 96]. A second organization achieved a startling 97% reuse rate—with further improvements expected [Salion 02].

- *intention*—whether the reuse component is to be treated as a black box and incorporated as-is, or whether the component will be modified for better fit into the system. Both black box and white box reuse can be successful. The choice of a black or white box approach clearly depends on characteristics of the target system that necessitate the need for modifications to the component, but also on whether the component can be efficiently changed without affecting its basic nature. For example, it is normally unwise to change commercial off-the-shelf (COTS) components, even when a vendor provides source code, because changes to a COTS component can defeat the very reasons for selecting a component where someone else (i.e., the vendor) is responsible for component maintenance, upgrade, and evolution.

  With other types of reuse components where source code is available and modification common, a key to successful reuse is a structured process that systematically analyzes the component, the costs of modifying the component, and the long-term impact of the modifications. OAR identifies one such process that addresses the inevitable limits to comprehension of the characteristics of unfamiliar large-grained code assets, and the equally limited comprehension of the overall system when components are typically selected.

- *product*—the actual software products that will be reused. These include source code, design, specifications, objects, text, architectures, and other types of artifacts.

- *goal*—the artifact that will be produced from the reused components or artifacts. For example, one organization might reuse a general template for a software engineering plan in generating a specific software engineering plan. Another may directly reuse a software component.

- *granularity*—the size or range of the reuse product. For example, source code reuse may involve anything from small artifacts such as individual subroutines to very large artifacts such as complete subsystems and even systems (within a system of systems). Other reuse products can have differing granularity as well. For example, a reused design may be for something as simple as an algorithm to sort widgets, or as complicated as a design for a complete command-and-control system.

### 6.3.2 Reuse of Non-Code Assets

There are several reasons why reusing non-code assets may offer significant benefit to organizations:

- Cost expenditure for software development (not including maintenance, which commonly dwarfs development costs) is normally allocated by the 40-20-40 rule: 40% of the budget is spent prior to coding (e.g., on requirements analysis, architecture, and design),

20% is spent during coding, and 40% is spent after coding (e.g., on integration and testing). By this rule of thumb, coding represents only a moderately sized fraction of total development costs. The artifacts created during other phases of the development process represent a greater share of the total cost of building a software system. Following this line of reasoning, reuse of non-code assets offers at least as great a benefit as reuse of code assets.

- Non-code assets carry much of the intellectual value of a program. For reuse to occur, the intellectually valuable content should be incorporated in a reusable manner in the artifacts produced by architects and designers.
- Many non-code artifacts are potentially more readily reused than code-based artifacts. This is because some non-code artifacts involve abstract representations of a code component or of specific processes used to create the component. These abstract representations represent processes and software before incorporation of the situation-specific nuances involved in development. Since these nuances are an impediment to reuse, an abstract representation that excludes them should be easier to reuse.

There are also several reasons why reuse of non-code assets may be less beneficial than use of code assets:

- Code assets are significantly more valuable than non-code assets, because the value associated with code assets (from both a cost and intellectual standpoint) is a cumulative sum of the values of its non-code predecessors. That is, a code asset incorporates the value of good requirements work, good architecture, good design, and other tasks that led to the code. Reusing a code component brings all of this cumulative value. This is effectively like saying that if you reuse the code, you also reuse the architecture, design, and other non-code assets, even if only indirectly.
- It is precisely the details that are incorporated only in the code that have the greatest value for reuse. Yes, it is possible for an abstract representation to represent many systems, but that is a problem and not a benefit. Organizations typically want to reuse one specific system or system component that allows them to circumvent significant development work. The closer the details of that system or component are to those expected for the new system, the greater the benefit of reuse. This explains why it is relatively easy to reuse simple algorithms (searches, sorts, and such) but there does not seem to be significant additional reuse benefit in doing so. So, while the devil may be in the details, so is the profit.

### 6.3.3 Relevance of OAR Method

In recent years, large DoD acquisition programs, such as the Army's Future Combat Systems (FCS) and the Air Force C130 upgrade, have routinely mandated substantial reuse of legacy assets. Both of these programs have a Lead System Integrator (LSI) with the responsibility to integrate components provided by a large number of suppliers. Both have goals of extensive reuse.

The assumption behind these goals is that if existing software has functionality similar to that of the target system, substantial reuse can be obtained through minor adaptations to the existing code. However, many factors must be considered before decisions can be made about the fit of legacy assets into new target systems, which requires an understanding of the constraints of the target architecture and an analysis of the interfaces and types of middleware that are to be used. Despite these very real constraints and consequent risks, program managers have often made key decisions based on vague estimates given by potential suppliers.

For large-scale acquisitions, the risks of accepting supplier estimates for reuse are obvious. However, until recently, the analysis of supplier estimates has been informal at best. Methods for performing changes to components are available [Sneed 98; DeLucia 97]. Approaches to identifying risks in reengineering projects have also been developed [Sneed 97].

The SEI Options Analysis for Reengineering (OAR) approach represents the first systematic method to identify which components to mine and to determine the effort, cost, and risks of the mining effort for code assets. In the study, we analyzed the OAR approach to determine its applicability to non-code assets.

### 6.3.4 Synopsis of OAR

OAR is a systematic, architecture-centric method for identifying and mining reusable software components within large and complex software systems. Mining involves rehabilitating parts of an old system for use in a new system. Users of the OAR method can identify potential reusable components and analyze the changes that would be needed to rehabilitate them for reuse within a software product line or new software architecture. They can also identify mining options and the cost, effort, and risks associated with each option.

Successful mining requires an understanding of the types of components that are worth extracting and how to extract them. Once decisions are made on whether to extract, a set of problems often must be addressed, including the fact that

- existing components are often poorly structured and poorly documented
- existing components differ in levels of granularity
- clear guidance is not yet available on how to perform the salvaging

The OAR method consists of five major activities with scalable tasks. These are outlined in Figure 8.

Each of these activities has a set of tasks and subtasks that must be accomplished to meet the goals of the activity. Many tasks have data templates that provide a starting point for data items, as well as execution templates to guide the process. Each activity has a set of exit crite-

ria. In addition there is a feedback task at the end of each activity to analyze current progress and to determine if the method should be tailored based on the current context. Appendix A provides a brief summary of each of the activities.

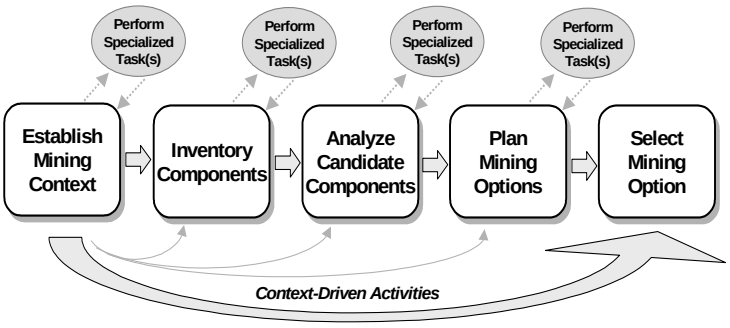



*Figure 8: Overview of OAR Activities*

### 6.3.5 Applicability of OAR to Non-Code Assets

The focus of OAR is on mining software components and the related artifacts needed to generate and use these components. We initially focused on determining whether OAR could be customized to make decisions on non-code assets. Our basic assumption was that OAR would be most relevant for non-code assets that have a one-to-one correspondence to the code. We identified a prospective set of non-code assets, and determined the amount of customization to OAR that would be required to evaluate that type of asset. This initial analysis is summarized in Table 3.

| **ASSET** (being mined) | **One-to-One Correspondence** | **Amount of Customization Required to OAR Process** |
|---|---|---|
| ***Software Components*** | ***Reference Point*** | ***NONE (Baseline Capability)*** |
| **Test Cases** | **YES** | **MODERATE** |
| | **NO** | **MAJOR** |
| **Use Cases** | **YES** | **MODERATE** |
| | **NO** | **MAJOR** |
| **Documentation Artifacts** | **YES** | **MINOR to MODERATE** |
| | **NO** | **MINOR to MAJOR** |

*Table 3: OAR Customization Required for Different Types of Non-Code Assets*

The reference point for Table 3 is the software component asset. Software components include all related artifacts needed to generate and execute the software component, such as makefiles, scripts, and required data files.

From this analysis we initially selected the example of test cases and determined that the types of modifications to OAR to accommodate mining test cases include:

- adding test cases to the elements being inventoried
- capturing test case characteristics
- reviewing test case documentation
- including test case characteristics in the screening process
- analyzing the changes (if any) required to rehabilitate the individual test cases
- estimating the corresponding cost, schedule, and risk of rehabilitating the individual test cases and comparative costs
- capturing and maintaining all test case information
- including test cases in the mining options and selectively rolling up the cost, schedule, risk, and comparative costs

We next made changes to the OAR process and templates to accommodate the test case example. We also examined other types of non-code assets, such as use cases and documentation artifacts. We are currently completing the application of several asset classes and have scheduled a pilot workshop with a DoD organization in October 2004.

## 6.4 Collaborators

The Carnegie Mellon/SEI team for this IR&D project consists of:

- Dennis Smith
- Liam O'Brien
- John Bergey
- Ed Morris
- Grace Lewis

This team has collaborated with a team from the Johns Hopkins University Applied Physics Laboratory (JHU-APL). The JHU-APL team is focusing on the reuse of artifacts from research prototypes and has provided insights on situations where these artifacts may be reusable. The JHU-APL team has also developed a model that focuses on how existing package elements (including non-code assets, such as design artifacts, requirements, algorithms, and

documentation) may be assessed to support transition from their current environment to new domains or applications. The revised OAR process represents a potential tool for making an evaluation about the applicability of reuse.

The JHU-APL team consists of:

- Rose Daley
- Mark Moulding

## 6.5 Evaluation Criteria

Evaluation criteria include:

- a review of the relevant reuse literature
- development of modifications to OAR to accommodate non-code assets
- feedback from a workshop with a potential user of the method
- a report that summarizes the study and its implications

## 6.6 Results

A review of the relevant literature has been conducted. These results have been used to determine modifications to the OAR method. The initial modifications have been made; additional modifications to accommodate other non-code assets will be completed before the conclusion of the study. These modifications have been done in conjunction with a pilot in the SEI Acquisition Support Program, which has added a second phase for more detailed evaluations and has broadened OAR to include COCOMO II (COnstructive COst MOdel) considerations. A workshop is being negotiated for October with FCS participants.

# 7 Emerging Technologies and Technology Trends

Technology scouting has always been an implicit activity of the Software Engineering Institute and is embedded in the SEI's mission of technology transition. Because of the institute's small size relative to other research institutions, the SEI applies the most leverage to its active initiatives, but it also watches for other emerging technologies, in the U.S. and internationally. The institute is currently organized into five programs with 10 initiatives.

The SEI has recently been asked to report on the state of the art of software technologies—those that are pushing the frontiers of the SEI's current programs and initiatives and also those that transcend them. The SEI Independent Research and Development program, described elsewhere in this document, is an example of explicit technology scouting at the SEI. The activities of the SEI New Frontiers Group, including information collection and dissemination, are further examples.

The SEI has also recently established a new technology scouting vehicle called the International Process Research Consortium (IPRC). The purpose of the IPRC is to develop a community of practice that regularly collaborates to examine and codify future process research opportunities and directions. IPRC members come from all over the world, bringing expertise in process research and a vision for the trends, challenges, and needs for software-intensive organizations over the next 5-10 years. Many IPRC members have ties to their regional governments and industries, which provides an excellent opportunity to transition the learning and recommendations that will come from the IPRC. Membership in the IPRC is limited to 40 individuals from academia, industry, and government. Deliverables over the next two years will include proceedings from collaboration activities (e.g., workshops) and a Process Research Roadmap detailing recommendations for process research covering the next 5-10 years. There will be six two- to three-day workshops between 2004 and 2006.

Another component of technology scouting involves watching technology trends, which requires the collection and analysis of information, and the dissemination of the information that the SEI considers relevant. All members of the SEI technical staff are indirectly, and often directly, involved in technology scouting when they engage in the activities of scientific inquiry, including:

- researching the literature relevant to their own work

- conducting searches on the Internet in relevant subject areas
- navigating the SEI's and other institutions' Web sites to learn about activities in areas other than their own. In the case of the SEI, this includes the Software Engineering Information Repository (available at seir.sei.cmu.edu).
- giving invited talks
- assembling a special issue for a prominent software journal
- attending conferences. Ideally, every person attending a conference writes a field report describing the new technologies, if any, that are the subject of presentations at the conference.
- participating on university advisory committees
- collaborating with customers
- participating in studios[5] for programs such as the Master of Software Engineering (MSE) at Carnegie Mellon University's School of Computer Science, or similar programs at other universities

In this report, we have provided descriptions of new or emerging technologies[6]. These descriptions include the technologies' purpose and origin. Where possible, we have indicated the technologies' level of maturity and have provided information about related trends. A bibliography for the technology descriptions is provided at the end of this report.

The following technologies are described:

- Open Grid Services Architecture
- Integrated Security Services for Dynamic Coalition Management
- Model-Driven Architecture
- Service-Oriented Architecture
- Web Services
- Automated Lexical and Syntactical Analysis in Requirements Engineering
- Q Methodology
- Emergent Algorithms for Interoperability

---

[5] A "studio" is a laboratory where students apply knowledge gained from core and elective courses in realistic, yet mentored, environments. In the case of the MSE program at Carnegie Mellon, former and/or practicing professionals are selected to mentor each studio project. Mentors bring their significant industrial experience to bear in guiding students in their application of methods, techniques, and technologies learned in the classroom to real-world problems encountered in studio.

[6] More detailed white papers, written by SEI technical staff members, are available for some of these technologies. For more information, contact SEI Customer Relations at 412-268-5800.

- Aspect-Oriented Software Development
- Generative Programming
- Software Assurance
- Recent Advances in Intrusion Detection Systems
- Advances in Software Engineering Processes
- Applying Statistics in Software Engineering

## 7.1 Open Grid Services Architecture

The open grid services architecture (OGSA) is a non-proprietary effort by Argonne National Laboratory, IBM, the University of Chicago, and other institutions, which combines grid computing with Web services. The goal of this architecture is to enable the integration of geographically and organizationally distributed components to form virtual computing systems that are sufficiently integrated to deliver a desired quality of service (QoS).

OGSA defines the mechanisms for creating, managing, and exchanging information among entities, called grid services. The open grid services infrastructure (OGSI) defines the standard interfaces and behaviors of a grid service [GGF 03]. The Globus Toolkit is an open source implementation of version 1 of the OGSI specification. Release 3.2 is available for download from the Globus Alliance Web site [Globus 04, Sandholm 03].

As stated previously, OGSA represents everything as a grid service. Grid services are stateful transient Web service instances that are discovered and created dynamically to form larger systems [Foster 02a]. Transience has significant implications for how services are managed, named, discovered, and used—and that is what makes a grid service different from a Web service. A grid service conforms to a set of conventions, expressed as Web service definition language (WSDL) interfaces, extensions, and behaviors, for such purposes as

- discovery; mechanisms for discovering available services and for determining the characteristics of those services so that they can be invoked appropriately
- dynamic service creation; mechanisms for dynamically creating and managing new service instances
- lifetime management; mechanisms for reclaiming services and state in the case of failed operations
- notification; mechanisms for asynchronously notifying changes in state

As OGSA evolves it will include interfaces for authorization, policy management, concurrency control, and monitoring and management of potentially large sets of grid service instances.

This emerging technology is currently being used mainly in e-science and e-business applications. However, there is great potential for its use in mission-critical systems, such as enabling collaborative targeting among multiple users and multiple sites. There is increasing support and research based on OGSA.

Given its growing industry support and the validity of its conceptual foundation, we believe there is a large possibility that OGSA is a technology that will emerge as a standard for grid computing.

## 7.2 Integrated Security Services for Dynamic Coalition Management

Integrated security services for dynamic coalition management is a DARPA-sponsored project managed by the U.S. Air Force Research Laboratory. It started in March 2000 with a duration of 36 months. The work was done out of the University of Maryland under Professor Virgil Gligor.

Coalitions are collaborative networks of autonomous domains where resource sharing is achieved by the distribution of access permissions to coalition members based on negotiated resource-sharing agreements—common access states. In a dynamic coalition, members may leave or new domains may join during the life of the coalition. To support security in dynamic coalitions, this project had two goals: (1) to enable the creation and management of coalitions with diverse and rapidly changing membership, dynamically, and (2) to provide solutions to fundamental problems of integrating diverse access control policies, public key infrastructure (PKI), and group communication technologies for dynamic coalition [Khurana 03].

The group at the University of Maryland developed a prototype of tools for coalition infrastructure services. The prototype includes joint policy administration services, certificate services, and group communication services. The tools support the joining, voluntary departure, and involuntary departure of coalition members. Accomplishments that are important to future systems include

- a definition of a common language to express access control policies using a role-based access control (RBAC) policy model
- automatic computation of access control states using constraint language and computation
- dynamic adaptation of policies based on joining and exit of participants

The results of this work are very important for network-centric warfare, the vision for which calls for dynamic coalitions sharing classified and unclassified information.

## 7.3 Model-Driven Architecture

Model-driven architecture (MDA)[7] is a conceptual framework for software development currently under development by the Object Management Group (OMG) [OMG 03]. The goal of MDA is to support software developers in separating business and application logic from the underlying execution platform technology. Promised benefits of using MDA for the software development process include reuse of higher-level artifacts (domain models), better implementation quality, improved portability of applications, and improved interoperability of applications and tools. MDA and related tools are not yet mature enough to predict the degree to which this promise will be realized.

The OMG has so far developed the fundamental concepts of model-driven architecture and, at the time of writing, working groups are defining new standards needed to realize the MDA concepts in practice. MDA is compatible with established OMG standards such as Common Object Request Broker Architecture (CORBA), Unified Modeling Language (UML), Meta Object Facility (MOF), Common Warehouse Metamodel (CWM) and the XML Metadata Interchange (XMI) as well as other industry standards (e.g., Web services and component frameworks such as Sun's J2EE and Microsoft's .NET). However, the overall MDA approach is vendor and technology neutral.

In the MDA approach, developers create platform-independent formal models—for example, in UML—of software applications, and generate platform-specific details through transformations of these models all the way down to the implementation source code. Model transformations rely heavily on the availability of sophisticated MDA tools, which are currently still at an early stage of development. Future tools are expected to implement a common model repository interface based on the OMG's MOF to support model exchange between tools through XMI, and to implement the upcoming QVT standard that will provide a tool-independent description of model transformations and code generation.

## 7.4 Service-Oriented Architecture

The simplest way to define a service-oriented architecture (SOA) is: an architecture built around a collection of services with well-defined interfaces—similar to the distributed component object model (DCOM) or object request brokers (ORBs) based on the CORBA specification. A system or application is designed and implemented as a set of interactions among these services.

[7] The term "architecture" in MDA does not refer to "software architecture." MDA is about model-driven development.

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 [Brown 02]. Common communication models are:

- Web services using simple object access protocol (SOAP) and Web services description language (WSDL)
- message-oriented middleware (MOM), such as IBM Websphere MQ
- publish–subscribe systems, such as Java Messaging Service (JMS)

What makes SOA different from DCOM or CORBA are the words “discoverable” and “coarse-grained” present in the previous definition of a service. Services must be able to be discovered at both design time and run time, not only by unique identity but also by interface identity and by service kind. Services are also ideally coarse-grained, that is, they usually implement more functionality and operate on larger data sets, as compared to components in component-based design. A typical example of a service is a credit card validation service.

Examples of service-oriented architectures are Web services using SOAP and universal description, discovery and integration (UDDI), Hewlett Packard’s E-Speak, and Sun’s Jini and ONE technologies.

Wrapping components and legacy systems as services is an approach to constructive interoperability.

Further descriptions of some elements of SOA follow.

## 7.5 Web Services

In its simplest definition, Web services are an instantiation of an SOA where service interfaces are described using WSDL, payload is transmitted using SOAP over HTTP, and UDDI is used as the directory service. Other combinations of technologies are possible, but this is the most common instantiation, which is why the terms SOA and Web services are often used interchangeably.

The growing success of Web services is due to a number of factors. Among them are:

- software components interact with one another dynamically via standard Internet technologies
- software components are built once and reused many times
- software components can be written in any programming language

- consumers do not need to worry about firewalls because communication is carried over HTTP
- systems can advertise their business processes so they can be consumed by other systems
- standards such as business process execution language for Web services (BPEL4WS), WS-security, WS-routing, WS-transaction, WS-coordination, and Web services conversation language (WSCL) are working toward the automatic discovery and composition of Web services

Because of the above, Web services are a widely used and proven approach to constructive interoperability.

#### Web Services Description Language (WSDL)

WSDL is used to describe what a Web service can do, where it resides, and how to invoke it. It is XML-based and supports simple and more complex transactions defined by message exchange patterns [W3C 04].

#### Universal Description, Discovery and Integration Service (UDDI)

UDDI is an XML-based distributed directory that enables businesses to list themselves, as well as dynamically discover each other [OASIS 02]. Businesses register and categorize the Web services they offer and locate the Web services they want to use. UDDI itself is a Web service. The directory contains three types of information, similar to a phone book:

- white pages, which contain basic information such as name, address, business description, and type of business
- yellow pages, which follow a categorization based on U.S. government and United Nations standard industry codes
- green pages, which contain technical information about the services that are exposed by the business that will help a client connect to the service

#### Simple Object Access Protocol (SOAP)

SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP uses XML technologies to define an extensible messaging framework containing a message construct that can be exchanged over a variety of underlying protocols, such as HTTP or email [W3C 03-1]. The most prominent use of SOAP over HTTP is to implement the message exchange mechanism for Web services.

SOAP is a stateless, one-way message exchange paradigm, but applications can create more complex interaction patterns (e.g., request/response, request/multiple responses) by combin-

ing such one-way exchanges. SOAP is silent on the semantics of any application-specific data it conveys, but it provides the framework by which application-specific information may be conveyed in an extensible manner. Also, SOAP provides a full description of the required actions taken by a SOAP node on receiving a SOAP message [W3C 03-0].

## 7.6 Automated Lexical and Syntactical Analysis in Requirements Engineering (QuARS)

An SEI affliate, Giuseppe Lami, has developed an automated tool that uses lexical and syntactical methods to analyze requirements and specifications. The output from the tool identifies sentences that are defective or weak. The tool also has the capability to cluster requirements by identifying selected word usage with a small lexicon. The tool also reports the distribution (physical location) of requirements throughout the document based on this lexicon.

Research into this tool is testing the following hypotheses:

- **Faster cycle time for inspecting and accepting requirements**
  Cycle time for requirements may be judged from many viewpoints: from the beginning of a project charter, from the start of customer interviews, or from the start of system specifications. This work analyzes the process from the first formal review of requirements until the requirements have passed the final validation step.
- **Fewer escaped defects**
  Several industry studies document the cost-ratio of fixing requirements versus fixing fielded defects. Values for these ratios have been reported as low as 1:200 and range as high as 1:1,000 or more. Organizations that analyze the root cause of defects can determine which defects were generated during requirements and specification.
- **Reduced effort or cost for inspection**
  There is some evidence that a group of inspectors has a limited find rate regardless of the defect density of the work product. Higher defect density then results in more hours of inspection. The test hypothesis is that the total time required for inspection is less because of the lower defect density after using QuARS.

Potential research questions include whether the capability to cluster requirements based on a small lexicon suggests investigating whether conflicting requirements can be more easily identified using this capability. Also, the capability to cluster requirements based on a small lexicon suggests the possibility of assisting domain experts with the identification of missing requirements.

## 7.7 Q Methodology

Q methodology was invented in 1935 by British physicist–psychologist William Stephenson and is most often associated with quantitative analysis because of its reliance on factor analysis [Stephenson 53]. Statistical procedures aside, Stephenson was interested in providing a way to reveal the subjectivity involved in any situation; for example in perceptions of risk, appraisal of costs, perspectives on user requirements, and opinions on training. Q methodology attempts to capture and ultimately measure life as lived from the standpoint of the person living it. It is a method that allows researchers to examine the subjective perceptions of individuals on any number of topics. It also helps to identify commonalities and differences in subjective perceptions across a sample group. In short, Q methodology is a research technique that allows the researcher (1) to identify, understand, and categorize individual perceptions and opinions, and (2) to cluster the perceptions in like groups.

The real utility of Q methodology lies in uncovering these opinion/perception clusters. Once identified, they can be targeted for follow-up activities, such as further research or programmatic activities. It is a combination of qualitative and quantitative research techniques that allows researchers to identify individuals who share common opinions. Q is often used for the following:

- identifying important internal and external constituencies
- defining participant viewpoints and perceptions
- providing sharper insight into preferred management directions
- identifying criteria that are important to clusters of individuals
- examining areas of friction, consensus, and conflict
- isolating gaps in shared understanding [Steelman 03]

The qualitative aspect of Q methodology is grounded in its ability to emphasize how and why people think the way they do. The primary goal is to uncover different patterns of thought—not to count how many people think the way they do [Valenta 97]. The quantitative aspect involves using factor analytic techniques (specifically, principal components analysis [PCA]) as a means for grouping like-minded individuals (down to discerning statistically significant variance of opinion from a single individual).

In short, Q methodology provides analysts with “a systematic and rigorously quantitative means for examining human subjectivity” [McKeown 88]. Q methodology constructs typologies of different perspectives based on subjective viewpoints.

Little known in software engineering circles, Q methodology may be useful to system development in many ways:

- understanding and mitigating pockets of resistance in system adoption
- targeting and tailoring system features, training needs, or security requirements
- isolating data standards requirements for system integration
- tailoring system performance measures and metrics
- understanding system risk elements
- tailoring checklists and criteria for understanding cost, schedule, and sizing estimates
- prioritizing requirements, risks, objectives, or other such factors in a group setting
- measuring group consensus
- revealing silent or minority voices in group settings
- measuring knowledge transfer between stakeholders

## 7.8 Emergent Algorithms for Interoperability

As systems become larger, with increasing numbers of autonomous components and greater geographic distribution, problems inherent in large-scale physical systems of all kinds become more prominent. Systems that were built and intended for specific limited purposes are combined to form systems-of-systems that apply their component systems to uses that were neither intended nor anticipated.

Problems that arise in the management, construction, and use of large-scale systems, distributed systems, and systems-of-systems are normally referred to as interoperability problems. Traditional technical approaches to solving interoperability problems focused on strengthening centralized control, increasing the visibility and transparency of components, imposing additional standards, and improving coordination among the organizations involved, so that traditional closed systems solutions become more effective. Obvious as this approach is, it is a losing battle. The continuing advance of memory, processor, and communications technologies ensures ever-increasing demands for systems and systems-of-systems that are more complex, more geographically distributed, with more poorly understood and unknown components, involving more and more organizations, and cooperating for purposes never anticipated by the component developers. Instead, effective solutions must be developed that recognize, act upon, and exploit the inherent characteristics of systems-of-systems.

Fortunately, problems of interoperability are not new. They are inherent in all physical systems whether biological, social, or economic. They are new only to software engineering and computer science. Effective solutions exist in biological, social, and economic systems, but those solutions frequently violate the traditions and methods commonly used in computer science and mathematics. Software engineering has been more open-minded in this regard with approaches such as Capability Maturity Modeling, which, in effect, reinterprets certain software engineering problems as management problems where there are known effective

approaches. Management methods of necessity focus on processes of human interaction drawn and refined from many centuries of history and the experience of complex social interactions.

Emergent algorithms exploit a similar approach but on a broader and more technical basis. In particular, research in emergent algorithms identifies, develops, and refines technical methods applicable to software engineering. These methods and techniques are derived by analogy from approaches that have been effective in social, biological, and economic systems. The methods of emergent algorithms include cooperation without coordination, dynamic adaptation, continuous trust validation, dynamic capability assessment, opportunistic actions, anticipatory neighbor assistance, encouragement and influence, perturbation, and survivable architectures. At the same time, emergent approaches demonstrate an aversion to tight coupling of systems, to dependency on centralized control and data, and to hierarchical structures.

At their most fundamental level, emergent algorithms exploit cascade effects in loosely-coupled contexts of dynamically changing, partially trusted neighbors to achieve a purpose shared by a subset of the participants. Desired global system properties, often in the form of continuity of services, emerge through the cumulative effects of the actions and interactions of all participants. Although emergent algorithms can be viewed as consensus algorithms that operate in the absence of needed information and effective control, this characterization is somewhat misleading because, as in politics, the consensus need be only of a minority.

Only a limited repertoire of emergent methods has been identified and they are not fully understood. The positive and ill effects of cascades are incompletely known. Phase shifts are a class of emergent effects that can occur in any physical system. They have seldom been studied, but offer the potential for both dramatic benefits and catastrophic failures. Initial research in emergent algorithms at the SEI has been primarily in support of survivable systems, infrastructure assurance, and Internet security.

There has been related research at other institutions for different purposes. Work in genetic algorithms aims at discovery of non-obvious solutions through dynamic mutation of algorithms. Decentralized thinking focuses on the mindset required to exploit the characteristics of real world systems by observing and experimenting with loosely coupled algorithms. Research in swarm intelligence has focused among other things on the development of specific algorithms that use emergent methods to find better but nontraditional solutions to well known problems.

To the best of our knowledge no other research has pursued emergent phenomena as a means to develop methods generally applicable to interoperability problems. There has, however, been a recent recognition of emergent characteristics in many quarters. Popular books such as

*The Tipping Point* and *Normal Accidents* discuss emergent phenomena in everyday life. The emergent effects of epidemics, not just of diseases but of rumors and fads, are well known.

The principles of network-centric warfare (NCW) are very similar to those of emergent algorithms but are specialized to a particular class of military operations. NCW also demonstrates a willingness to embrace new radical methods with demonstrable benefits when the inherent limitations of traditional approaches become apparent.

## 7.9 Aspect-Oriented Software Development

Aspect-oriented software development (AOSD) is a promising emerging technology. AOSD addresses problems experienced with object-oriented development, but has much greater applicability across software development in general. This is not a mature technology but its large-scale adoption by IBM promises to greatly accelerate its maturation.

The ability to build large, complex systems rests on the ability to separate *concerns* (because the parts are less complex than the whole), and to encapsulate information about a concept so that unnecessary dependencies are avoided [Parnas 72]. AOSD provides support for separating development concerns using different sets of partitioning criteria at the same time, and in most of its forms, provides support for encapsulating information about that concern. Aspect-oriented design provides the software engineer with options for finer-grained design elements that can be implemented using aspect-oriented programming (AOP).

Prevailing design techniques, particularly object-oriented techniques, end up decomposing the software based on a specific point of view. AOSD provides a means for additional decompositions from other points of view that cut across the original, primary decomposition. Informally, those additional decompositions are aspects. Some implementations of AOSD provide automated support for combining the additional decompositions with the primary decomposition to produce a single program.

AOSD is a direct result of the practical experiences gained from object-oriented developments and the problems experienced there. There are certain behaviors that cannot be encapsulated conveniently within a single class, so the definition of these behaviors must be manually coordinated between classes and between the developers of those classes. AOSD provides an extension to object-oriented design that gives an additional means of defining a type of structure beyond the association and inheritance relationships between classes. This additional dimension addresses cross-cutting behaviors.

Aspect-oriented design supports a number of design techniques. For example, it allows object-oriented designers to achieve code normalization in a manner similar to database normalization. Tim Ottinger has defined a set of normal forms for objects. This capability sup-

ports the independence of individual concerns and isolates each from changes in other concerns that are composed in the same product [Ottinger 04].

AOP provides implementation capability for designs developed using aspect-oriented design. AOP is supported mainly by extensions to existing object-oriented programming languages. These extensions have been enabled by static code analysis techniques and dynamic reflection.

AOSD yields two primary benefits: an increase in flexibility and a reduction of complexity. Software designed and implemented with aspects is more flexible because certain elements of the design are bound later in the development process. AOSD achieves a reduction in complexity because design concerns can be separated, maintained, and then integrated with a minimum of interactions. Other benefits include finer-grained encapsulation, for greater control and consistency in the code, and enhanced reusability, through the reuse of aspects.

AOSD has implications for large, complex products such as Web servers, and even smaller, simpler products that are being built en masse, such as printer drivers. Therefore, it is of interest to both the software architecture and product line communities.

In organizations adopting the product line approach AOSD allows for capturing, managing, and implementing product variations. The choice of a value at a variation point often affects the implementation of several components in the product. An aspect provides a vehicle for conceptually encapsulating the value while physically distributing that information across a number of implementations.

AOSD has implications for the production planning activity in a product line. The product production strategy defines concerns, related to the product line goals, which conceptually cut across the definitions of the core assets. The concerns cut across the definitions because the primary decomposition of the core assets relates to how the core assets are built but not how products are built. Each product production concern can be mapped on to multiple production aspects. Each aspect defines exactly how specific core assets must be defined to support the product production goals.

AOSD has an impact on software architecture by providing an additional approach to decomposition and a different-sized building block for system definition. Aspects can be identified during attribute-driven design of the architecture and then mapped forward to be implemented as code-based aspects using AOP [Bass 04].

Finally, AOSD is of interest to the DoD community as an effective technique for implementing large, complex systems. Decomposing a complex problem into a primary decomposition and a set of aspects reduces the complexity of any one piece, allows for more concurrent de-

velopment, and supports mass customization of products. AOSD provides enormous flexibility to make changes. This facilitates prototyping and requirements investigations.

AOSD can have a great impact for DoD because it can be used in the development and maintenance of large, complex systems. However, AOSD requires a higher level of skill at design and closer coordination of producers and consumers of pieces (in this case the aspects). Current DoD development practices make this difficult to do.

Aspects provide an opportunity for implementing architectural design patterns rather directly since they cut across multiple modules in a program. DoD-specific design patterns for concerns such as security and error handling can be defined, reference implementations developed, and canonical solutions achieved.

Product production is the main DoD goal. The ability to develop production plans more quickly, efficiently, and accurately should be a major benefit. This could particularly be important on projects in which multiple vendors are cooperating to produce the products.

AOSD is a promising new technology and bears continued monitoring. The AOSD community has much in common with the early object-oriented community: a new technology born of a programming language breakthrough; a small but fervent community focused on the programming implications of that technology; and a splinter group within that community investigating the wider implications of the technology [Northrop 97].

The commitment of industry including HP, IBM, and BEA may well hasten the maturation of AOSD. Their use of the technology will more quickly uncover the gaps of knowledge that only appear when solving industrial-strength problems. The focus on e-commerce and Web servers will also hasten the discovery of relevant design patterns.

Because of the strong connections between aspect-oriented software development with software architecture and software product lines, the SEI has been carefully monitoring developments in this field as it has evolved over the past few years. The Product Lines System Program is already involved with the AOSD community. It needs to elicit their support to make needed software tools a reality. Further investigation of the connections between AOSD and software architecture and software product lines is required.

## 7.10 Generative Programming

Recent innovations in software modeling in languages for specific software domains, and in the composition of systems from components are bringing generative programming from the research lab forward to meet the practical needs of software development in a range of appli-

cation areas. While not yet state of the practice, generative programming in its various manifestations deserves further investigation and technology tracking. It may lead to ways for programmers to automate the development process.

Generative programming (GP) captures the concepts of automated development based on "modeling software system families such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge" [Czarnecki 00]. More simply stated, "generation is about writing programs that write programs" [Harrington 03]. The technologies of GP include templates that are expanded like macros, the reflective facility in certain languages, domain specific languages, and model-driven approaches.

Generative programming is of interest to the software architecture and product line communities. The elementary reusable components used as the basis for generating products are designed and implemented in the context of a specific architecture. Attribute-driven design can have significant impact on the shape of the reusable components and likewise the nature of generative programming can have an impact on the attributes that shape the architecture. In a recent survey, 10 of 22 respondents stated that they used some form of automatic product generation technique for product production in a product line [Chastek 04].

Generative programming can provide meaningful benefits to the DoD community. DoD systems often require a significant level of customization for specific configurations of hardware or, in current plans for command and control, Web- or other service-oriented architectures. GP provides a paradigm for system development acquisition of individual products, which actually means acquisition of the capability of generating either customized components or customized integrations of those components into systems. The DoD can rely on in-house domain expertise or expertise in the field for collecting requirements that generators use to produce products within the scope of a product line.

## 7.11 Software Assurance

The overall goal of software assurance efforts is to build confidence that software does what it is supposed to do and does not do what it is not supposed to do. Discussions of software assurance can be organized into four general focus areas:

1. **People**, including developer and team composition
2. **Process**, including high-level criteria (like those in Capability Maturity Models [CMMs] and ISO-9000), design, and code review and inspection
3. **Technology**, including language evolution, automated verification, dynamic assurances, and testing

4. **Acquisition/business/return-on-investment**, including efforts to add automated product verification to common criteria as well as vendor and product evaluation techniques and software-assurance-oriented enhancements to the purchasing process. There is also strong support for defining a minimum set of verifiable product characteristics with an eye toward creating an infrastructure like the "UL for software."

Past progress in each of these areas has had some impact on the goal of the SEI's software assurance efforts. However, a focus on the key leverage points—efforts that can be shown to be effective, adoptable, and sustainable in real-world projects—is likely to produce the most rapid results. The most promising projects to satisfy these criteria are efforts that overcome the shortcomings of past efforts that have shown partial success.

**Focus of "people" efforts: improving relevant software-assurance information sources**

There is a huge gap between the state of the art and the state of the practice, especially when it comes to developers knowing how to avoid common security pitfalls. Effort is needed to improve developer access to effective, adoptable, and sustainable information for use on real-world projects. Ongoing research is needed to assure that the information is highly usable and remains tied to the most troublesome vulnerabilities. The Department of Homeland Security (DHS), through the U.S. Computer Emergency Readiness Team (US-CERT), is funding such an effort. The work for this effort is being carried out by the SEI and Cigital, Inc.

**Focus of "process" efforts: a psychologically sound, feedback-driven process**

One such project is the SEI Team Software Process (TSP). TSP builds upon much of the "what" defined by the various Capability Maturity Models and other process-evaluation criteria and provides a specific "how" that has been shown to be highly effective. A key element of TSP is its feedback loops, which mimic the continuous improvement foundation that has been shown to be effective in other industries, such as the auto industry. TSP's effectiveness has been demonstrated by more than 10 years' worth of data across thousands of projects. An analysis of recent results from 20 real-world projects showed that TSP teams met their project schedules while producing products that had 10-100-times fewer defects than typical projects. TSP was developed with acute attention to the psychology of individual programmers, the behavior of working units, and the realities of the business world, thus making it adoptable and highly sustainable. Its ability to co-exist with popular process definitions like Extreme Programming (XP) further enhances its adoptability.

Further work is needed to build upon the success of TSP to provide a culture of widespread support in the form of process tools integrated into programmers' daily development environment. Research is needed to identify how best to integrate with the configuration management, task/defect tracking, and integrated development environment (IDE) tools that are used today. Additional research is needed to identify how best to incorporate the benefits of

automated verification tools in a TSP-like process. Research is needed to explore the benefit of slackening some of the more demanding aspects of TSP in exchange for a shorter ramp-up, better leveraging today's tools and much more wide-spread adoption. A 10-times improvement would still be significant and might provide a stepping stone for the 100-times improvement that full TSP can provide. In addition, more effort is needed to increase awareness of the effectiveness of TSP-like process definitions.

**Focus of "technology" efforts: a new generation of automated verification tools**

Code scanners have been around for almost as long as software. Formal methods came later, but in their original forms, both suffered from adoptability problems. The high false-positive rates of early code scanners frustrated programmers, and such tools soon become shelfware. Many research tools in the area of formal methods did not scale to real-world-sized projects and could not be applied to legacy code, which is involved in almost all projects today.

Recently, however, significant strides have been made in the area of automated verification. Work at research universities, including Carnegie Mellon, Stanford, the University of Virginia, the University of Washington, and others, as well as at commercial labs such as Microsoft's, have focused on the issues of scalability, adoptability, and applicability to legacy code. Such tools are now effective at making assurances with respect to some attributes in million-lines-of-code systems. Adoptability has been greatly enhanced, as a result of integration with common integrated development environments (IDEs), taking advantage of the existing user interface paradigms that are already familiar to the programmer, and a reduction in false-positive rates. One such tool, the Fluid project at Carnegie Mellon, allows the developer to incrementally evolve the tool's understanding of the original design intent, all the while assuring the areas where the code complies with that design intent, and highlighting where it does not. This incrementality allows the tool to be effective on real-world projects involving legacy code. Furthermore, it takes away the feeling of hopelessness that developers invariably feel when a long list of "potential violations" is spit out of a static analysis tool.

Perhaps just as significantly, the focus has shifted from "finding potential violations" to making positive assurances about an attribute of code. It is much more valuable to say that a given body of code is sound with respect to some attribute (e.g., it has no potential for race conditions) than to find a few instances where that attribute is non-conforming (e.g., here is a race condition).

One of the key factors that distinguishes automated verification from other areas that have been identified above is that it can have an impact even after the code is written. It can be effective against problems that are hard to inspect and hard to test, like concurrency. This alternative coverage, combined with its applicability even to released and legacy code, makes it a good complement to the other focus areas described above. It should also be noted that the expense of running an automated verification tool can be significantly lower than project-

wide adoption of the other measures mentioned. Such low cost—especially when combined with high effectiveness, IDE, and process integration—should lead to sustained use by developers.

Significant additional research is needed to expand the breadth of attributes about which we can make automated assurances. This research effort should be guided by reviews of the most troublesome vulnerabilities and defects.

**Focus of "acquisition efforts": focus on the product; use automated verification**

Most acquisition-oriented software assurance resources focus on evaluating the processes used by the software vendor in producing the product. What is needed are tools for acquirers to actually evaluate the product. Automated verification tools offer the best hope for providing this capability. The use of such tools is needed to complement the common criteria. Research to define a minimum set of verifiable product characteristics is needed. This list will start out focused on assurances against the most troublesome software security vulnerabilities. This will later evolve to allow increasingly higher levels of assurance. The feasibility of establishing the "UL for software" should be investigated.

## 7.12 Recent Advances in Intrusion Detection Systems

Network-based intrusion detection systems (NIDS) remain the best practice for most security operations. This software monitors a network link to detect malicious behavior. While approaches to identify this malicious behavior have evolved over time and vary by product, most detect attacks through content inspection, the modeling of protocols and observing state violations, or through the modeling of the semantics or relationships in the communication. It is common for a single product to implement multiple detection algorithms.

NIDS have historically been plagued with three problems: high false-positive rates, the inability to detect new types of attacks, and the lack of scalability on high-speed networks. To this end, the technology has evolved to not only minimize these concerns, but also to respond to new threats and technologies. This improvement is primarily the result of advances in the following three areas:

- **Intrusion prevention systems (IPSs)** have addressed the reaction time of the operations centers. No longer merely passive devices, IPSs attempt to stop intruders and worms from making inroads into a network either at detection time or after an initial compromise has been discovered. As with an IDS, an IPS uses a detection engine to identify malicious activity. However, by integrating into the switching or routing fabric of a network (either in-line or through a command-and-control channel), the IPS can block certain traf-

fic deemed malicious, and as appropriate, make changes in the network by selectively disabling the hardware ports associated with a compromised computer or network. IPSs can provide traffic filtering that is significantly more sophisticated than is possible with a firewall. IPSs are also capable of quarantining individual machines or entire networks to contain damage. Despite the potential benefits of automated response, organizations should be leery of relying too much on this capability. Given that many current IPSs have detection engines similar to those found in IDSs, they suffer from comparable false-positive rates. The implications of false positives in an IPS, when the response may entail direct manipulation of the network, are significantly more acute. There is great danger in allowing an attacker to force a change in the network infrastructure.

- **Flow analysis** allows for greater scalability and the detection of new attacks. Flow analysis concentrates not on details of the communication (i.e., packet payload), but on the properties of the communication, making use of the protocol header fields. Such detection engines take into account factors such as timing, duration, and volumetric metrics often compared against a historical baseline of traffic. These approaches allow for the detection of denial-of-service (DOS) attacks and reconnaissance (even when done extremely slowly), and can be deployed on high-speed networks. The additional benefit of this approach is that certain flow-analysis engines can identify previously unknown attacks because they look for malicious patterns of behavior rather than specific attack methodologies. While powerful in detecting certain classes of activity, flow analysis should be used in conjunction with other approaches. Flow analysis is only able to assess the behavior of an event rather than the event's specific features, making it ideal for detecting only pervasive activity, rather than activity of limited scope. Likewise, given its behavior approach, trends will likely only be seen when observing a large network. Partial instrumentation or access to only subsets of data also impact the effectiveness of this analytical approach.

- **Security event managers (SEMs)** and other data-sharing systems have lowered false-positive rates through the use of contextual information. In order to create a unified view of security across an enterprise, SEMs and security information managers (SIMs) were developed. This new class of technology aggregates, centralizes, and analyzes data from many different types of technologies. SIMs will normalize similar data regardless of the vendor, provide a framework in which to correlate the same activity detected across multiple devices, and potentially identify malicious activity not previously possible to detect from the vantage point of only a single device. In addition to fusing the obvious security information such as IDS, firewall, and access logs, SIMs also have the ability to incorporate contextual information to reduce false positives and set priorities based on organizational mission. SIMs often include vulnerability scanning, penetration testing, topological information, and usage policy information into the analysis process. While SIMs have the ability to provide an aggregated, high-level view, the quality of their analysis is based entirely on the underlying data sources used. If the underlying security devices do not provide a sufficient level of detail or vary the frequency of their reporting, the analytical value of SIMs is diminished.

**Future Considerations**

Neither flow analysis nor content inspection is a universal solution. The former scales to high-speed networks, but only performs surface analysis. Content inspection provides deep analysis, but cannot scale to all networks. The ideal deployment makes use of the strengths of both of these approaches: deploying flow analysis at the border and content inspection toward the leaves in the network, with attention-focusing mechanisms between the two. Flow analysis can invoke content inspection to more closely examine suspicious activity. Content inspection can use flow analysis to better monitor the communication of previously observed suspicious actors.

Also, the analytical and detection capability of any security operation must mirror the adoption of technology in the market. To this end, more comprehensive support for IPv6 and wireless technology is necessary in IDS technology. Many IDSs currently support IPv4 making them completely unaware of any activity on an IPv6 overlay network. In short, detection engines must be completely rewritten to support this protocol, and the new traffic patterns it will imply after full adoption and during the transitional period from IPv4. Given the ease at which physical and link-layer attacks can be made on wireless networks, the detection of this activity must be shifted from specialized tools to more mainstream IDSs, or integrated into operations through a SIM.

With the effective demise of a strict perimeter in the network through the use of VPNs, wireless networks, handhelds, and cell phones, the notion of “inside vs. outside” the network is quite fluid. Given this fluidity, architectures that presuppose this old binary model must be reevaluated with the goal of identifying the new attack vectors of a malicious outsider, while also considering the largely ignored threat of a trusted insider. Furthermore, increased data sharing, facilitated by IDSs and SIMs that support standard data exchange formats, sanitization filters, and analysis algorithms that make sense of data that spans multiple policy domains, can provide a more complete view of security for organizations by incorporating the threat information provided by external parties.

A positive advance is the widespread adoption of anti-virus software and firewalls bundled with operating systems. In order to differentiate themselves from their competitors, antivirus vendors are increasing the sophistication of their software. No longer do they merely detect viruses; they are branching out to become the equivalent of host-based intrusion detection systems (HIDSs) with the preventive capabilities of IPSs. These tools monitor changes in the file system, the execution of privileged operations, and network communication. This trend will yield better security for the end-hosts. The challenge is making these tools properly integrate into organizational network management and the security infrastructure.

IDS technology has evolved to provide a more proactive role in network defense. Likewise, organizations have realized the importance of aggregating and correlating different types of

security and network-management information from inside and outside the organization. The future of this class of technology lies in continuing to integrate it as seamlessly as possible into larger security infrastructures, while understanding the security implications of newly adopted or composed technology. It is clear that while automated tools such as IDSs can help a trained security analyst, there is no substitute for an analyst's skills and the consistent application of best practices in managing the network infrastructure.

## 7.13 Applying Statistics in Software Engineering

The last decade or so has seen an increasing number of companies learn how to apply statistical concepts to software development. This is evidenced by the increase in organizational maturity over that period (see the SEI Maturity Profile at www.sei.cmu.edu/sema/profile.html), which stipulates more and better data collection and analysis.

In spite of this, there is still debate as to the applicability of statistical analysis to more than a limited subset of the many development environments in existence today. There is not a full understanding of how statistical methods can be applied in software engineering scenarios and, to date, limited case studies and examples have been published.

As organizations seek to improve their software engineering processes, they are turning to quantitative measurement and analysis methods. Statistical process control (SPC), a discipline that is common in manufacturing and industrial environments, but has only recently received attention as an aid for software engineering [Florac 99], has been generating some interest, as have six-sigma applications, and capture/recapture methods. Hopefully, these will be areas of further research and application that might yield results in the future. Effective use of these applications requires a detailed understanding of processes and a willingness to pursue exploratory analysis. As with anything new, there is a learning curve. To learn how to use a specific method or technology, one needs to be willing to conduct research, try things, make mistakes, and try again. Knowing and understanding the process is fundamental; consistency in data collection and reporting is imperative; and clarifying and understanding how the data is defined is crucial to knowing what the data represents.

Transitioning some of these concepts and techniques into actual software engineering practice remains a challenge. Many organizations do not collect appropriate data about their products and processes. Good data is a prerequisite to good analysis. Also, software engineering curricula at universities need to emphasize data collection and analysis topics, perhaps through joint efforts with statistics departments.

## 7.14 Advances in Software Engineering Processes

The movement to improve software engineering processes continues to make incremental advances in a number of areas. Several notable advances and trends are described below.

**Reducing Software Defects to Improve Security**

Defective software is not secure. This is a position advocated by the SEI and a few other organizations (e.g., PRAXIS and Cigital), and has been accepted by the Department of Homeland Security (DHS) Software Process Subgroup of the Task Force on Security. This position is supported by the fact that the leading cause of software vulnerabilities is common defects in software design and implementation (i.e., bugs). Also, tools for developing secure software, although needed, are not sufficient and address only a small part of the problem. Formal methods, better processes, and training for software professionals will have more impact and are critically needed. The DHS subgroup made the following recommendations:

- Principal short-term recommendations
  - Adopt software development processes that can measurably reduce defects in software specification, design, and implementation.
  - Adopt practices for producing secure software.
  - Determine the effectiveness of available practices in measurably reducing software security vulnerabilities, and adopt the ones that work.
  - The Department of Homeland Security should support the U.S. Computer Emergency Readiness Team (US-CERT), the Information Technology Information Sharing and Analysis Center (IT-ISAC), and other entities to work with software producers to determine the effectiveness of practices that reduce software security vulnerabilities.
- Principal mid-term recommendations
  - Establish a security verification and validation program to evaluate candidate software processes and practices for effectiveness in producing secure software.
  - Industry and the DHS should establish measurable annual security goals for the principal components of the U.S. cyber infrastructure and track progress.
- Principal long-term recommendations
  - Certify those processes demonstrated to be effective for producing secure software.
  - Broaden the research into, and the teaching of, secure software processes and practices.

These recommendations are likely to have far-reaching impact on software development practices, tools, training, and education. For example, attention to these recommendations could reverse recent trends in software engineering that advocate less formal, and more defect-prone development methods. The initial impact will likely be in the area of secure software development, but safety-critical systems have similar characteristics. Research in the security area should also interest the DoD and DoD contractor communities.

### Use of Tabular Expressions

Part of a larger area called relational methods, tabular expressions are a way to formulate software specifications and software design so that they are more easy to implement and review and are less error-prone. This work, originally done by David Parnas, dates from the late 1970s. More information is available at the Software Engineering Research Group Web site at McMasters University, Canada [need reference]. There has been no experimental research in industrial settings using high-quality, fully instrumented processes such as the SEI Team Software Process (TSP).

### Stratified Systems Theory

A formal theory of management, stratified systems theory, which incorporates some operational methods, has gained some traction in the general marketplace, including the U.S. Army. It has potentially broad application in both a general technology transition sense and in a more narrow software engineering management sense. What many practitioners have recognized intuitively (or from hard experience) as “hopeless” organizational situations might actually be formally describable and potentially fixable by applying these methods. This work is the brainchild of the late Eliot Jaques.

### Model-Based Process Improvement

The value of model-based process improvement is becoming more widely recognized. Process improvement is gradually expanding beyond the software and systems development groups and IT/IS shops to other parts of the enterprise to encompass non-software/systems portions of product and service development (i.e., hardware engineering). Indicators of progress in this area include

- operations and services are increasingly targeted by new standards
- safety and security are increasingly called for
- new industries are increasingly getting involved

### Increasing Synergistic Use of Multiple Process-Improvement Technologies

Increasing synergistic use of multiple process-improvement technologies is gaining recognition. This includes

- deploying Six-Sigma, TSP/PSP and Agile with CMMI
- increasing use of the Project Management Body of Knowledge (PMBOK) to improve management competencies

### Increasing Efforts to Harmonize Various Systems and Software Standards

This is another recognizable trend. Indicators include

- efforts by IEEE to harmonize its standards with ISO and CMMI
- efforts by ISO to harmonize its standards related to quality as well as integrating systems and software

**Wider Use of Appraisal Methods**

A broad spectrum of appraisal methods is increasingly in use, as indicated by the upsurge in the use of the following SEI methods:

- SCAMPI Class A methods used to establish benchmark public ratings
- ARC Class B and C methods used to (1) motivate process improvement, (2) gain familiarity with the CMMI model, (3) ascertain progress in process improvement, and (4) determine readiness for SCAMPI Class A appraisal

**More Quantification of Process Improvement**

There is an increasing desire to quantify the results of process improvement, including

- quantifying the cost and benefit of particular improvements
- sustaining process improvement funding
- establishing public benchmarks of organizational capability

# Appendix A: Bibliography for Emerging Technologies and Technology Trends

For further reading on the technologies described in Emerging Technologies and Technology Trends, see the sources listed below. URLs are valid as of the publication date of this document.

### Open Grid Services Architecture

Foster, I.; Kesselman, C.; Nick, J. & Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration <http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf> (June 2002).

Global Grid Forum—Open Grid Services Infrastructure Working Group. Open Grid Services Infrastructure (OGSI) Version 1.0 <www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf> (June 2003).

The Globus Alliance. <www.globus.org/ogsa/>.

Sandholm, T. & Gawor, J. Globus Toolkit 3 Core—A Grid Service Container Framework <www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf> (July 2003).

### Integrated Security Services for Dynamic Coalition Management

Khurana, H.; Gavrila, S; Bobba, R.; Koleva, R.; Sonalker A.; Dinu, E.; Gligor, V. & Baras, J. “Integrated Security Services for Dynamic Coalitions.” *Proceedings of the DARPA Information Survivability Conference and Exposition*. 2003.

### Model-Driven Architecture

Bass, Len; Clements, Paul; Kazman, Rick. *Software Architecture in Practice, Second Edition.* Addison-Wesley, 2003.

Brown, Alan. “An Introduction to Model Driven Architecture; Part 1: MDA and Today’s Systems.” *The Rational Edge* <www-106.ibm.com/developerworks/rational/library/3763.html> (February 2004).

Clements, Paul & Northrop, Linda. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

Clements, Paul; Kazman, Rick & Klein, Mark. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.

Clements, Paul; Bachmann, Felix; Bass, Len; Garlan, David; Ivers, James; Little, Reed; Nord, Robert; Stafford, Judy. *Documenting Software Architectures: Views and Beyond*. Addison Wesley, 2003.

Cook, Steve. "Domain-Specific Modeling and Model Driven Architecture." *Business Process Trends* <www.bptrends.com> (January 2004).

Duggan, Jim; Driver, Mark; Feiman, Joseph; Light, Matt; Lanowitz, Theresa; Blechar, Michael; Vecchio, Dale; Sinur, Jim; Natis, Yefim; Andrews, Whit; Valdes, Ray; Fenn, Jackie; Linden, Alexander; & Pezzini, Massimo. *Hype Cycle for Application Development, 2004*. Gartner, Strategic Analysis Report, 25 June 2004.

Guttman, Michael. "A Response to Steve Cook." *Business Process Trends* <www.bptrends.com> (February 2004).

Institute of Electrical and Electronics Engineers. *IEEE Std 1471-2000*. Piscataway, NJ: IEEE Computer Press, 2000.

Kleppe, Anneke G.; Warmer, Jos; & Bast, Wim. *MDA Explained*. Boston, MA: Addison-Wesley, 2003.

McNeile, Ashley. "MDA: The Vision with the Hole?" <www.metamaxim.com/download/documents/MDAv1.pdf> (2003).

Mellor, Stephen J. & Balcer, Marc J. *Executable UML: A Foundation for Model-Driven Architecture*. Boston, MA: Addison-Wesley, 2002.

Mellor, Stephen J.; Scott, Kendall; Uhl, Axel; & Weise, Dirk. *MDA Distilled*. Boston, MA: Addison-Wesley, 2004.

Object Management Group. *MDA Guide Version 1.0.1*. Document number omg/2003-06-01, 12 June 2003.

Selic, Bran. “The Pragmatics of Model-Driven Development.” *IEEE Software 20*, 5 (September/October 2003): 19–25.

Thomas, Dave. “MDA: Revenge of the Modelers or UML Utopia?” *IEEE Software 21*, 3 (May/June 2004): 15–17.

Wallnau, Kurt. *Volume III: A Technology for Predictable Assembly from Certifiable Components* (CMU/SEI-2003-TR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

Wallnau, Kurt & Ivers, James. *Snapshot of CCL: A Language for Predictable Assembly* (CMU/SEI-2003-TN-025). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

Weis,Torben; Ulbrich, Andreas; & Geihs, Kurt. “Model Metamorphosis.” *IEEE Software 20*, 5 (September/October 2003): 46–51.

## Service-Oriented Architecture

Barry, D. Introduction to Web Services and Service-Oriented Architectures <www.service-architecture.com/> (2004).

Barry, D. *Web Services and Service-Oriented Architectures: The Savvy Managers Guide*, Morgan Kaufmann, 2003.

Brown, A; Johnston, S.; & Kelly, K. “Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications.” Rational Software Corporation <www-106.ibm.com/developerworks/rational/library/510.html> (2002).

Joint Vision 2020. <www.dtic.mil/jointvision/jvpub2.htm>.

Koch, Christopher. “The Battle for Web Services.” *CIO Magazine*, Oct. 1, 2003.

UDDI Version 3.0 <www.uddi.org/> (July 2002).

Ogbuji, Uche. “The Past, Present and Future of Web Services.” WebServices.Org, 07.10.2002 <http://www.mywebservices.org/index.php/article/view/663/>.

Paolucci, M. & Sycara, K. “Autonomous Semantic Web Services.” IEEE Internet Computing, September/October 2003.

Papazoglou, M.P. & Georgakopoulos, D. “Service-Oriented Computing: Introduction.” *Communications of the ACM 46*, 10 (2003): 24-28.

## Web Services

W3C. “SOAP Version 1.2 Part 0: Primer.” W3C Recommendation 24 June 2003 <www.w3.org/TR/soap12-part0/>.

W3C. “SOAP Version 1.2 Part 1: Messaging Framework.” W3C Recommendation 24 June 2003 <www.w3.org/TR/soap12-part1/>.

W3C. “Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.” W3C Working Draft 26 March 2004 <www.w3.org/TR/wsdl20/>.

W3C. “Working Group Note—Web Services Glossary” <www.w3.org/TR/ws-gloss/> (February 2004).

## Q Methodology

McKeown, B.F. & Thomas, D.B. *Q Methodology*. Newbury Park, CA: Sage, 1988.

Steelman, T.A. & Maguire L.A. “Understanding Participant Perspectives: Q Methodology in National Forest Management,” <www.nicholas.duke.edu/faculty/maguire/env316/q5.htm> (2003).

Stephenson, W. The study of behavior: Q-technique and its methodology. Chicago: University of Chicago Press, 1953.

Valenta, A.L. & Wigger, U. (1997). “Q-methodology: Definition and Application in Health Care Informatics.” *Journal of the American Medical Informatics Association 4*, 6 (1997): 501–510.

## Aspect-Oriented Software Development

AspectJ <www.aspectj.org> (2004).

Bachmann, Felix; Bass, Len & Klein, Mark. *Deriving Architectural Tactics: A Step Towards Methodical Architectural Design* (CMU/SEI-2003-TR-004). Pittsburgh, PA: Software Engi-

neering Institute, Carnegie Mellon University <www.sei.cmu.edu/publications/documents /03.reports/03tr004.html> (2003).

Bass, Len; Clements, Paul; & Kazman, Rick. *Software Architecture in Practice, Second Edition*. Boston, MA: Addison-Wesley, 2003.

Bonér, Jonas. "What are the key issues for commercial AOP use—how does AspectWerkz address them?" *Proceedings of the Third International Conference on Aspect-Oriented Software Development* (AOSD2004): 5–6. Lancaster, UK. ACM, March 2004.

Chastek, Gary; Donohoe, Patrick; Kang, Kyo Chul; & Thiel, Steffan. *Product Line Analysis: A Practical Introduction* (CMU/SEI-2001-TR-001, ADA396137). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <http://www.sei.cmu.edu /publications/documents/01.reports/01tr001.html> (2001).

Chastek, Gary & McGregor, John. *Guidelines for Developing a Product Line Production Plan* (CMU/SEI-2002-TR-006, ADA407772). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <http://www.sei.cmu.edu /publications/documents /02.reports/02tr006.html> (2002).

Chastek, Gary; Donohoe, Patrick & McGregor, John. *Product Line Production Planning for the Home Integration System* (CMU/SEI-2002-TN-029). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <http://www.sei.cmu.edu /publications /documents/02.reports/02tn029.html> (2002).

Chastek, Gary; Donohoe, Patrick; Kang, Kyo Chul; & Thiel, Steffan. *Product Line Analysis for Practitioners* (CMU/SEI-2003-TR-008). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <http://www.sei.cmu.edu/publications/documents/03.reports /03tr008.html> (2003).

Chastek, Gary; Donohoe, Patrick & McGregor, John. *A Study of Product Production in Software Product Lines* (CMU/SEI-2004-TN-012). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <http://www.sei.cmu.edu /publications/documents /04.reports/04tn012.html> (2004).

Chastek, Gary & McGregor, John. "Early Product Production Aspects in Software Product Lines." Submitted to Early Aspects Workshop, Object-Oriented Programming, Systems, Languages, & Applications (OOPSLA 2004), 2004.

Clements, Paul & Northrop, Linda. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.

Clements, Paul & Northrop, Linda. *A Framework for Software Product Line Practice*, Version 4.1. <http://www.sei.cmu.edu/plp /framework.html> (2003).

Concern Manipulation Environment Project <www.research.ibm.com/cme/>.

Cockburn, Alistair. "Goals and Use Cases." *Journal of Object-Oriented Programming 10*, 5 (September 1997): 35–40.

Colyer, Adrian & Clement, Andrew. "Large-Scale AOSD for Middleware." *Conference Proceedings, 3rd International Conference on Aspect-Oriented Software Development* (AOSD2004): 56–65. Lancaster, UK. ACM, March 2004.

The Eclipse Project <www.eclipse.org/>.

Griss, Martin L. "Implementing Product Line Features by Composing Component Aspects." *Proceedings of the First Software Product Lines Conference (SPLC1)*, 2000.

IBM <www-306.ibm.com/software/info1/websphere/index.jsp> (2004).

Jacobson, Ivar. "Four Macro Trends in Software Development Y2004." Available under "recent presentations" at <www.ivarjacobson.com>.

Jboss <www.jboss.org> (2004).

Kang, Kyo C.; Cohen, Sholom G.; Hess, James A.; Novak, William E.; & Peterson, A. Spencer. *Feature-Oriented Domain Analysis Feasibility Study* (CMU/SEI-90-TR-021, ADA235785). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <http://www.sei.cmu.edu/publications/documents/90.reports /90.tr.021.html> (1990).

Kiczales, Gregor; Lamping, John; Mendhekar, Anurag; Maeda, Chris; Lopes, Cristina; Loingtier, Jean-Marc; & Irwin, John. "Aspect-Oriented Programming." *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Finland. Springer-Verlag LNCS 1241 (June 1997).

Kiczales, Gregor. “An Overview of AspectJ.” *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, Budapest, Hungary. Springer-Verlag LNCS 2072 (June 2001).

Kiczales, Gregor; Hilsdale, Erik; Hugunin, Jim; Kersten, Mik; Palm, Jeffrey; & Griswold, William G. “Getting Started with AspectJ.” *Communications of the ACM 44*, 10 (October 2001): 59–65.

Lieberherr, Karl; Lorenz, David H.; and Wu, Pengchecg. “A case for statically executable advice: checking the law of Demeter with AspectJ,” *Proceedings of the Second International Conference on Aspect-Oriented Software Development*. Boston, Mass. ACM, 2003.

Northrop, Linda M. Chapter 6, “Object-Oriented Development,” 148-159. *Software Engineering, Volume 1: The Development Process, Second Edition*. Dorfman, M. & Thayer, R.H., eds. Los Alamitos, CA: IEEE Computer Society, 1997.

Object Management Group. Unified Modeling Language <www.omg.org>.

Ottinger, Tim <c2.com/cgi/wiki?CodeNormalization> (2004).

Parnas, D.L. “On the Criteria to be Used in Decomposing Systems into Modules.” *Communications of the ACM 15*, 12 (December 1972): 1053–1058.

Sabbah, Daniel. “Aspects—from Promise to Reality.” *Conference Proceedings, Third International Conference on Aspect-Oriented Software Development (AOSD2004)*: 1–2. Lancaster, UK. ACM, March 2004.

Spring Framework <www.springframework.org> (2004).

Tarr, Peri; Ossher, Harold; Harrison, William; & Sutton, Stanley M. Jr. “N Degrees of Separation: Multi-Dimensional Separation of Concerns.” *Proceedings of the International Conference on Software Engineering (ICSE)*, Los Angeles, California, May 1999.

Yan, Hong; Aldrich, Jonathan; Garlan, David; Kazman, Rick; & Schmerl, Bradley. *Discovering Architectures from Running Systems: Lessons Learned* (CMU/SEI-2004-TR016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <http://www.sei.cmu.edu/publications/documents/04.reports/04tr016.html> (2004).

## Generative Programming

Abrahams, D. “Generic Programming Techniques” <www.boost.org/more/generic_programming.html> (18 August 2004).

Batory, D.; Sarvela, J.; & Rauschmayer, A. “Scaling Step-Wise Refinement.” *Proceedings of the International Conference on Software Engineering*, Portland, Oregon, 2003.

Chastek, G.; Donohoe, P.; & McGregor, J. D. *A Study of Product Production in Software Product Lines* (CMU/SEI-2004-TN-012). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University (2004).

Clement, P. & Northrop, L. *Software Product Lines: Practices and Patterns*. Boston, MA: Addison-Wesley, 2002.

Czarnecki, K.; Eisenecker, U.; Glück, R.; Vandevoorde, D.; & Veldhuizen, T. “Generative Programming and Active Libraries (Extended Abstract)” <osl.iu.edu/~tveldhui/papers /dagstuhl1998/>.

Czarnecki, K. & Eisenecker, U. *Generative Programming: Methods, Tools, and Applications.* Boston, MA: Addison-Wesley, 2000.

Czarnecki, K. “Generative Programming: Methods, Techniques, and Applications: Tutorial Abstract.” *Software Reuse: Methods, Techniques, and Tools: Seventh International Conference*, ICSR-7, Austin, TX, April 15–19, 2002: 351.

Greenfield, J. & Short, K. “Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.” OOPSLA ’03, Anaheim, CA. ACM, 2003.

Harrington, J. *Code Generation in Action.* Greenwich, CT: Manning, 2003.

IBM. “Autonomic Computing” <www-306.ibm.com/autonomic/index.shtml> (18 August 2004).

IEEE Computer Society. IEEE standard computer dictionary: a compilation of IEEE standard computer glossaries. Standards Coordinating Committee of the IEEE Computer Society.

Jones, N.D.; Gomard, C.K.; & Sestoft, P. *Partial Evaluation and Automatic Program Generation*. Englewood Cliffs, NJ: Prentice Hall, 1993.

"Metaprogramming." *Wikipedia.* 2004 <en.wikipedia.org/wiki/Metaprogramming> (18 August 2004).

Software Engineering Institute. "Software Product Lines." <www.sei.cmu.edu/plp/product_line_overview.html> (18 August 2004).

TenFold Corporation. "Tsunami" <tsunami.tenfold.com/> (18 August 2004).

Tristam, Claire. "Everyone's a Programmer." *Technology Review*. November 2003: 34–43.

Weiss, D. & Lai, C. *Software Product Line Engineering: A Family Based Software Development Process.* Reading, MA: Addison Wesley, 1999.

## Applying Statistics in Software Engineering

El Emam, Khaled & Carleton, Anita D., eds. "Applications of Statistics in Software Engineering" (special issue). *The Journal of Systems and Software 73*, 2 (October 2004). Amsterdam: Elsevier Publishing.

Florac, William & Carleton, Anita. *Measuring the Software Process: Statistical Process Control for Software Process Improvement*. Reading, MA: Addison-Wesley, 1999.

# Appendix B: OAR Activities

The following are activities of the SEI Options Analysis for Reengineering (OAR) method.

## Establish Mining Context

Through the Establish Mining Context (EMC) activity, OAR users establishe an understanding of the organization's product line or new single system needs, legacy base, and expectations for mining legacy components. They develop a baseline of the goals and expectations for the mining project and the component needs that mining is to address. They also determine the programmatic and technical drivers for making decisions, and select a set of potential candidate components for mining. These candidate components are analyzed and evaluated in later OAR activities based on the drivers that are elicited in the EMC activity.

## Inventory Components

In this activity, users identify the legacy system components that can potentially be mined for use as product line components. In this activity, users identify the characteristics of the product line component needs. Legacy components are evaluated based on these criteria and those that do not meet the criteria are screened out.

## Analyze Candidate Components

In this activity, users analyze the candidate set of legacy components to evaluate their potential for use as product line or new single system components. Users perform additional screening on the candidate components and identify for each candidate component the types of changes that are required to mine them.

## Plan Mining Options

In this activity, alternative options for mining are developed, based on schedule, cost, effort, risk, and resource considerations. Users perform a final screening of candidate components and analyze the impacts of different aggregations of components.

## Select Mining Option

In this activity users select the mining option or combination of options that can best satisfy the organization's goals by balancing programmatic and technical considerations. Each mining option is evaluated and the optimal option or combination of options is selected. A summary report and justification for the selected option are prepared.

# References

*URLs are valid as of the publication date of this document.*


**[Aldrich 02]** Aldrich, J.; Chambers, C.; & Notkin, D. “ArchJava: Connecting Software Architecture to Implementation.” *Proceedings of the International Conference on Software Engineering*, May 2002.

**[Bass 04]** Bass, Len; Klein, Mark; & Northrop, Linda. “Identifying Aspects Using Architectural Reasoning,” Early Aspects Workshop, Third International Conference on Aspect-Oriented Software Development (AOSD2004). Lancaster, UK, March 2004. (unpublished).

**[Bergey 01]** Bergey, John; O’Brien, Liam; & Smith, Dennis. *Options Analysis for Reengineering (OAR): A Method for Mining Legacy Assets* (CMU/SEI-2001-TN-013). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.

**[Bergey 02]** Bergey, J.; O'Brien, L.; & Smith, D. “Using Options Analysis for Reengineering (OAR) for Mining Components for a Product Line,” 316-327. *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2)*. San Diego, CA, August 19-22, 2002. Berlin, Germany: Springer Lecture Notes in Computer Science Vol. 2379, 2002.

**[Boehm 88]** Boehm, Barry W. & Papaccio, Philip N. “Understanding and Controlling Software Costs,” *IEEE Transactions on Software Engineering 14*, 10 (October 1988): 1462-1477.

**[Brown 02]** Brown, A; Johnston, S.; & Kelly, K. Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications. Rational Software Corporation. 2002. Available WWW: <http://www-106.ibm.com/developerworks/rational/library/510.html>.

**[Chastek 04]** Chastek, G.; Donohoe, P. & McGregor, J.D. A Study of Product

Production in Software Product Lines (CMU/SEI-2004-TN-012). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA: 2004.

**[Clements 01]** Clements, P. & Northrop, L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

**[Czarnecki 00]** Czarnecki, K. & Eisenecker, U. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston: 2000.

**[De Lucia 97]** De Lucia, A.; Di Lucca, G. A.; Fasolino, A. R.; Guerra, P.; & Petruzzelli, S. "Migrating legacy systems towards object-oriented platforms." *International Conference on Software Maintenance* (1997): 122–129.

**[DSB 2000]** *Defense Science Board Task Force: Report on Defense Software*. <www.acq.osd.mil/dsb/defensesoftware.pdf> (November, 2000).

**[Facemire 03]** Facemire, J. & Silva, H. "Experiences with Leveraging Six Sigma to Implement CMMI Levels 4 and 5." *Proceedings of NDIA and CMMI Users Group*, November 2003.

**[Florac 99]** Florac, William & Carleton, Anita. *Measuring the Software Process: Statistical Process Control for Software Process Improvement*. Reading, MA: Addison-Wesley, 1999.

**[Foster 02a]** Foster, I.; Kesselman, C.; Nick, J. & Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. June 2002. Available WWW: <http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf>.

**[GGF 03]** Global Grid Forum—Open Grid Services Infrastructure Working Group. Open Grid Services Infrastructure (OGSI) Version 1.0. June 2003. Available WWW: <http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf>.

**[Globus 04]** The Globus Alliance. http://www.globus.org/ogsa/

**[Harrington 03]** Harrington, J. Code Generation in Action. Manning, Greenwich,

CT: 2003.

**[J2EE]** Sun Microsystems. <http://java.sun.com/docs/books/j2eetutorial/index.html>. URL valid as of September 2004.

**[Kazman 99]** Kazman, R. & Carriere, S.J. “Playing Detective: Reconstructing Software Architecture from Available Evidence.” *Journal of Automated Software Engineering 6*, 2 (1999).

**[Khurana 03]** Khurana, H.; Gavrila, S; Bobba, R.; Koleva, R.; Sonalker A.; Dinu, E.; Gligor, V. & Baras, J. “Integrated Security Services for Dynamic Coalitions.” Proceedings of the DARPA Information Survivability Conference and Exposition. 2003.

**[Lipson 99]** Lipson, Howard F. & Fisher, David A. “Survivability—A New Technical and Business Perspective on Security.” *Proceedings of the 1999 New Security Paradigms Workshop*, September 21–24, 1999, Caledon Hills, ON. Association for Computing Machinery, New York, NY <www.cert.org /archive/pdf/busperspec.pdf>.

**[Lipson 02]** Lipson, Howard. *Tracking and Tracing Cyber-Attacks: Technical Challenges and Global Policy Issues* (CMU/SEI-2002-SR-009). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <www.cert.org/archive/pdf/02sr009.pdf>, 2002.

**[McKeown 88]** McKeown, B.F. & Thomas, D.B. *Q Methodology*. Newbury Park, CA: Sage, 1988.

**[McMaster 04]** http://www.cas.mcmaster.ca/serg/

**[Murphy 95]** Murphy, G.C.; Notkin, D.; & Sullivan, K.J. “Software Reflexion Models: Bridging the Gap Between Source and High-Level Models.” *Proceedings of FSE 1995* (1995).

**[McGraw 02]** McGraw, G. “Software Security.” *IEEE Security & Privacy 2*, 2 (March/April 2004).

**[Northrop 97]** Northrop, Linda M. Chapter 6, “Object-Oriented Development,” 148-159. Software Engineering, Volume 1: The Development Process, Second Edition. Dorfman, M & Thayer, R. H., eds. Los Alami-

tos, CA: IEEE Computer Society, 1997.

**[OASIS 02]** UDDI Version 3.0. July 2002. Available WWW: <http://www.uddi.org/>.

**[OMG 03]** OMG. MDA Guide Version 1.0.1. Available WWW: <http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf>.

**[Ottinger 04]** Ottinger, Tim. c2.com/cgi/wiki?CodeNormalization, 2004.

**[Parnas 72]** Parnas, D.L. "On the Criteria to be Used in Decomposing Systems into Modules." Communications of the ACM 15, 12 (December 1972): 1053–1058.

**[Sandholm 03]** Sandholm, T. & Gawor, J. Globus Toolkit 3 Core—A Grid Service Container Framework. July 2003. Available WWW: <http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf>.

**[Shaw 95]** Shaw, M.; Deline, R.; Klein, D.; Ross, T.L.; Young, D.M.; & Zelesnik, G. "Abstractions for Software Architecture and Tools to Support Them." *IEEE Transactions on Software Engineering 21*, 4 (1995).

**[Sneed 98]** Sneed, H. & Majnar, R. "A Case Study in Software Wrapping." *International Conference on Software Maintenance* (1998): 86-93.

**[Sneed 99]** Sneed, H. "Risks Involved in Reengineering Projects." *Sixth Working Conference on Reverse Engineering* (1999): 204–211.

**[Steelman 03]** Steelman, T.A. & Maguire L.A. "Understanding Participant Perspectives: Q Methodology in National Forest Management," <www.nicholas.duke.edu/faculty/maguire/env316/q5.htm> (2003).

**[Sweeney 02]** Sweeney, Latanya. "*k*-anonymity: A Model for Protecting Privacy." *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems* 10, 5 (2002): 557-570.

**[Taylor 96]** Taylor, R.N.; Medvidovic, N.; Anderson, K.M.; Whitehead, E.J.; Robbins, J.E.; Nies, A.; Oriezy, P.; & Dubrow, D. "A Component- and Message-Based Architectural Style for GUI Software." *IEEE*

*Transactions on Software Engineering 22*, 6 (1996).

**[Valenta 97]** Valenta, A.L. & Wigger, U. (1997). "Q-methodology: Definition and Application in Health Care Informatics." *Journal of the American Medical Informatics Association 4*, 6 (1997): 501–510.

**[Vestal 96]** Vestal, S. *MetaH Programmer's Manual, Version 1.09.* Technical Report, Honeywell Technology Center, 1996.

**[W3C 03-0]** W3C. SOAP Version 1.2 Part 0: Primer. W3C Recommendation. 24 June 2003. Available WWW: <http://www.w3.org/TR/soap12-part0/>.

**[W3C 03-1]** W3C. SOAP Version 1.2 Part 1: Messaging Framework. W3C Recommendation. 24 June 2003. Available WWW: <http://www.w3.org/TR/soap12-part1/>.

**[W3C 04]** W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Working Draft. 26 March 2004. Available WWW: <http://www.w3.org/TR/wsdl20/>.

**[Willis 98]** Willis, Ronald R., et al. *Hughes Aircraft's Widespread Deployment of a Continuously Improving Software Process* (CMU/SEI-98-TR-006). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998.

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

**[Yan 04b]** Yan, H.; Aldrich, J.; Garlan, D.; Kazman, R.; & Schmerl, B. *Discovering Architectures from Running Systems: Lessons Learned* (CMU/SEI-2004-TR-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| (Leave Blank) | October 2004 | Final |

| 4. TITLE AND SUBTITLE | 5. FUNDING NUMBERS |
|---|---|
| Results of SEI Independent Research and Development Projects and Report on Emerging Technologies and Technology Trends | F19628-00-C-0003 |

6. AUTHOR(S)

John Bergey, Sven Dietrich, Donald Firesmith, et al.

| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) | 8. PERFORMING ORGANIZATION REPORT NUMBER |
|---|---|
| Software Engineering Institute<br>Carnegie Mellon University<br>Pittsburgh, PA 15213 | CMU/SEI-2004-TR-018 |

| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER |
|---|---|
| HQ ESC/XPK<br>5 Eglin Street<br>Hanscom AFB, MA 01731-2116 | ESC-TR-2004-018 |

11. SUPPLEMENTARY NOTES

| 12A DISTRIBUTION/AVAILABILITY STATEMENT | 12B DISTRIBUTION CODE |
|---|---|
| Unclassified/Unlimited, DTIC, NTIS | 102 |

13. ABSTRACT (MAXIMUM 200 WORDS)

Each year, the Software Engineering Institute (SEI) undertakes several Independent Research and Development (IR&D) projects. These projects serve to (1) support feasibility studies investigating whether further work by the SEI would be of potential benefit, and (2) support further exploratory work to determine whether there is sufficient value in eventually funding the feasibility study work as an SEI initiative. Projects are chosen based on their potential to mature and/or transition software engineering practices, develop information that will help in deciding whether further work is worth funding, and set new directions for SEI work. This report describes the IR&D projects that were conducted during fiscal year 2004 (October 2003 through September 2004). In addition, this report provides information on what the SEI has learned in its role as a technology scout for developments over the past year in the field of software engineering.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| software engineering research and development | 107 |

16. PRICE CODE

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |